

Utilizing GPT-4 to Solve TextWorld Commonsense Games Efficiently

Binggang Zhuo, Masaki Murata

Tottori University

zhuobinggang@gmail.com, murata@tottori-u.ac.jp

Abstract

Most artificial intelligence agents in interactive fiction games are implemented using reinforcement learning. Considering the recent rapid development of large language models, we propose an approach that utilizes a large language model to tackle interactive fiction game tasks. The chosen test dataset is TextWorld Commonsense, an interactive fiction game environment designed for artificial intelligence agents. In these games, the AI agent’s task is to organize rooms and place items in appropriate locations. To achieve a high score in the game, common sense knowledge about “which items belong to which locations” is important. Our approach is based on GPT-4 and a carefully designed prompt. Experimental results demonstrate that our approach outperforms prior research. Specifically, GPT-4 with feedback-augmented prompt successfully completed all tasks in both simple and medium level game environments without fine-tuning. In hard level game environments, our approach achieved a normalized score of 0.70, surpassing the best baseline score of 0.57.

Keywords: GPT4, text game, common sense

1. Introduction

Interactive fiction games pose a challenging natural language processing task. In interactive fiction games, both player commands and feedback from the engine are in natural language form. Therefore, to play these games, artificial intelligence (AI) agents not only need to comprehend the text responded by the game engine but also generate natural language to take an action. In addition to language understanding, AI agents often require other complex abilities such as long-term memory, planning, and common sense to attain high scores. Due to sparse rewards, traditional AI agents for interactive fiction games have relied on reinforcement learning techniques.

The main challenge of reinforcement learning lies in its difficulty of training. Considering the rapid development of large language models (LLMs) in recent years (Zhao et al., 2023), we propose an approach that utilizes GPT-4 (Bubeck et al., 2023) to solve interactive fiction games. The carefully designed prompt is the main part of our research approach, drawing inspiration from key achievements in the field of prompt engineering, such as Chain-of-Thought (COT), few-shot prompting and output templates. COT encourages deep reasoning within LLMs through specific prompt phrases (e.g., “think step by step”), resulting in more coherent outputs (Wei et al., 2022). Few-shot prompting helps LLMs deepen their understanding of the task by providing examples in the prompt. Output template, inspired by Park et al. (Park et al., 2023), standardizes model outputs using “<fill in>” prompts, facilitating subsequent automated processing. Additionally, we introduce feedback augmented prompting, a technique that simply clarifies reward and punish-

ment information responded by the game engine, aiding large language models in better grasping the situation.

We evaluated the performance of our approach on the TextWorld Commonsense dataset (Murugesan et al., 2021) (details in Section 4.1). Experimental results show that GPT-4 with our carefully designed prompts successfully completed all tasks in simple and medium level games. In hard level game environments, our approach achieved a normalized score of 0.70, surpassing the best baseline performance (score of 0.57). The contributions of our study are as follows:

- We evaluated the performance of large language models on the TWC dataset, and by integrating Chain-of-Thought (Wei et al., 2022), few-shot prompting, and feedback augmentation, our research approach achieved the state-of-art performance on the TWC dataset as we known. While demonstrating outstanding performance, our approach also possesses the advantage of simplicity in implementation.
- Through ablation experiments, we confirmed the contributions of each module to our research approach and highlighted the significance of the feedback augmentation technique proposed in this study.

2. Related Work

2.1. Datasets

Interactive fiction games were initially designed for human players, with one of the most famous games being Zork. Numerous studies on Zork indicate that the game poses a challenging task for existing AI agents (Haroush et al., 2018a; Yao et al.,

2021; Tsai et al., 2023) (including agents based on large language models). Instead of directly studying difficult games, we try to incrementally improve the performance of AI agents in simpler game environments. To address the need of customizing game environments, Côté et al. (Côté et al., 2019) introduced TextWorld, a sandbox environment that allows users to create games manually or through a set of generation mechanisms. The TextWorld Commonsense environment (Murugesan et al., 2021) (TWC) is built on TextWorld, with the primary goal of assessing agents' ability to apply common sense. Following TWC, more complex interactive fiction game environments emerged, such as Science World (Wang et al., 2022) and Complex World (Basavatia et al., 2023). The challenge with Science World lies in its domain-specific environment, where an agent performing well may lack general skills to apply in other contexts. Complex World, being more intricate than TWC, is a subject for our future study.

