Can Language Models Serve as Text-Based World Simulators?

Ruoyao Wang[†], **Graham Todd**[‡], **Ziang Xiao**[♠], **Xingdi Yuan**[♢]
Marc-Alexandre Côté[♢], **Peter Clark**[♣], **Peter Jansen**[†♣]
[†]University of Arizona [♢]Microsoft Research Montréal
[‡]New York University [♠]Johns Hopkins University [♣]Allen Institute for AI
{ruoyaowang,pajansen}@arizona.edu gdrtodd@nyu.edu
ziang.xiao@jhu.edu {eric.yuan,macote}@microsoft.com
PeterC@allenai.org

Abstract

Virtual environments play a key role in benchmarking advances in complex planning and decision-making tasks but are expensive and complicated to build by hand. Can current language models themselves serve as world simulators, correctly predicting how actions change different world states, thus bypassing the need for extensive manual coding? Our goal is to answer this question in the context of text-based simulators. Our approach is to build and use a new benchmark, called BYTE-SIZED32-State-Prediction, containing a dataset of text game state transitions and accompanying game tasks. We use this to directly quantify, for the first time, how well LLMs can serve as text-based world simulators. We test GPT-4 on this dataset and find that, despite its impressive performance, it is still an unreliable world simulator without further innovations. This work thus contributes both new insights into current LLM's capabilities and weaknesses, as well as a novel benchmark to track future progress as new models appear.

1 Introduction and Related Work

Simulating the world is crucial for studying and understanding it. In many cases, however, the breadth and depth of available simulations are limited by the fact that their implementation requires extensive work from a team of human experts over weeks or months. Recent advances in large language models (LLMs) have pointed towards an alternate approach by leveraging the huge amount of knowledge contained in their pre-training datasets. But are they ready to be used directly as simulators?

We examine this question in the domain of text-based games, which naturally express the environment and its dynamics in natural language and have long been used as part of advances in decision making processes (Côté et al., 2018; Fan et al., 2020; Urbanek et al., 2019; Shridhar et al., 2020; Hausknecht et al., 2020; Jansen, 2022; Wang et al., 2023), information extraction (Ammanabrolu and Hausknecht, 2020; Adhikari et al., 2020), and artificial reasoning (Wang et al., 2022).

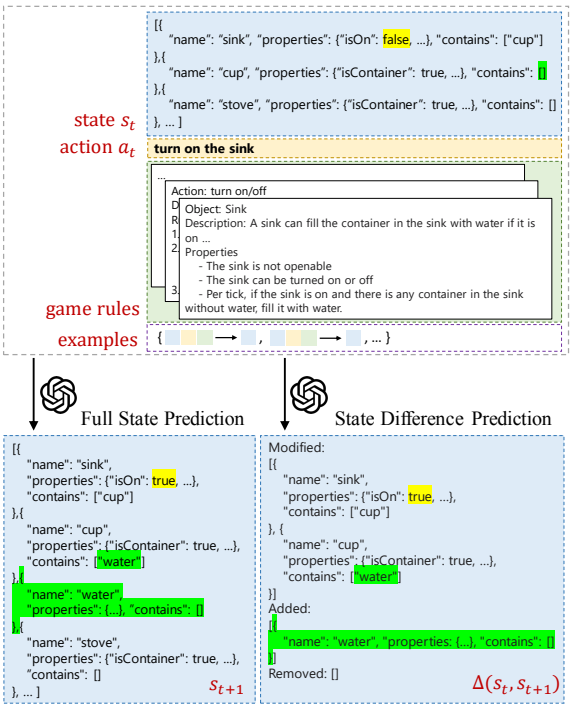


Figure 1: An overview of our two approaches using an LLM as a text game simulator. The example shows the process that a cup in the sink is filled by water after turning on the sink. The full state prediction includes all objects in the game including the unrelated stove, while the state difference prediction excludes the unrelated stove. State changes caused by $\mathcal{F}_{\text{act}}$ and $\mathcal{F}_{\text{env}}$ are highlighted in yellow and green, respectively.

Broadly speaking, there are two ways to leverage LLMs in the context of world modeling and simulation. The first is *neurosymbolic*: a number of efforts use language models to generate code in a symbolic representation that allows for formal planning or inference (Liu et al., 2023; Nottingham et al., 2023; Wong et al., 2023; Tang et al., 2024). REASONING VIA PLANNING (RAP) (Hao et al., 2023) is one such approach – it constructs a world model using LLM priors and then uses a

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 1–17

dedicated planning algorithm to decide on agent policies (LLMs themselves continue to struggle to act directly as planners (Valmeekam et al., 2023)). Similarly, BYTESIZED32 (Wang et al., 2023) tasks LLMs with instantiating simulations of scientific reasoning concepts in the form of large PYTHON programs. These efforts are in contrast to the second, and comparatively less studied, approach of *direct simulation*. For instance, AI-DUNGEON represents a game world purely through the generated output of a language model, with inconsistent results (Walton, 2020). In this work, we provide the first quantitative analysis of the abilities of LLMs to directly simulate virtual environments. We make use of *structured representations* in the JSON schema as a scaffold that both improves simulation accuracy and allows for us to directly probe the LLM's abilities across a variety of conditions.

In a systematic analysis of GPT-4 (Achiam et al., 2023), we find that LLMs broadly fail to capture state transitions not directly related to agent actions, as well as transitions that require arithmetic, common-sense, or scientific reasoning. Across a variety of conditions, model accuracy does not exceed 59.9% for transitions in which a non-trivial change in the world state occurs. These results suggest that, while promising and useful for downstream tasks, LLMs are not yet ready to act as reliable world simulators without further innovation.[1]

2 Methodology

We examine the abilities of LLMs to serve as world simulators in text-based virtual environments, in which an agent receives observations and proposes actions in natural language in order to complete certain objectives. Each text environment can be formally represented as a goal-conditioned partially observable Markov decision process (POMDP) (Kaelbling et al., 1998) with the 7-tuple $(S, A, \mathcal{T}, O, R, C, D)$, where S denotes the state space, A denotes the action space, $\mathcal{T} : S \times A \to S$ denotes the transition function, O denotes the observation function, $R : S \times A \to \mathbb{R}$ denotes the reward function, C denotes a natural language "context message" that describes the goal and action semantics, and $D : S \times A \to \{0, 1\}$ denotes the binary completion indicator function.

[1]Code and data are available at https://github.com/cognitiveailab/GPT-simulator.

States (avg. per game)	2463.5
Action verbs (avg. per game)	7.4
Object types (avg. per game)	5.5
Object instances (avg. per state)	10.4
Total games	31
Total transitions	76,369

Table 1: Corpus statistics of BYTESIZED32-SP.

2.1 LLM-Sim Task

We propose a prediction task, which we call LLM-as-a-Simulator (LLM-Sim), as a way of quantitatively evaluating the capacity of language models to serve as reliable simulators. The LLM-Sim task is defined as implementing a function $\mathcal{F} : C \times S \times A \to S \times \mathbb{R} \times \{0, 1\}$ as a world simulator that maps from a given context, state, and action (i.e. c, s_t, a_t) to the subsequent state, reward, and game completion status (i.e. $s_{t+1}, r_{t+1}, d_{t+1}$).

In practice, the whole state transition simulator $\mathcal{F}$ should consider two types of state transitions: action-driven transitions and environment-driven transitions. For the example in Figure 1, the action-driven transition is that the sink is turned on (`isOn=true`) after taking the action *turn on sink*, and the environment-driven transition is that water fills up the cup in the sink when the sink is on. To better understand LLM's ability to model each of these transitions, we further decompose the simulator function $\mathcal{F}$ into three steps:

$$s^{\text{act}}_{t+1} = \mathcal{F}_{\text{act}}(c, s_t, a_t)$$
$$s_{t+1} = \mathcal{F}_{\text{env}}(c, s^{\text{act}}_{t+1})$$
$$r_{t+1}, d_{t+1} = \mathcal{F}_R(c, a_t, s_{t+1})$$

1. **Action-driven transition simulator** $\mathcal{F}_{\text{act}} : C \times S \times A \to S$ predicts s^{act}_{t+1} given c, s_t, and a_t, where s^{act}_{t+1} represents the direct state change caused by actions.

2. **Environment-driven transition simulator** $\mathcal{F}_{\text{env}} : C \times S \to S$ predicts s_{t+1} given c and s^{act}_{t+1}, where s_{t+1} is the state that results after any environment-driven transitions.

3. **Game progress simulator** $\mathcal{F}_R : C \times S \times A \to \mathbb{R} \times \{0, 1\}$ predicts the reward r_{t+1} and the game completion status d_{t+1} given c, s_{t+1}, and a_t.