2.2. Agents

Due to the sparse rewards in interactive fiction games, reinforcement learning has naturally become the primary implementation method for AI agents. LSTM-DQN (Narasimhan et al., 2015) stands out as one of the earliest achievements in this field, where Karthik et al. used LSTM to encode textual observation into hidden states. DRRN (He et al., 2015), an improvement upon LSTM-DQN, separated the neural network into two parts to encoding states and actions separately, enhancing the model's performance and stability. KG-A2C (Ammanabrolu and Hausknecht, 2020), during the exploration process, generates a knowledge graph to further guide the model's exploration, proving to enhance the learning efficiency and performance of agents. These approaches mentioned above are used as baselines in the TextWorld Commonsense paper. It is worth noting that as stated in the TWC paper (Murugesan et al., 2021), LSTM-DQN is implemented as LSTM-A2C, which represents a different reinforcement learning paradigm more suitable for the TWC environment. The proposed TWC Agent in the TWC paper extracts common sense from ConceptNet and, by combining this common sense knowledge, achieves the highest performance among all models mentioned above.

2.3. Prompt Engineering

Due to the vast number of parameters in large language models, fine-tuning has become a costly task. This problem have prompted people to seek better ways to utilize large language models, and prompt engineering is one of the best options. Prompt engineering significantly enhances models' performance on specific tasks without modify-

ing their parameters, achieved through cleverly designed prompts. There is a wealth of prior research in the field of prompt engineering, and we will only introduce three techniques used in this study: (1) Few-shot prompting. Brown et al. (Brown et al., 2020) trained GPT-3 and found that significant performance improvement in large language models can be achieved by providing only a few examples within prompts, without the need for costly fine-tuning. (2) Chain-of-Thought. Wei et al. (Wei et al., 2022) found that rather than directly outputting results, prompting the model to output the thought process at the same time leads to more accurate results. They prompt the model to output in a similar manner by providing Chain-of-Thought examples. Subsequently, Kojima et al. (Kojima et al., 2022) further pointed out that similar effects can be achieved by simply appending special key words (such as "think step by step") in prompts. (3) Output template. Joon et al.'s study on AI villages (Park et al., 2023) is one of the most impressive works of large language models in recent years. We borrowed the idea of output templates from this paper. Specifically, prompts like "<fill in>" can standardize the model's output, facilitating subsequent automation processes.

3. Model

3.1. Problem Setup

Our study focuses on using large language models to solve interactive fiction games. The dataset selected for our study is the TextWorld Commonsense (TWC), which includes five games for each of the three difficulty levels: easy, medium, and hard. The dataset can be found on GitHub¹. In each game, the AI agent's task is to organize rooms, specifically by placing certain objects back in their designated locations. For example, a used tissue on the ground should be put in the trash can, and a dirty piece of clothing should be placed in the washing machine. The criteria for determining which object belongs to a specific location are based on common sense, and the validity of these object-location pairs has been confirmed by human evaluators (Murugesan et al., 2021).

At each time step, the agent receives an observation from the game engine, presented in natural language form. Subsequently, the agent needs to respond in natural language to present its consideration and take action. After the chosen action is executed, the game engine returns a new observation, and then the agent needs to choose a new action. This cycle has two termination conditions: either the agent completes all tasks or the number of actions taken exceeds the predefined limit. In our study, the action limit is set to 20.

¹<https://github.com/IBM/commonsense-rl/tree/master/games/twc>

It should be noted that as stated in the TWC paper (Murugesan et al., 2021), the maximum number of steps for all agents is set to 50, while we limit the number of steps to 20 for two reasons. Firstly, our study aims to improve the efficiency of GPT-4 under restricted conditions. Secondly, limiting the number of steps not only reduces the frequency of API calls but also keeps the action history concise, which leads to a lower API usage cost. On the other hand, as stated in the TWC paper, the average number of steps taken by humans to complete hard-level games is 15.

The observations returned by the game engine include following content:

- **Action Feedback:** If an action is executed, the agent receives feedback from the environment. If the model earns a reward in an action, that information is also included in the feedback. For example, a pair of action and feedback could be: *"insert dirty yellow T-shirt into washing machine -> You put the dirty yellow T-shirt into the washing machine. Your score has just gone up by one point"*. In this example, *"insert dirty yellow T-shirt into washing machine"* is the action, and the part after the arrow is feedback from the game engine.
- **Environment Description:** Textual description of the current environment.
- **Inventory:** A list of currently held items.
- **Available Action List:** A list of actions currently available to the agent.

3.2. Prompts

Our study proposes an approach using large language models to play interactive fiction games, with the carefully designed prompt being the most crucial part of our approach. Specifically, the prompt engineering techniques included in our prompts are: one-shot prompting, Chain-of-Thought, output template, and feedback augmentation.

An example of prompts is provided below:

Task: You are an experienced text game player, your goal is to put things in their proper locations and improve your score.

Example walkthrough: Action 0: insert dirty yellow dress into washing machine -> You put the dirty yellow dress into the washing machine. Your score has just gone up by one point. Right position. Action 1: take dirty yellow T-shirt from bench -> You take the dirty yellow T-shirt from the bench. Action 2: insert dirty yellow T-shirt into washing machine -> You put the dirty yellow T-shirt into the washing machine. Your score has just gone up by one point.

Right position. Action 3: take wet azure dress from suspended shelf -> You take the wet azure dress from the suspended shelf. Action 4: insert wet azure dress into clothes drier -> You put the wet azure dress into the clothes drier. Your score has just gone up by one point. Right position. Action 5: take white cap from bench -> You take the white cap from the bench. Action 6: go east -> -= Corridor -=You've entered a corridor. Action 7: put white cap on hat rack -> You put the white cap on the hat rack. Your score has just gone up by one point. Right position. Action 8: take dirty cardigan -> You pick up the dirty cardigan from the ground. Action 9: take dirty checkered shirt from shoe cabinet -> You take the dirty checkered shirt from the shoe cabinet. Action 10: take dirty maroon dress -> You pick up the dirty maroon dress from the ground. Action 11: go west -> -= Laundry Room -=You've entered a laundry room. Action 12: insert dirty cardigan into washing machine -> You put the dirty cardigan into the washing machine. Your score has just gone up by one point. Right position. Action 13: insert dirty checkered shirt into washing machine -> You put the dirty checkered shirt into the washing machine. Your score has just gone up by one point. Right position. Action 14: insert dirty maroon dress into washing machine -> You put the dirty maroon dress into the washing machine. Your score has just gone up by one point. Right position.

Action history: Action 0: examine suspended shelf ->The suspended shelf is reliable. Action 1: take dirty singlet ->You pick up the dirty singlet from the ground.

Inventory: You are carrying: a dirty singlet

Current environment: -= Laundry Room -= You've entered a laundry room. You begin to take stock of what's here. You can see a washing machine. Empty! What kind of nightmare TextWorld is this? You make out an opened clothes drier. The clothes drier is empty! What a waste of a day! You can see a laundry basket. Hmm... what else, what else? The laundry basket is empty! What a waste of a day! You can see a suspended shelf. The suspended shelf is ordinary. But oh no! there's nothing on this piece of garbage. You move on, clearly infuriated by TextWorld. You scan the room, seeing a bench. But the thing hasn't got anything on it. Oh! Why

couldn't there just be stuff on it? You make out a work table. But oh no! there's nothing on this piece of garbage.

Action you can take:

- * close clothes drier
- * close laundry basket
- * close washing machine
- * drop dirty singlet
- * examine bench
- * examine clothes drier
- * examine laundry basket
- * examine suspended shelf
- * examine washing machine
- * examine work table
- * insert dirty singlet into clothes drier
- * insert dirty singlet into laundry basket
- * insert dirty singlet into washing machine
- * look
- * put dirty singlet on bench
- * put dirty singlet on suspended shelf
- * put dirty singlet on work table

Question: To put things in their proper locations and improve your score, what should you do? Think step by step then choose 'one' action from above list.

Consideration: <fill in>

Next action: <fill in>

"**Task**" is the task description that remains constant for each time step. "**Example walkthrough**" is an one-shot example we provide to large language models, which is basically the action history taken by a human player in a hard-level game from the training dataset. To demonstrate the movement of objects across rooms, we deliberately selected an environment² consisting of two rooms.

The "**Action history**" is the memory of LLMs, getting longer as the time step increases, and it is textualized in the format of "*action -> action feedback*". Action is extracted from the response of LLM, while action feedback is basically the content returned by the game engine. However, it should be noted that we have appended content to item-placing action feedback for **feedback augmentation**. For details, please refer to Section 3.3. Another point to note is that while action feedback is always short, when the agent moves to a new room, the feedback provides a complete description of the environment, consistent with the "**Current environment**" part in the

²<https://github.com/IBM/commonsense-rl/blob/master/games/twc/hard/train/tw-iqa-cleanup-objects7-take6-rooms2-train-aE0OFxjEcxEI9Xo.uIx>

prompt. Therefore, to limit the length of prompts, we only retain the content before the first period in such cases.