In our experiments, we measure the ability for LLMs to model $\mathcal{F}_{\text{act}}$, $\mathcal{F}_{\text{env}}$, and $\mathcal{F}_R$ separately, as well as the complete $\mathcal{F}$ (i.e. in which all transitions are captured in a single step). We consider two variants of the LLM-Sim task:

Full State Prediction: The LLM outputs the complete state. For example, when functioning as $\mathcal{F}$, given c, s_t and a_t, the model generates the full game state s_{t+1} alongside r_{t+1} and d_{t+1}.

State Difference Prediction: The LLM outputs only the difference between the input and output states. For example, when functioning as $\mathcal{F}$, given c, s_t and a_t, the model generates only the difference between the current and subsequent game states, $\Delta((s_t, r_t, d_t), (s_{t+1}, r_{t+1}, d_{t+1}))$, as a way to reduce the need to generate redundant or unchanging information. We do not apply state difference prediction to the game progress simulator $\mathcal{F}_R$ as its output (r_{t+1} and d_{t+1}) is not complex.

2.2 Data

To facilitate evaluation on the LLM-Sim task, we introduce a novel dataset of text game state transitions. Our dataset, BYTESIZED32-State-Prediction (BYTESIZED32-SP), consists of 76,369 transitions represented as $(c, s_t, r_t, d_t, a_t, s^{\text{act}}_{t+1}, s_{t+1}, r_{t+1}, d_{t+1})$ tuples collected from 31 distinct text games. Additional corpus statistics are summarized in Table 1.

Data Collection: Our dataset is derived from the open BYTESIZED32 corpus (Wang et al., 2023), which consists of 32 human-authored text games that each simulate a different scientific or commonsense reasoning concept. We first modify each BYTESIZED32 game to dump the game state (s_t, r_t, d_t) as well as its intermediate state s^{act}_{t+1} at each time step t as a JSON object. We hold out one game as an example and seed our dataset of transitions by first following the gold-label goal-following trajectory provided with each game. We then deterministically collect every valid transition that is at most one step away from the gold-label trajectory by querying the game for the set of valid actions at each step.

Additional Context: Each game also includes a context message, c, that provides additional information to the model. The context consists of four parts: *action rules* describing the effect of each action on the game state, *object rules* describing the meaning of each object property and whether they are affected by the game's underlying dynamics, *scoring rules* describing how an agent earns reward and the conditions under which the game is won or lost, and one or two *example transitions* (see Appendix B for details) from the held-out game mentioned above. For each game we generate three versions of the context, one where the rules are written by a human expert (one of the game authors), and one where they are produced by an LLM with access to the game code, and one where no rules are provided. See Appendix C for additional details.

Rules	State Change	$\mathcal{F}$ Full	$\mathcal{F}$ Diff	$\mathcal{F}_{\text{act}}$ Full	$\mathcal{F}_{\text{act}}$ Diff	$\mathcal{F}_{\text{env}}$ Full	$\mathcal{F}_{\text{env}}$ Diff
LLM	*dynamic*	59.0	59.5	76.1	75.2	44.1	49.7
	static	62.8	72.2	73.0	89.5	61.9	93.8
Human	*dynamic*	59.9	51.6	77.1	68.4	38.6	22.2
	static	63.5	73.9	77.5	90.2	73.8	92.3
No rule	*dynamic*	54.1	52.2	70.8	67.7	24.4	22.3
	static	56.6	70.4	65.3	84.6	73.0	91.7

Table 2: Average accuracy per game of GPT-4 predicting the whole state transitions ($\mathcal{F}$) as well as action-driven transitions ($\mathcal{F}_{\text{act}}$) and environment-driven transitions ($\mathcal{F}_{\text{env}}$). We report settings that use LLM generated rules, human written rules, or no rules. Dynamic and static denote whether the game object properties and game progress should be changed; Full and diff denote whether the prediction outcome is the full game state or state differences. Numbers are shown in percentage.

Rules	**Game Progress**
LLM	92.1
Human	81.8
No rule	61.5

Table 3: GPT-4 game progress prediction results

2.3 Evaluation

Performance on LLM-Sim is determined by the model's prediction accuracy w.r.t. the ground truth labels over a dataset of test samples. Depending on the experimental condition, the LLM must model object properties (when simulating $\mathcal{F}_{\text{act}}$, $\mathcal{F}_{\text{env}}$, or $\mathcal{F}$) and / or game progress (when simulating $\mathcal{F}_R$ or $\mathcal{F}$), defined as:

Object Properties: a list of all objects in the game, along with each object's properties (e.g., temperature, size) and relationships to other objects (e.g., being within or on top of another object).

Game Progress: the status of the agent w.r.t. the overall goal, consisting of the current accumulated reward, whether the game has terminated, and whether the overall goal has been achieved.

We note that in each case the LLM is provided with the ground truth previous state (when functions as $\mathcal{F}_{\text{env}}$ the previous state is s^{act}_{t+1}) as well as the overall task context. That is to say, the LLM always performs a single-step prediction.

3 Experiments

Figure 1 demonstrates how we evaluate the performance of a model on the LLM-Sim task using

Game	Avg. Annotator	GPT-4
bath-tub-water-temperature	0.99	0.60
clean-energy	0.50	0.35
take-photo	0.83	0.00
metal-detector	0.86	0.50
mix-paint	0.85	0.50
Average	0.80	0.49

Table 4: Comparison between accuracy of human annotators and GPT-4 on a subset of the BYTESIZED32-SP dataset. Transitions were sampled to normalize GPT-4 performance at 50% (if possible) and annotators were tasked with modeling the complete transition function $\mathcal{F}$ and outputting the full state.

in-context learning. We evaluate the accuracy of GPT-4 in both the *Full State* and *State Difference* prediction regimes. The model receives the previous state (encoded as a JSON object), previous action, and context message, it produces the subsequent state (either as a complete JSON object or as a diff). See Appendix A for details.

We note that the transition dynamics between states depend primarily on the verb used in the action (e.g., *take*, *put*, *cook*, ...). In addition, some state-action pairs do not result in any changes to object properties or game progress. To ensure balance across these conditions (and increase the tractability of our experiments), we sub-sample a dataset $\mathcal{D}$ from the full BYTESIZED32-SP set. Formally, let s_{in} be the input state of a simulator function and s_{out} be the output state of the simulator function (e.g. $s_{\text{in}} = s_t$ and $s_{\text{out}} = s^{\text{act}}_{t+1}$ for $\mathcal{F}_{\text{act}}$). We call any transition in which $s_{\text{out}} = s_{\text{in}}$ (according to the ground-truth) ***static*** and call each other transition ***dynamic***. Note that the environment-driven transition following a *dynamic* action-driven transition is not necessarily *dynamic*. For example, a state in which the agent takes an apple while the remaining objects in the environment remain the same is a *dynamic* action-driven transition and a *static* environment-driven transition. We construct $\mathcal{D}$ by randomly sampling 10 *dynamic* transitions and 10 *static* transitions from BYTESIZED32-SP for each possible action verb (taking as many as possible if fewer than 10 exist) w.r.t *action-driven transitions*. The resulting experimental dataset consists of 2954 transition tuples.

4 Results

Table 2 presents the accuracy of GPT-4 simulating the whole state transitions as well as its accuracy of simulating action-driven transitions and environment-driven transitions alone.[2] We report some major observations below:

Predicting action-driven transitions is easier than predicting environment-driven transitions: At best, GPT-4 is able to simulate 77.1% of *dynamic* action-driven transitions correctly. In contrast, GPT-4 simulates at most 49.7% of *dynamic* environment-driven transitions correctly. This indicates that the most challenging part of the LLM-Sim task is likely simulating the underlying environmental dynamics.

Predicting static transitions is easier than dynamic transitions: Unsurprisingly, modeling a *static* transition is substantially easier than a *dynamic* transition across most conditions. While the LLM needs to determine whether a given initial state and action will result in a state change in either case, *dynamic* transitions *also* require simulating the dynamics in exactly the same way as the underlying game engine by leveraging the information in the context message.

Predicting full game states is easier for dynamic states, whereas predicting state difference is easier for static states: Predicting the state difference for dynamic state significantly improves the performance (>10%) of simulating *static* transitions, while decreases the performance when simulating *dynamic* transitions. This may be because state difference prediction is aimed at reducing potential format errors. However, GPT-4 is able to get the response format correct in most cases, while introducing the state difference increases the complexity of the output format of the task.

Game rules matter, and LLMs are able to generate good enough game rules: Performance of GPT-4 on all three simulation tasks drops in most conditions when game rules are not provided in the context message. However, we fail to find obvious performance differences between game rules generated by human experts and by LLMs themselves.