"**Current environment**" is a textual description of the current room obtained from the game engine. The "**Action you can take**" is a list of possible actions obtained from the game engine. The "**Question**," "**Consideration**," and "**Next Action**" remain unchanged at each time step and can be considered as an output template. Of particular note are the keywords "Think step by step" and "Consideration: <fill in>," which are simple implementations of the Chain-of-Thought.

For the above prompt, a sample output from GPT-4 turbo could be:

Consideration: To properly place the dirty singlet, it should be put somewhere related to laundry processing. Inserting it into the washing machine seems like the most appropriate step since typically, dirty clothes need to be washed before they are dried or placed in a laundry basket.

Next action: insert dirty singlet into washing machine

3.3. Feedback augmentation

We propose feedback augmentation due to the ambiguous feedback provided by the TWC game engine, especially concerning item placement actions. This ambiguity hinders GPT-4 from making accurate judgments.

For instance, in the case of incorrectly placing an item, we might get action-feedback pairs like: "*insert clean white polo shirt into chest of drawers -> You put the clean white polo shirt into the chest of drawers.*" As seen, the environment informs us that the action has been successfully executed, but it does not indicate that the item (polo shirt) does not belong to that location (chest of drawers).

However, when an item is correctly placed, we get action-feedback pairs like: "*insert clean white polo shirt into wardrobe -> You put the clean white polo shirt into the wardrobe. Your score has just gone up by one point.*" By comparing the two environmental feedbacks, we find that a score increase implies the item is correctly placed. While a fine-tuned model might learn to judge the effectiveness of an action based on the presence or absence of score growth information, for a 0-shot setting, the default action feedback is undoubtedly ambiguous.

Considering these factors, we introduce feedback augmentation technique. The approach is very simple; we just append a sentence of explanatory text after the item-placing action feedback. Specifically, when an item is placed correctly, we append "**Right position**" to the feedback text as a hint. An example is as follows.

Insert clean white polo shirt into wardrobe
-> You put the clean white polo shirt into the wardrobe. Your score has just gone up by one point. **Right position.**

For incorrectly placement, we append "**Wrong position, you should put it somewhere else, maybe the other room**" to the feedback. For example:

Insert clean white polo shirt into chest of drawers -> You put the clean white polo shirt into the chest of drawers. **Wrong position, you should put it somewhere else, maybe the other room.**

In this example, the action is incorrect because the shirt should be placed in the wardrobe but not the chest of drawers. The "**maybe the other room**" part encourages the model to explore across rooms. These action-feedback pairs will then be stored in the action history.

3.4. Using Prompts via API

In section 3.2, we introduced prompts used in our approach. In this section, we briefly discuss the usage of prompts. We utilize the OpenAI API to invoke GPT-4 Turbo (specifically named gpt-4-1106-preview) and GPT-3.5 Turbo (specifically named gpt-3.5-turbo-0613). For example, when calling GPT-4 Turbo, the specific Python code is as follows:

```
1 from openai import OpenAI
2 client = OpenAI()
3 completion = client.chat.completions.
4     create(
5         model='gpt-4-1106-preview',
6         messages=[
7             {"role": "system", "content":
8                 system_msg},
9             {"role": "user", "content": user_msg
10            }
11        ]
12    )
```

Listing 1: Python code for calling GPT-4 Turbo

Here, 'system_msg' and 'user_msg' are string variables containing the text content. In our usage, they respectively encompass the following text content:

- **system_msg**: Task, Example walkthrough, Action history, Inventory, Current environment.
- **user_msg**: Available action List, Question, Consideration, Next action.

4. Experiments

4.1. Dataset

The dataset used to evaluate the performance of the approach is TextWorld Commonsense (TWC),

which can be found on GitHub³. This dataset includes five games each for easy, medium, and hard levels. The levels differ mainly in the number of items to organize and the number of rooms. The easiest game involves one item and one room, while the most challenging game consists of seven items and two interconnected rooms. Agents can move between rooms using commands like "go east."

4.2. Experimental Setup

We use the OpenAI API to call GPT-4 Turbo (specifically named gpt-4-1106-preview) and GPT-3.5 Turbo (named gpt-3.5-turbo-0613). We did not fine-tune these models. For each game, we set a maximum limit of 20 actions that the model can take. For each approach proposed and each game, we run the experiment only once and obtain its result. Similar to the TWC paper (Murugesan et al., 2021), we use normalized scores (score achieved ÷ maximum achievable score) to evaluate the performance of different approaches.

4.3. Baselines

The baselines used in our study include LSTM-A2C, DRRN, KG-A2C, and TWC Agent (introducing common sense knowledge through Numberbatch). Characteristics of these various baselines have been introduced in the related works section. It is important to note that the scores of these models are directly obtained from the TWC paper rather than being re-implemented and tested. The scores for these baselines are taken as the average of ten repeated experiments (Murugesan et al., 2021).

5. Results

The experimental results are presented in Table 1. The scores in the table are normalized scores, calculated by dividing the achieved score by the maximum achievable score in games. Note that each difficulty level includes five games. As shown in the table, our proposed approach outperformed the best-performing baseline in various difficulty levels. It is noteworthy that all baseline methods underwent 100 episodes of training on the training set, while we did not fine-tune the large language models.

We will discuss in the ablation study section about which prompt engineering techniques are crucial for our research method.

5.1. Ablation Study

"-GPT4": Our approach relies on GPT-4, and a natural question is how the performance would change by replacing it with other large language models. Therefore, we replace GPT-4 with GPT-3.5 in our

³<https://github.com/IBM/commonsense-rl/tree/master/games/twc>

	Easy	Medium	Hard
LSTM-A2C	0.86	0.74	0.54
DRRN	0.81	0.73	0.44
KG-A2C	0.85	0.72	0.46
TWC-Agent	0.96	0.85	0.57
Our approach	1.00	1.00	0.70

Table 1: Experimental Results. For detailed experimental settings, please refer to Section 4.

approach and name it "-GPT4". It is important to note that due to the presence of the output template (*Question*, *Consideration*, and *Next Action*), GPT-4's responses are well-structured, and actions can be automatically extracted from the response text. However, GPT-3.5's responses are rarely structured, requiring additional grounding techniques to extract a valid action. In our study, we manually extract action from GPT-3.5's response text. The human assistant's task is to copy a part of text from the model's response (in the example above, it is the part after "*Next action*"), and paste it into the interactive interface.

"-COT": The output template in our prompt is like:

Question: To put things in their proper locations and improve your score, what should you do? Think step by step then choose 'one' action from above list.

Consideration: <fill in>

Next action: <fill in>

This output template was inspired by the Chain-of-Thought (COT) study (Wei et al., 2022). A natural question is whether removing the COT part from the prompt would lead to a performance decline. Therefore, we modify the output template as follows:

Question: To put things in their proper locations and improve your score, what should you do? Choose 'one' action from above list.

Next action: <fill in>

Compared to the previous template, we removed "*Think step by step then*" and "*Consideration: <fill in>*". We name the approach based on this prompt as "-COT".

"-one-shot": Brown et al. (Brown et al., 2020) suggest that providing examples within prompts to large-scale language models can significantly improve their performance on specific tasks. To test the effectiveness of the one-shot setting, we removed the "Example walkthrough" part from our prompts and named this technique "-one-shot".

"-FA": Feedback augmentation is the main innovation in our study. To show the performance gain

	Easy	Medium	Hard
TWC-Agent	0.96	0.85	0.57
Our approach	1.00	1.00	0.70
-GPT4	1.00	0.58	0.18
-COT	1.00	1.00	0.61
-one-shot	1.00	1.00	0.64
-FA	1.00	1.00	0.52

Table 2: Ablation experiment results. **COT** means Chain-of-Thought, and **FA** means feedback augmentation. For detailed experimental settings, please refer to Section 5.1.

from feedback augmentation, we remove the explanatory text appended after the action feedback (basically restoring the default behavior of the game environment, refer to Section 3.3), naming this approach "-FA" (-Feedback Augmentation). For example, below is the augmented action-feedback pair when an item is misplaced. In the "-FA" approach, we simply remove the hint marked in bold.