GPT-4 can predict game progress in most cases: Table 3 presents the results of GPT-4 predicting game progress. With game rules information in the context, GPT-4 can predict the game progress correctly in 92.1% test cases. The presence of these rules in context is crucial: without them, GPT-4's prediction accuracy drops to 61.5%.

Humans outperform GPT-4 on the LLM-Sim task: We provide a preliminary human study on the LLM-Sim task. In particular, we take the 5 games

[2] See Appendix E for the results of GPT-3.5.

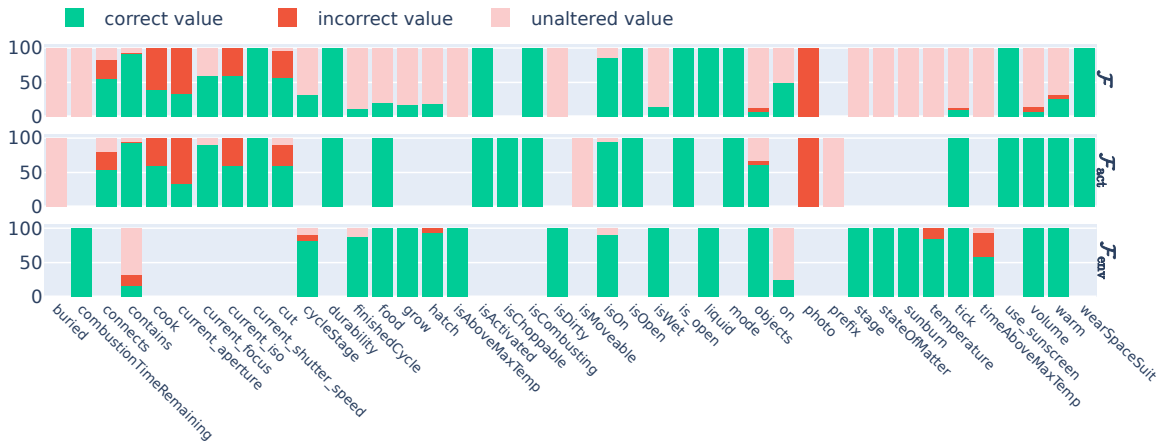


Figure 2: Simulation performance of whole state transition (top), action-driven transitions (middle) and environment-driven transitions (bottom) as a function of the property being modified, in the *GPT-4, full state prediction, with human written rules* condition. The *x-axis* represents specific object properties, and *y-axis* represents performance (0-100%). Errors are broken down into incorrect value and unaltered value. Refer to Table 7 for the meaning of each property.

from the BYTESIZED32-SP dataset in which GPT-4 produced the worst accuracy at modeling $\mathcal{F}_{\text{act}}$. For each game, we randomly sample 20 games with the aim of having 10 transitions where GPT-4 succeeded and 10 transitions where GPT-4 failed (note that this is not always possible because on some games GPT-4 fails/succeeds on most transitions). In addition, we balance each set of 10 transitions to have 5 *dynamic* transitions and 5 *static* transitions. We instruct four human annotators (4 authors of this paper) to model as $\mathcal{F}_{\text{act}}$ using the human-generated rules as context in a full game state prediction setting. Results are reported in Table 4. The overall human accuracy is 80%, compared to the sampled LLM accuracy of 50%, and the variation among annotators is small. This suggests that while our task is generally straightforward and relatively easy for humans, there is still a significant room for improvement for LLMs.

GPT-4 is more likely to make an error when arithmetic, common-sense, or scientific knowledge is needed: Because most errors occur in modeling *dynamic* transitions, we conduct an additional analysis to better understand failure modes. We use the setting with the best performance on *dynamic* transitions (GPT-4, Human-written context, full state prediction) and further break down the results according to the specific object properties that are changed during the transition. Figure 2 shows, for the whole state transitions, action-driven transitions, and environment-driven transitions, the proportion of predictions that are either correct, set the property to an incorrect value, or fail to change the property value (empty columns means the property is not changed in its corresponding condition). We observe that GPT-4 is able to handle most simple boolean value properties well. The errors are concentrated on non-trivial properties that requires arithmetic (e.g., `temperature`, `timeAboveMaxTemp`), common-sense (e.g., `current_aperture`, `current_focus`), or scientific knowledge (e.g., `on`). We also observe that when predicting the action-driven and environment-driven transitions in a single step, GPT-4 tends to focus more on action-driven transitions, resulting in more unaltered value errors on states that it can predict correctly when solely simulating environment-driven transitions.

5 Conclusion

We propose BYTESIZED32-State-Prediction, a benchmark of 76,369 virtual text environment state transitions for testing LLMs as simulators. We evaluate GPT-4 on this world modeling task. Across models and conditions, the best recorded performance is 59.9% on accurately simulating state transitions that involve non-trivial changes. Because simulation errors accumulate across steps, a simulator with modest single-step accuracy has limited utility in practice – for example, after 10 steps, average simulation accuracy would reduce to 0.599^{10}, or less than 1%. Our results indicate that **LLMs are not yet able to reliably act as text world simulators**. Further error analysis shows that while LLMs are better at simulating the results of user actions, it is difficult for LLMs to handle environment-driven transitions and transitions that require arithmetic, common sense, or scientific knowledge.

6 Limitations and Ethical Concerns

6.1 Limitations

This work considers two strong in-context learning LLMs, GPT-3.5 and GPT-4, in their ability to act as explicit formal simulators.We adopt these models because they are generally the most performant off-the-shelf models across a variety of benchmarks. While we observe that even GPT-3.5 and GPT-4 achieve a modest score at the proposed task, we acknowledge that we did not exhaustively evaluate a large selection of large language models, and other models may perform better. We provide this work as a benchmark to evaluate the performance of existing and future models on the task of accurately simulating state space transitions.

In this work, we propose two representational formalisms for representing state spaces, one that includes full state space, while the other focuses on state difference, both represented using JSON objects. We have chosen these representations based on their popularity and compatibility with the input and output formats of most LLM pretraining data (e.g. Fakhoury et al., 2023), as well as being able to directly compare against gold standard simulator output for evaluation, though it is possible that other representational formats may be more performant at the simulation task.

Finally, the state spaces produced in this work are focused around the domain of common-sense and early (elementary) scientific reasoning. These tasks, such as opening containers or activating devices, were chosen because the results of these actions are common knowledge, and models are likely to be most performant in simulating these actions. While this work does address a selection of less frequent actions and properties, it does not address using LLMs as simulators for highly domain-specific areas, such as physical or medical simulation. A long term goal of this work is to facilitate using language models as simulators for high-impact domains, and we view this work as a stepping-stone to developing progressively more capable language model simulators.

6.2 Ethical Concerns

We do not foresee an immediate ethical or societal impact resulting from our work. However, we acknowledge that as an LLM application, the proposed LLM-Sim task could be affected in some way by misinformation and hallucinations introduced by the specific LLM selected by the user. Our work highlights the issue with using LLMs as text-based world simulators. In downstream tasks, such as game simulation, LLMs may generate misleading or non-factual information. For example, if the simulator suggests burning a house to boil water, our work does not prevent this, nor do we evaluate the ethical implications of such potentially dangerous suggestions. As a result, we believe such applications are neither suitable nor safe to be deployed to a setting where they directly interact with humans, especially children, e.g., in an educational setting. We urge researchers and practitioners to use our proposed task and dataset in a mindful manner.

Acknowledgements

We wish to thank the three anonymous reviewers for their helpful comments on an earlier draft of this paper.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. 2020. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33:3045–3057.

Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. *arXiv preprint arXiv:2001.08837*.

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. 2018. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Sarah Fakhoury, Saikat Chakraborty, Madan Musuvathi, and Shuvendu K Lahiri. 2023. Towards generating functionally correct code edits from natural language issue descriptions. *arXiv preprint arXiv:2304.03816*.

Angela Fan, Jack Urbanek, Pratik Ringshia, Emily Dinan, Emma Qian, Siddharth Karamcheti, Shrimai Prabhumoye, Douwe Kiela, Tim Rocktaschel, Arthur Szlam, and Jason Weston. 2020. Generating interactive worlds with text. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1693–1700.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910.

Peter Jansen. 2022. A systematic survey of text worlds as embodied natural language environments. In *Proceedings of the 3rd Wordplay: When Language Meets Games Workshop (Wordplay 2022)*, pages 1–15.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. 2023. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, pages 26311–26325. PMLR.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Hao Tang, Darren Key, and Kevin Ellis. 2024. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *arXiv preprint arXiv:2402.12275*.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Nick Walton. 2020. *How we scaled AI Dungeon 2 to support over 1,000,000 users*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298.