Insert clean white polo shirt into chest of drawers -> You put the clean white polo shirt into the chest of drawers. **Wrong position, you should put it somewhere else, maybe the other room.**

The results of the ablation study are presented in Table 2. Firstly, it is evident that GPT-4 is the most crucial part for our approach. If GPT-4 is replaced with GPT-3.5, the approach's performance on hard-level games drops from a normalized score of 0.70 to 0.18. Moreover, many outputs from GPT-3.5 turbo do not meet the valid format, requiring manual adjustments. The second crucial module for our method is feedback augmentation. After removing feedback augmentation, the model's performance on hard-level games drops from 0.70 to 0.52. Lastly, we find that Chain-of-Thought and one-shot setting also significantly improves the model's performance.

6. Discussion

6.1. One-shot Prompting

One-shot (as well as few-shot) prompting assists large language models in understanding task detail by providing examples within prompts. The main issue with one-shot prompting lies in the significant impact of example selection on the model's performance.

As mentioned in Section 3.2, to provide a demonstration of moving items across rooms, we deliberately chose a game environment consisting of two rooms for obtaining the example walkthrough.

To investigate the impact of sample selection on the performance of our approach, in this section,

	Easy	Medium	Hard
TWC-Agent	0.96	0.85	0.57
Our approach	1.00	1.00	0.70
-one-shot	1.00	1.00	0.64
Easy one-shot	1.00	1.00	0.64

Table 3: One-shot prompting with easy-level example experiment result. For detailed experimental settings, please refer to Section 6.1.

we obtained an example walkthrough from an easy-level game and replaced the hard-level example in the prompts. The specific easy-level walkthrough is as follows:

Example walkthrough: Action 0: take dirty gray underpants from work table -> You take the dirty gray underpants from the work table. Action 1: insert dirty gray underpants into washing machine -> You put the dirty gray underpants into the washing machine. Your score has just gone up by one point. Right position.

The performance of the model after adopting the easy-level one-shot setting (denoted as "**Easy one-shot**") is shown in Table 3.

As shown in Table 3, we found that example selection will significantly affect model's performance. After replacing the hard-level walkthrough with that of the easy level, the model's performance decreased from 0.70 to 0.64, which remained consistent with the performance of not zero-shot setting (denoted as "**-one-shot**").

6.2. Case Study

In simple and medium level games, our approach keeps completing all tasks in the fewest steps. However, in hard-level games, the model may still make **incorrect actions**. These actions are primarily due to certain items not belonging to the current room. For example, the following case is from the first game at the hard level:

Action 0: take wet white jumper from patio chair -> You take the wet white jumper from the patio chair. Action 1: put wet white jumper on clothesline -> You put the wet white jumper on the clothesline. Your score has just gone up by one point. Right position. Action 2: **put milk on patio table** -> You put the milk on the patio table. Wrong position, you should put it somewhere else, maybe the other room.

The part marked in bold indicates the incorrect action made by the model. As indicated by the engine feedback, in this example, the milk does not belong

to the patio table. While making this action, GPT-4's consideration is as follows:

Consideration: The milk you are carrying is likely intended to be placed somewhere it can remain cool or be utilized. It is not appropriate for the BBQ, clothesline, or the patio chair. Among the options, the patio table seems to be the most logical place to put the milk, as it is a typical location to place food and beverages when you're spending time outdoors.

Next action: **put milk on patio table**

We can see that the model correctly ruled out BBQ, clothesline, and patio chair as incorrect locations. However, the model is not aware that there is a refrigerator (the correct place for the milk) in another room. Therefore, the model placing the milk on the patio table is an understandable mistake. After receiving feedback from the engine, the model's consideration is as follows:

Consideration: The milk is out of place; it should likely be put somewhere with refrigeration to prevent spoilage, such as a refrigerator, which could be inside the house.

Next action: take milk from patio table

We can see that the model makes reasonable inferences and successfully returns the milk to its correct location in the subsequent actions:

Action 2: **put milk on patio table** -> You put the milk on the patio table. Wrong position, you should put it somewhere else, maybe the other room. Action 3: take milk from patio table -> You take the milk from the patio table. Action 4: go west -> -= Kitchen -=You find yourself in a kitchen. Action 5: **insert milk into fridge** -> You put the milk into the fridge. Your score has just gone up by one point. Right position.