Ruoyao Wang, Graham Todd, Xingdi Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter Jansen. 2023. ByteSized32: A corpus and challenge task for generating task-specific world models expressed as text games. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13455–13471, Singapore. Association for Computational Linguistics.

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

A Model details

For the GPT-3.5 model, we use the `gpt-3.5-turbo-0125` model. For the GPT-4 model, we use the `gpt-4-0125-preview` model. For both models, the temperature is set to 0 to get deterministic results. We also turn on the JSON mode of both models, which ensures that the model gives a valid JSON response. Our experiments cost approximately $5,000 for OpenAI API usage.

B Game transition examples

We manually pick the wash-clothes game in BYTESIZED32 as the example game as it contains both state transitions driven by actions and game's underlying dynamics. In tasks where the model predicts action transition, environment-driven transitions, or the game progress alone, we provide one corresponding in-context example. In the task that requires the model to predict everything, we offer two in-context examples in the prompt. The two examples are manually picked such that in one example the game state is changed directly by the action taken while in the other example the game state is changed by the game's underlying dynamics.

C Game rules generation

C.1 LLM generated rules

For LLM generated rules, we manually check all of them to avoid misinformation and offensive content.

We prompt GPT-4 (`gpt-4-0125-preview`) with the code of each object class to acquire the rules of each object. We also provide one in-context example. We ask GPT-4 to describe the meaning of each critical property (i.e. properties that do not inherit from parent) of the object and the tick function of the object (i.e. a function that defines how object properties may change at each time step regardless of the action taken). Below is an example of our prompt of object rule generation:

Object Rule Generation Prompt

You will be given a Python class which defines an object in a text game. List the classes inherited by this class and explain the properties of the object based on your understanding of the code. The properties you need to explain are commented as critical properties in the init function. If the class contains a tick method function, you should also decribe how the object properties will be changed at each game tick. Otherwise, do not explain any property. Your response should follow the format of the example below:
Here is the code for the example:
{**OBJECT_CLASS_CODE**}
The expected output is:
Object: Stove
Inherits: Container, Device
Properties:
maxTemperature: the maximum temperature of the stove in degrees Celsius
tempIncreasePerTick: the temperature increases per tick for objects on the stove if the stove is on.

Now here is another object class that needs you to explain:
{**OBJECT_CLASS_CODE**}

For action rules generation, we prompt GPT-4 (`gpt-4-0125-preview`) with the code of the whole game, but unlike object rules, we do not offer any in-context example. We ask GPT-4 to describe each of the actions in the game. Below is an example of our prompt for action rule generation:

Action Rule Generation Prompt

You will be given a Python program which defines an a text game. Describe the all actions based on your understanding of the code. You can find all actions listed in the comments at the beginning of the program. You should describe all constraints of each action and how game states will be changed by taking each action.
Here is the code of the game:
{**GAME_CODE**}

Similar to action rules, we generate score rules by prompting GPT-4 (`gpt-4-0125-preview`) with the code of the game and ask GPT-4 to describe how the game can be won or lose and how rewards can be earned. Below is an example of our prompt for score rule generation:

Score Rule Generation Prompt

You will be given a Python program which defines an a text game. Describe how the game can be won or lose, and how game scores can be earned based on your understanding of the calculateScore function in the TextGame class.
Here is the code of the game. Do not describe the main function.
{**GAME_CODE**}

C.2 Human-Written Action Rules

The action rules describe how each action can change the game states. The expert annotator reads the game description and source code for each game. They went through the list of available actions in the game and their corresponding functions in the game. Each action rule has three main parts: Action, Description, and Rules. The Action specifies the name of the action (e.g., action). The Description explains the general purpose of the ac-

tion (e.g., connect two objects with input terminals). The Rules is an unordered list of rule descriptions that describe the constraints of the action when interacting with different objects (e.g., At least one of the objects should be a wire or a multimeter) or how the rule might function under different conditions (e.g., Disconnect terminal if the terminal is already connected to other objects). To ensure accuracy, the annotator plays through the game and checks if the written object rules were correctly reflected in the gameplay.

C.3 Human-Written Object Rules

The object rules describe the meaning of each object property (e.g., temperature, size, weight, etc.) and how they will be changed at each time step. The expert annotators read the game description and source code for each game. They went through the object classes in the code script and wrote the object rules. Each object rule has three main parts: Object, Description, and Properties. The Object specifies the name of the object. The Description explains the general purpose of the object (e.g., GarbageCan is a container that can hold garbage). In the Description, the inheritance of the object class has been noted. The Properties is an unordered list of property descriptions that describe each property of that object (e.g., A Mold has its shape.) and their default value (e.g., By default, a GameObject is not combustible.) if the object is an abstract class. For objects with tick function, there is another property describing how an object may change under each tick. To ensure accuracy, the annotator plays through the game and checks if the written object rules were correctly reflected in the gameplay.

C.4 Human-Written Score Rules

Score rules describe the conditions to win or lose the game and how rewards can be earned. An expert annotator (one of the BYTESIZED32 game authors) creates the rules by reading the game description and the code of the score function.

D Prompts

The prompts introduced in this section includes game rules that can either be human written rules or LLM generated rules. For experiments without game rules, we simply remove the rules from the corresponding prompts.

D.1 Prompt Example: $\mathcal{F}_{\text{act}}$

D.1.1 Full State Prediction

Full State Prediction Prompt ($\mathcal{F}_{\text{act}}$)

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.
Your response should be in the same JSON format as the given game state.
Here is an example:
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here are the descriptions of all game actions in the example game:

{**ACTION_RULES**}
Here is the game state:
{**GAME_STATE**}
The action to take is put plate (ID: 5) in dirty cup (ID: 4)
The expected response is:
{**GAME_STATE**}
Here is the game that you need to simulate:
Task Description:
Your task is to figure out the weight of the cube. Use the answer action to give your answer.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here are the descriptions of all game actions:
{**ACTION_RULES**}
Here is the game state:
{**GAME_STATE**}
The action to take is:
look

D.1.2 State Difference Prediction

State Difference Prediction Prompt ($\mathcal{F}_{\text{act}}$)

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.
Your response should be in the JSON format. It should have two keys: 'modified' and 'removed'. The 'modified' key stores a list of all the object states that are added or changed after taking the action. Keep it an empty list if no object is added or modified. The 'removed' key stores a list of uuids of the objects that are removed. Keep it an empty list if no object is removed.
Here is an example:
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here are the descriptions of all game actions in the example game:

{**ACTION_RULES**}
Here is the game state:
{**GAME_STATE**}
The action to take is put plate (ID: 5) in dirty cup (ID: 4)
The expected response is:
{**GAME_STATE_DIFFERENCE**}
Here is the game that you need to simulate:
Task Description:
Your task is to figure out the weight of the cube. Use the answer action to give your answer.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here are the descriptions of all game actions:
{**ACTION_RULES**}
Here is the game state:
{**GAME_STATE**}
The action to take is:
look

D.2 Prompt Example: $\mathcal{F}_{\text{env}}$

D.2.1 Full State Prediction

Full State Prediction Prompt ($\mathcal{F}_{\text{env}}$)

You are a simulator of a text game. Read the task description. Given the current game state in JSON, you need to decide how the game state changes in the next time step (without considering the agent actions). Rules for such changes are described as the tick function of each object.
Your response should be in the same JSON format as the given game state.
Here is an example:
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here is the game state:
{**GAME_STATE**}
The expected response is:
{**GAME_STATE**}
Here is the game that you need to simulate:
Task Description:
Your task is to figure out the weight of the cube. Use the answer action to give your answer.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here is the game state:
{**GAME_STATE**}

D.2.2 State Difference Prediction

State Difference Prediction Prompt ($\mathcal{F}_{\text{env}}$)

You are a simulator of a text game. Read the task description. Given the current game state in JSON, you need to decide how the game state changes in the next time step (without considering the agent actions). Rules for such changes are described as the tick function of each object.
Your response should be in the JSON format. It should have two keys: 'modified' and 'removed'. The 'modified' key stores a list of all the object states that are added or changed after taking the action. Keep it an empty list if no object is added or modified. The 'removed' key stores a list of uuids of the objects that are removed. Keep it an empty list if no object is removed.
Here is an example:
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here is the game state:
{**GAME_STATE**}
The expected response is:
{**GAME_STATE_DIFFERENCE**}
Here is the game that you need to simulate:
Task Description:
Your task is to figure out the weight of the cube. Use the answer action to give your answer.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here is the game state:
{**GAME_STATE**}

D.3 Prompt Example: $\mathcal{F}_R$ (Game Progress)

Game Progress Prediction Prompt ($\mathcal{F}_R$)

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 predict the current game score, whether the game is over, and whether the agent wins the game.
Your response should be a JSON with three keys: 'score', 'gameOver', and 'gameWon'. 'score' stores the current game score, 'gameOver' stores a bool value on whether the game is over, and 'gameWon' stores a bool value on whether the game is won.
Here is an example:
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here is a description of the game score function:
{**SCORE_RULES**}
Here is the previous game state:
{**GAME_STATE**}
The game score of the preivous state is:
{'score': –1, 'gameOver': False, 'gameWon': False}
The action to take is use dish soap (ID: 12) on glass (ID: 8)
{**GAME_STATE**}
The expected response is:
{'score': 3, 'gameOver': True, 'gameWon': True}
Here is the game that you need to simulate:
Task Description:
Your task is to figure out the weight of the cube. Use the answer action to give your answer.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here is a description of the game score function:
{**SCORE_RULES**}
Here is the previous game state:
{**GAME_STATE**}
The game score of the preivous state is:
{'score': 0, 'gameOver': False, 'gameWon': False}
The action to take is:
look
Here is the current game state after taking the action:
{**GAME_STATE**}

D.4 Prompt Example: $\mathcal{F}$

D.4.1 Full State Prediction

Full State Prediction Prompt ($\mathcal{F}$)

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. You should assign the uuid of new objects starting from the UUID base given in the instructions.Your response should be in the same JSON format as the given game state.
Note that while game states can be changed by actions, some game states may change over the time, which is described in the tick function of each object class.
Here are two examples of both cases. Both examples are from the same example game.
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here are the descriptions of all game actions in the example game:

{**ACTION_RULES**}
Here is a description of the score function of the example game:
{**SCORE_RULES**}
In the first example, the game state is changed by an action:
Here is the game state:
{**GAME_STATE**}
The current game UUID base is 12
The action to take is: put plate (ID: 5) in dirty cup (ID: 4)
The expected response is:
{**GAME_STATE**}
In the second example from the same example game, the game state is changed over the time. Note that while in this example the game state is changed by time only, it is possible that a game state is changed by both an action and time.
Here is the game state:
{**GAME_STATE**}
The current game UUID base is 13
The action to take is: eat dishwasher (ID: 2) with dirty plate (ID: 5)
The expected response is:
{**GAME_STATE**}
Here is the game that you need to simulate:
{**OBJECT_RULES**}
Here are the descriptions of all game actions:
{**ACTION_RULES**}
Here is a description of the game score function:
{**SCORE_RULES**}
Here is the game state:
{**GAME_STATE**}
The current game UUID base is 12
The action to take is:
look

D.4.2 State Difference Prediction

State Difference Prediction Prompt ($\mathcal{F}$)

You are a simulator of a text game. Read the task description and the current environment observation description. Given the current game state in \textsc{JSON}, you need to decide the new game state after taking an action.
Your response should be in the \textsc{JSON} format. It should have three keys: 'modified', 'removed', and 'score'. The 'modified' key stores a list of all the object states that are added or changed after taking the action. Keep it an empty list if no object is added or modified. The 'removed' key stores a list of uuids of the objects that are removed. Keep it an empty list if no object is removed.
The 'score' key stores a dictionary with three keys: 'score' is the current game score, 'gameOver' is a boolean of whether the game is over, and 'gameWon' is a boolean of whether the agent won the game. If a player earns a score or wins/loses the game, you should reflect that change in the dictionary saved under the 'score' key. Otherwise, you should set value of the 'score' key to an empty dictionary.Note that while game states can be changed by actions, some game states may change over the time, which is described in the tick function of each object class.
Note that while game states can be changed by actions, some game states may change over the time, which is described in the tick function of each object class.
Here are two examples of both cases. Both examples are from the same example game.
Example game task description:
Your task is to wash the dirty dishes.
Here are the descriptions of all game objects properties in the example game:
{**OBJECT_RULES**}
Here are descriptions of all game actions in the example game:
{**ACTION_RULES**}
Here is a description of the score function of the example game:
{**SCORE_RULES**}
In the first example, the game state is changed by an action:
Current observation:
{**GAME_OBSERVATION**}
Here is the game state:
{**GAME_STATE**}
The action to take is put dirty plate (ID: 5) in mug (ID: 6)
The expected response is:
{**GAME_STATE_DIFFERENCE**}
In the second example from the same example game, the game state is changed over the time. Note that while in this example the game state is changed by time only, it is possible that a game state is changed by both an action and time.
Current observation:
{Example_2 observation}
Here is the game state:
{**GAME_STATE**}
The action to take is eat dishwasher (ID: 2) with dirty plate (ID: 5)
The expected response is:
{**GAME_STATE_DIFFERENCE**}
Here is the game that you need to simulate:
Task Description:
Your task is to boil water.
Here are the descriptions of all game objects properties:
{**OBJECT_RULES**}
Here are the descriptions of all game actions:
{**ACTION_RULES**}
Here is a description of the score function of the game:
{**SCORE_RULES**}
Current observation:
{**GAME_OBSERVATION**}
Here is the game state:
{**GAME_STATE**}
The current game UUID base is 12
The action to take is:
look

D.5 Other Examples

Below is an example of the rule of an action:

Action Rule Example

put:
Description: put an object into a target container
Rules:
1. The target must be a container (Container)
2. The target container must be open
3. The object must be in the inventory
4. The object must be moveable (isMoveable)

Below is an example of the rule of an object:

Object Rule Example

Object: Container
Description: Abstract class for things that can be considered ' containers' (e.g. a drawer, a box, a table, a shelf, etc.)
Properties:
– A Container is a container.
– A Container could be opened (e.g., e.g. a drawer, a door, a box, etc.), or is it always 'open' (e.g. a table, a shelf, etc.).
– A Container has a property indicating if it is opened.
– A Container has a property indicating the prefix to use when referring to the container (e.g. "in the drawer", "on the table", etc.). By default, the prefix is 'in'

Below is an example of the score rule:

Score Rule Example

The player wins the game by getting all dishes clean.
The player gets one point for each dish that is cleaned.
The player loses one point for each dish that is made dirty.

Below is an example of a game state:

Game State Example

{'game_state': [{'name': 'agent (ID: 0)', 'uuid': 0, 'type': 'Agent', ' properties': {'isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen': True, 'containerPrefix': 'in'}, 'contains': ['plate (ID: 5)', 'mug (ID: 6)', 'knife (ID: 7)']}, {'name': 'plate (ID: 5)', 'uuid': 5, ' type': 'Dish', 'properties': {'isContainer': True, 'isMoveable': True, ' isOpenable': False, 'isOpen': True, 'containerPrefix': 'on', 'dishType ': 'plate', 'isDirty': True, 'foodMessName': 'orange'}, 'contains': []}, {' name': 'mug (ID: 6)', 'uuid': 6, 'type': 'Dish', 'properties': {' isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen ': True, 'containerPrefix': 'in', 'dishType': 'mug', 'isDirty': True, ' foodMessName': 'sandwhich'}, 'contains': []}, {'name': 'knife (ID: 7) ', 'uuid': 7, 'type': 'Dish', 'properties': {'isContainer': True, ' isMoveable': True, 'isOpenable': False, 'isOpen': True, ' containerPrefix': 'in', 'dishType': 'knife', 'isDirty': True, ' foodMessName': 'apple (ID: 11)'}, 'contains': []}, {'name': ' dishwasher (ID: 2)', 'uuid': 2, 'type': 'DishWasher', 'properties': {' isContainer': True, 'isMoveable': False, 'isOpenable': True, 'isOpen ': True, 'containerPrefix': 'in', 'isDevice': True, 'isActivatable': True, ' isOn': False, 'cycleStage': 0, 'finishedCycle': False}, 'contains': [' cup (ID: 4)']}, {'name': 'cup (ID: 4)', 'uuid': 4, 'type': 'Dish', ' properties': {'isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen': True, 'containerPrefix': 'in', 'dishType': 'cup', ' isDirty': True, 'foodMessName': 'peanut butter'}, 'contains': []}, {' name': 'bottle of dish soap (ID: 3)', 'uuid': 3, 'type': 'DishSoapBottle ', 'properties': {'isContainer': False, 'isMoveable': True, 'isDevice': True, 'isActivatable': True, 'isOn': False}, 'contains': []}, {'name': ' glass (ID: 8)', 'uuid': 8, 'type': 'Dish', 'properties': {'isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen': True, ' containerPrefix': 'in', 'dishType': 'glass', 'isDirty': False}, 'contains': []}, {'name': 'bowl (ID: 9)', 'uuid': 9, 'type': 'Dish', 'properties': {' isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen ': True, 'containerPrefix': 'in', 'dishType': 'bowl', 'isDirty': False}, ' contains': []}, {'name': 'banana (ID: 10)', 'uuid': 10, 'type': 'Food', ' properties': {'isContainer': False, 'isMoveable': True, 'isFood': True}, 'contains': []}, {'score': –1, 'gameOver': False, 'gameWon': False}]}

Rules	State Change	$\mathcal{F}$ Full	$\mathcal{F}$ Diff	$\mathcal{F}_{\text{act}}$ Full	$\mathcal{F}_{\text{act}}$ Diff	$\mathcal{F}_{\text{env}}$ Full	$\mathcal{F}_{\text{env}}$ Diff
LLM	*dynamic*	34.5	21.4	36.0	31.7	7.8	2.9
	static	37.5	54.0	44.6	65.9	41.8	63.1
Human	*dynamic*	26.8	21.2	43.3	36.1	12.5	0.4
	static	35.6	58.9	42.3	64.7	22.0	74.2
No rule	*dynamic*	15.4	23.5	43.8	35.7	1.7	0.8
	static	26.9	50.0	35.2	63.0	17.2	54.8

Table 5: Average accuracy per game of GPT-3.5 predicting the whole state transitions ($\mathcal{F}$) as well as action-driven transitions ($\mathcal{F}_{\text{act}}$) and environment-driven transitions ($\mathcal{F}_{\text{env}}$). We report settings that use LLM generated rules, human written rules, or no rules. Dynamic and static denote whether the game object properties and game progress should be changed; Full and diff denote whether the prediction outcome is the full game state or state differences. Numbers shown in percentage.

Rules	**Game Progress**
LLM	73.9
Human	63.3
No rule	64.2

Table 6: GPT-3.5 game progress prediction results

Below is an example of a JSON that describes the difference of two game states:

Game State Difference Example

{'modified': [{'name': 'agent (ID: 0)', 'uuid': 0, 'type': 'Agent', ' properties': {'isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen': True, 'containerPrefix': 'in'}, 'contains': ['mug (ID: 6)', 'knife (ID: 7)']}, {'name': 'mug (ID: 6)', 'uuid': 6, 'type': 'Dish', ' properties': {'isContainer': True, 'isMoveable': True, 'isOpenable': False, 'isOpen': True, 'containerPrefix': 'in', 'dishType': 'mug', ' isDirty': True, 'foodMessName': 'sandwhich'}, 'contains': ['plate (ID: 5)']}], 'removed': [], 'score': {}}

E GPT-3.5 results

Table 5 and Table 6 shows the performance of a GPT-3.5 simulator predicting objects properties and game progress respectively. There is a huge gap between the GPT-4 performance and GPT-3.5 performance, providing yet another example of how fast LLM develops in the two years. It is also worth notices that the performance difference is larger when no rules is provided, indicating that GPT-3.5 is especially weak at applying common sense knowledge to this few-shot world simulation task.

F Histograms

1. In Figure 3, we show detailed experimental results on the **full state prediction task** performed by **GPT-4**.

Property Name	Description
buried	Objects buried in the room
combustionTimeRemaining	Number of time steps remaining to combust of a combusting object
connects	Electrical objects connecting to the current object
contains	Objects in the current object
cook	How an ingredient is cooked
current_aperture	Current aperture of a camera
current_focus	The object that the camera is currently focusing on
current_iso	Current ISO of a camera
current_shutter_speed	Current shutter speed of a camera
cut	How an ingredient is cut
cycleStage	The current stage of the washing machine's cycle (running/washing/finished).
durability	Number of times left for a shovel to dig something
finishedCycle	A boolean indicator of whether the washing machine has finished
food	The food level of a young bird. Reduce 1 if the young bird is not fed at each time step.
grow	Number of time steps that a young bird has grown
hatch	Number of time steps that an egg is hatched
isAboveMaxTemp	Whether the temperature of the current food is above its maximum preservation temperature
isActivated	Whether a device is activated
isChoppable	Whether an object is choppable
isCombusting	Whether an object is combusting
isDirty	Whether a dish is dirty
isMoveable	Whether the current object is moveable
isOn	Whether a device is turned on
isOpen	Whether a container is open
isWet	Whether a clothes is wet
is_open	Whether a door is open
liquid	Whether there is liquid in a container
mode	Mode of a multimeter
objects	Record of the number of time steps that each object is on the inclined plane
on	Whether a light bulb is on
photo	The object that the camera has taken a picture of
prefix	Prefix abstract to describe the object. E.g., **a** tree and **some** firewood
stage	Life stage of a bird
stateOfMatter	State of matter of a substance
sunburn	Whether the player's skin is burnt by the sun
temperature	Object temperature
tick	Number of ticks that an object is placed on an inclined plane
timeAboveMaxTemp	Number of time steps that a food is above its maximum preservation temperature
use_sunscreen	Whether the player has used the sunscreen
volume	Volume of an object
warm	The warmth received by an egg during its hatching stage
wearSpaceSuit	Whether the agent wears the spacesuit

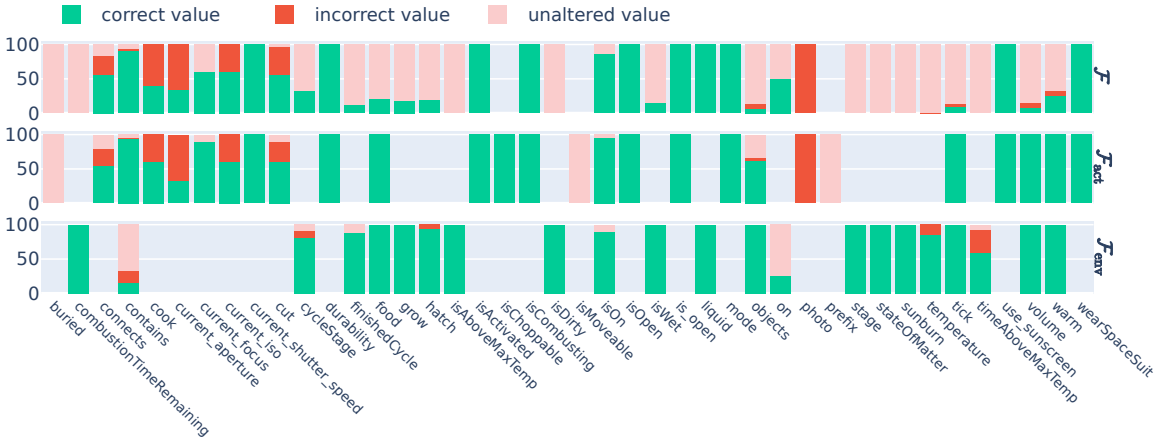
Table 7: Description of object properties mentioned in Figure 2

2. In Figure 4, we show detailed experimental results on the **state difference prediction task** performed by **GPT-4**.

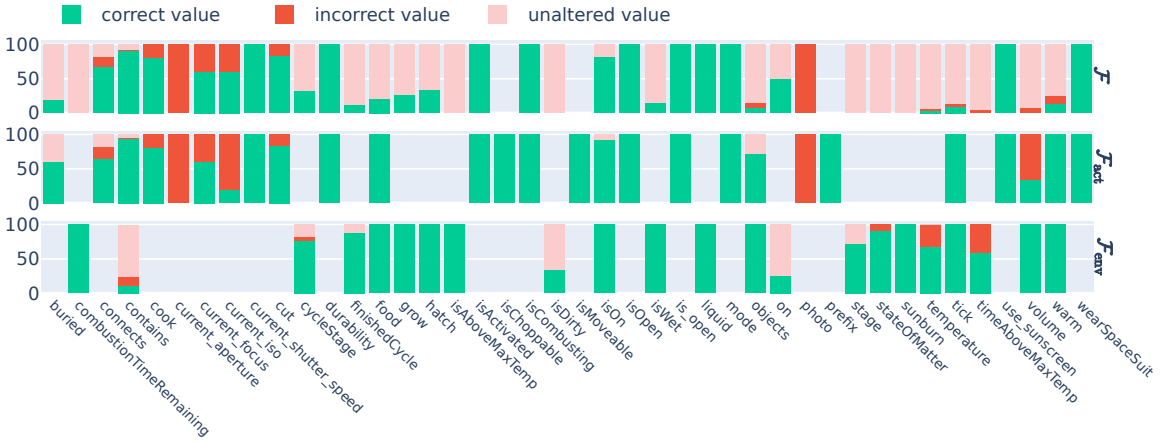
3. In Figure 5, we show detailed experimental results on the **full state prediction task** performed by **GPT-3.5**.

4. In Figure 6, we show detailed experimental results on the **state difference prediction task** performed by **GPT-3.5**.

(a) Human-generated rules.



(b) LLM-generated rules.



(c) No rules.

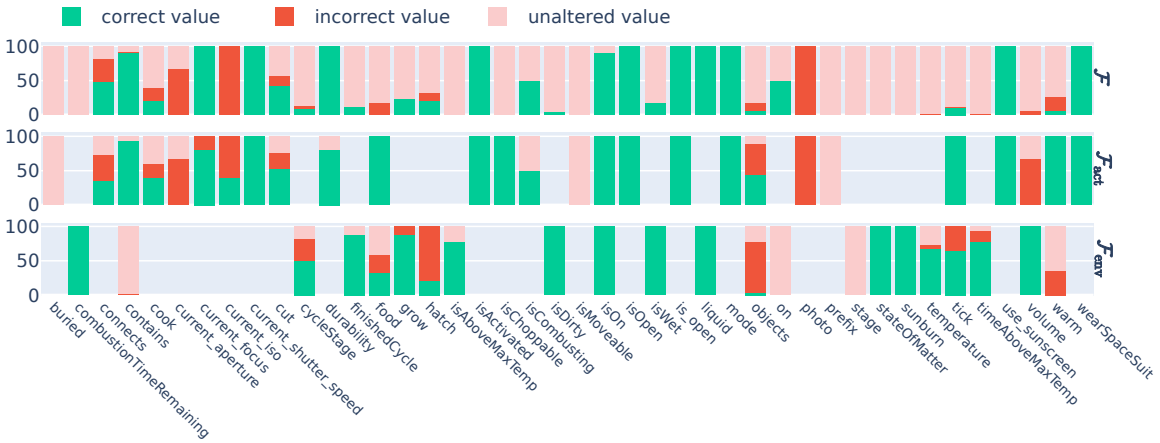
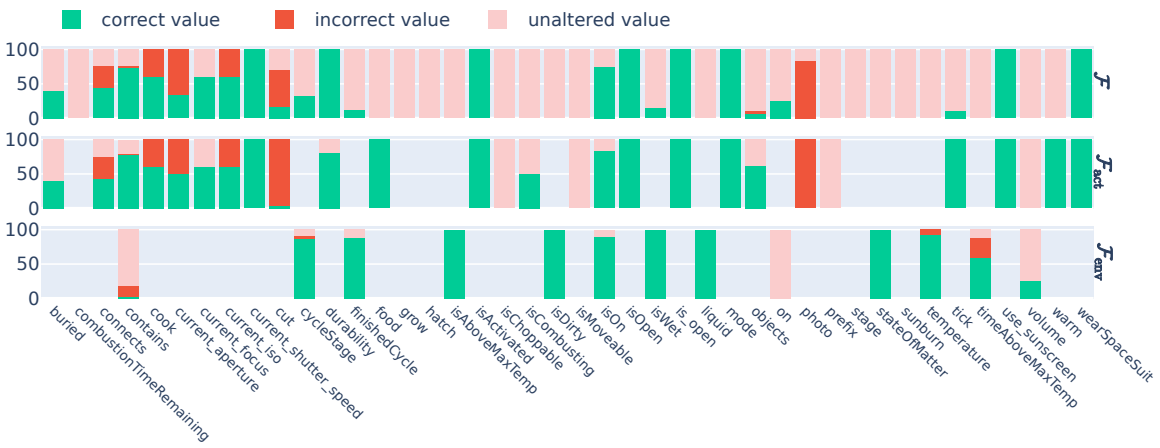
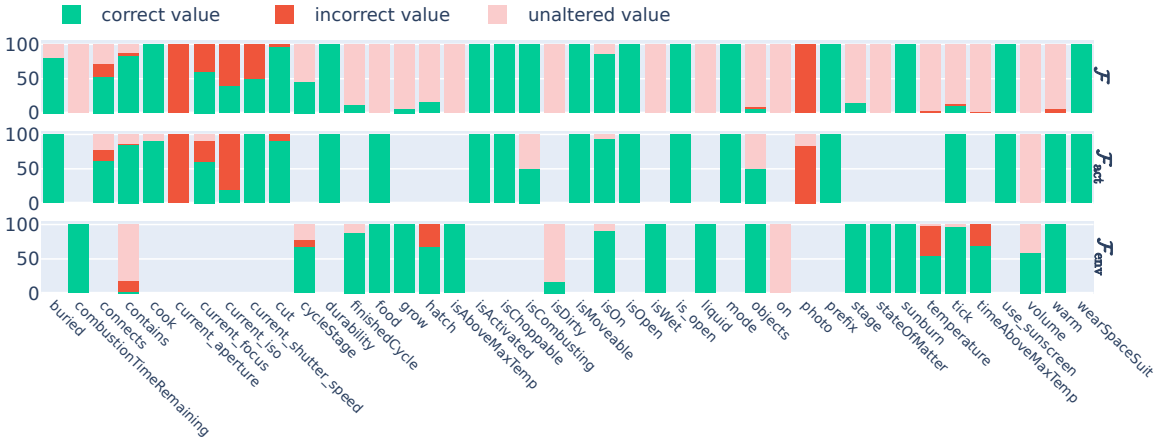


Figure 3: GPT-4 - Full State prediction from a) Human-generated rules, b) LLM-generated rules, and c) No rules.

(a) Human-generated rules.



(b) LLM-generated rules.



(c) No rules.

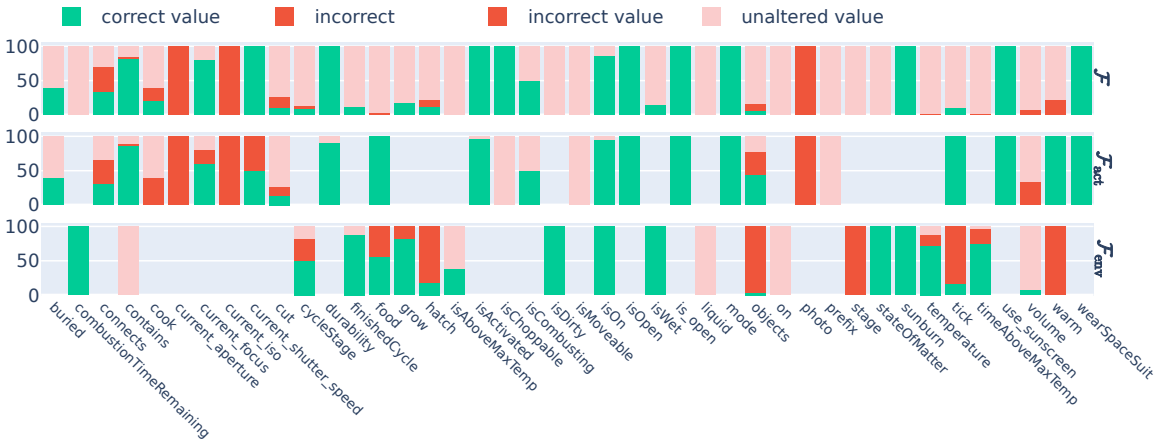
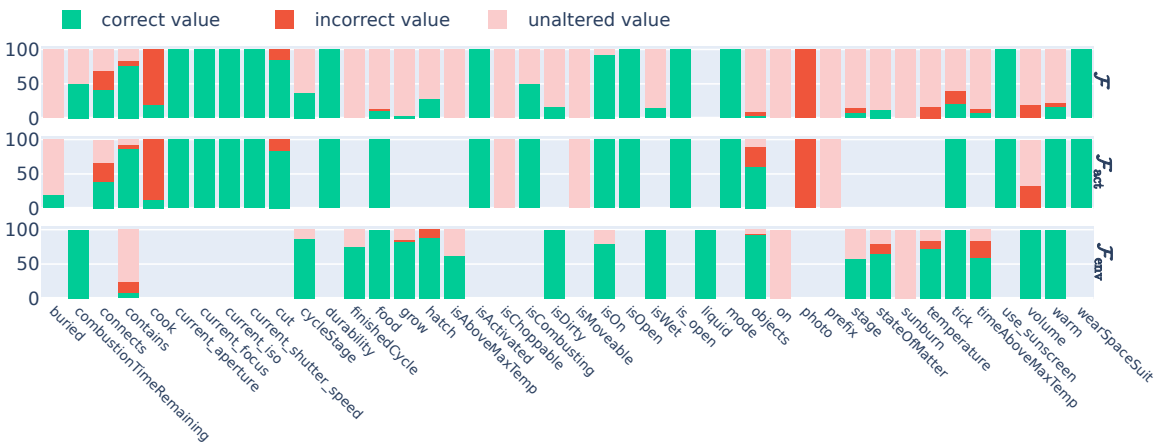
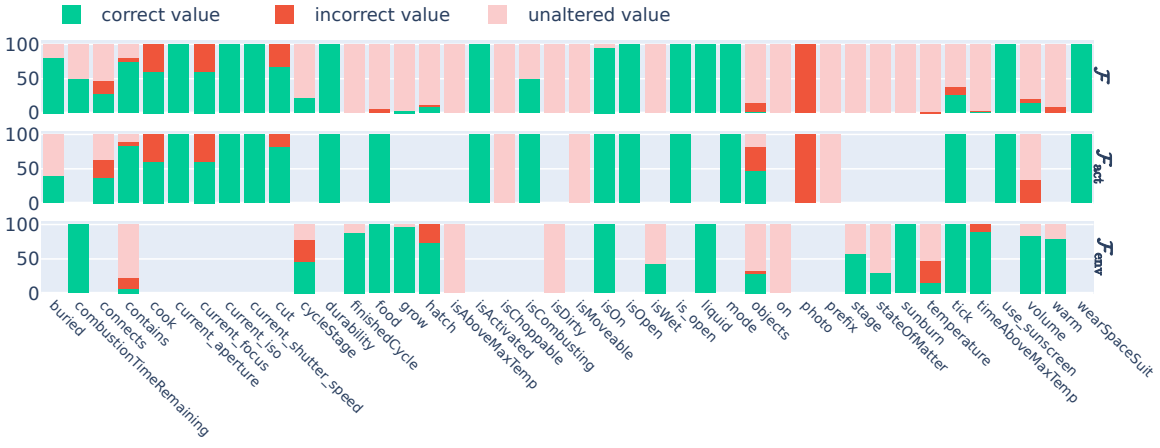


Figure 4: GPT-4 - Difference prediction from a) Human-generated rules, b) LLM-generated rules, and c) No rules.

(a) Human-generated rules.



(b) LLM-generated rules.



(c) No rules.

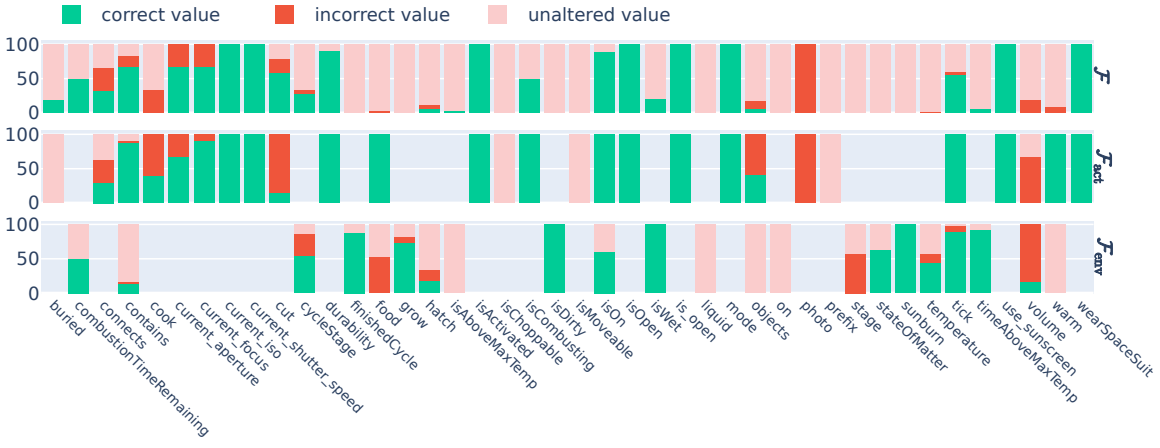
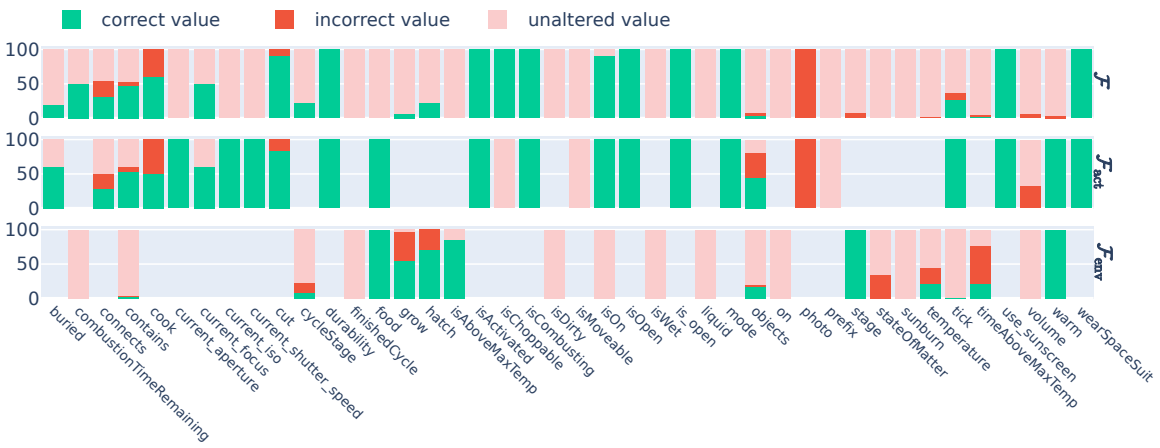
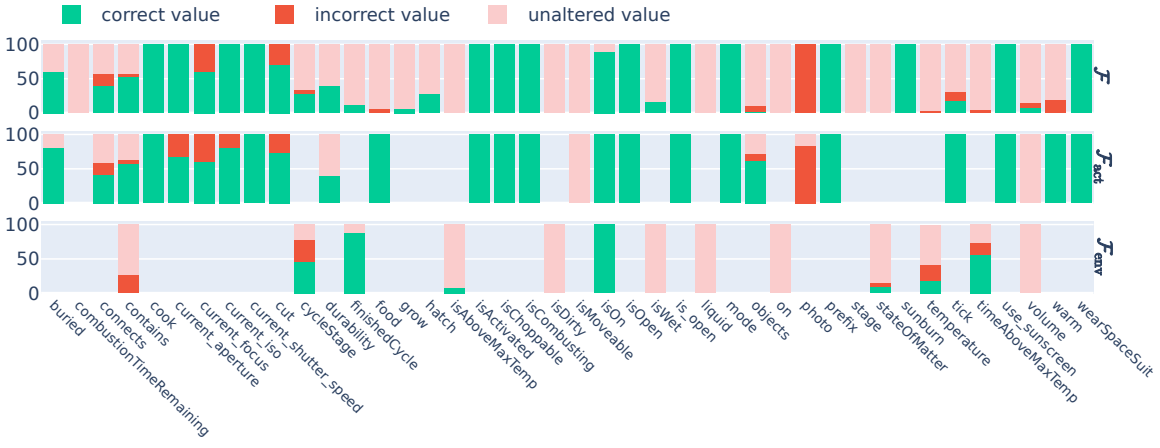


Figure 5: GPT-3.5 - Full State prediction from a) Human-generated rules, b) LLM-generated rules, and c) No rules.

(a) Human-generated rules.



(b) LLM-generated rules.



(c) No rules.

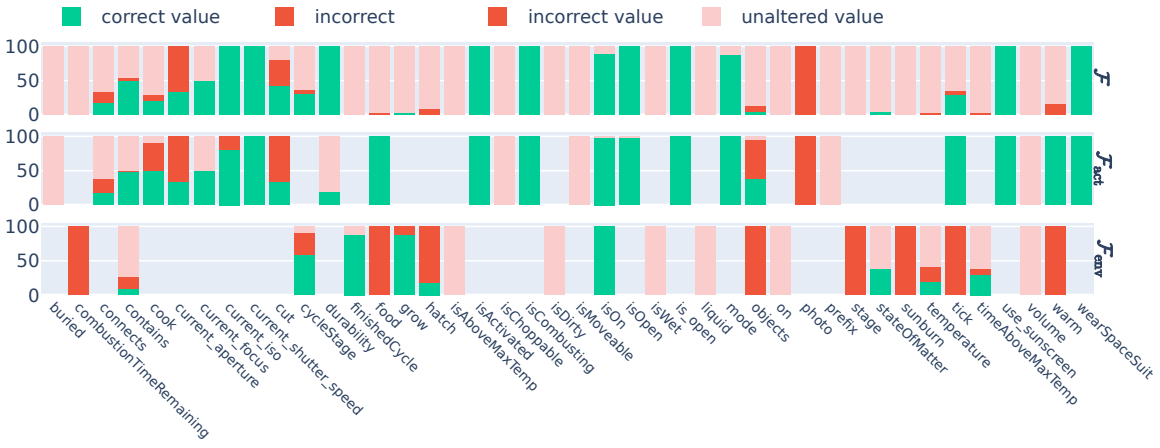


Figure 6: GPT-3.5 - Difference prediction from a) Human-generated rules, b) LLM-generated rules, and c) No rules.