However, there are also instances where, after multiple unsuccessful attempts, the model **shifts its attention to other items**. For example, in the following case (from the second game at the hard level), the model gives up picking up the white polo shirt after two attempts:

Action 6: **put clean white polo shirt on sofa** -> You put the clean white polo shirt on the sofa. Wrong position, you should put it somewhere else, maybe the other room. Action 7: take clean white polo shirt from sofa -> You take the clean white polo shirt from the sofa. Action 8: go west

-> -= Bedroom -=You find yourself in a bedroom. Action 9: **insert clean white polo shirt into chest of drawers** -> You put the clean white polo shirt into the chest of drawers. Wrong position, you should put it somewhere else, maybe the other room. Action 10: go east -> -= Livingroom -=You are in a livingroom. Action 11: take brown suit from end table -> You take the brown suit from the end table.

Additionally, although rare, GPT-4 may also make mistakes by **repeating the same incorrect action twice**, as shown in the example below.

Action 8: go south -> -= Corridor -=Well, here we are in a corridor. Action 9: take clean white socks -> You pick up the clean white socks from the ground. Action 10: **insert clean white socks into shoe cabinet** -> You put the clean white socks into the shoe cabinet. Wrong position, you should put it somewhere else, maybe the other room. Action 11: take clean white socks from shoe cabinet -> You take the clean white socks from the shoe cabinet. Action 12: **insert clean white socks into shoe cabinet** -> You put the clean white socks into the shoe cabinet. Wrong position, you should put it somewhere else, maybe the other room.

Let's take a look at the model's consideration when making action 12 (repeatedly putting socks into the shoe cabinet).

Consideration: In the current scenario, you are in a corridor with various items and furniture pieces such as a shoe cabinet, hat rack, key holder, and umbrella stand, all of which are empty. You're carrying a pair of clean white socks. The socks typically belong in a bedroom or a designated storage area for clothing, but considering that the shoe cabinet is mentioned to be empty and is a suitable location for storing such items, placing clean socks there seems reasonable. Although it was previously stated that the shoe cabinet was not the right place for the clean white socks, given the context within the game, it may be that different types of items have specific correct placements; the previous location could have been incorrect for the other item but might be correct for the clean white socks. Additionally, the placement of items is often based on logical organization within a living space.

Next action: **insert clean white socks into shoe cabinet**

The model seems to be questioning the accuracy of the feedback provided by the game environment. Given that there is no other more suitable place to put the socks in the room besides the shoe cabinet, such questioning is understandable. The actual place where the socks belong is in another room, which again highlights the difficulty of moving items across rooms.

Apart from incorrect actions, there is a category of actions that potentially do harm to the model's performance, referred to as **redundant actions**. For instance, in the following case (also from the second game at the hard level):

Action 0: insert clean plaid polo shirt into wardrobe -> You put the clean plaid polo shirt into the wardrobe. Your score has just gone up by one point. Right position. Action 1: **close wardrobe** -> You close the wardrobe.

In this example, action 1 (close wardrobe) is a redundant action. Due to the wardrobe being closed, the option "insert into wardrobe" will not appear when picking up other clothing items subsequently. Closing anything in the TWC game environment does not yield any benefits (such as score improvement) and is entirely redundant. If we include a hint like "no need to close anything," the model's performance may improve further, but we do not want to introduce dataset-specific knowledge to harm the generality of our approach.

7. Conclusion

In our study, we proposed a method for implementing artificial intelligence agents in interactive fiction using large language models. Compared to traditional reinforcement learning methods, our approach is simple to implement and performs well. Due to the extensive training on massive datasets, GPT-4 have acquired basic knowledge about the human world. On the TextWorld Commonsense dataset, our agent achieved scores surpassing the best-performing baseline (reinforcement learning models trained on the training set) without fine-tuning. Specifically, using feedback-augmented prompts, GPT-4 perfectly completed all tasks in simple and medium-level game environments. In hard-level environments, our method's performance (0.70) also far surpassed the best baseline (0.57).

8. Bibliographical References

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

- Shreyas Basavatia, Shivam Ratnakar, and Keerthiram Murugesan. 2023. Complexworld: A large language model-based interactive fiction learning environment for text-based reinforcement learning agents. In *International Joint Conference on Artificial Intelligence*.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrike, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.
- Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097.
- Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2019. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer.
- Matan Haroush, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. 2018a. Learning how not to act in text-based games. In *6th International Conference on Learning Representations, ICLR 2018*. Publisher Copyright: © 6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings. All rights reserved.; 6th International Conference on Learning Representations, ICLR 2018 ; Conference date: 30-04-2018 Through 03-05-2018.
- Matan Haroush, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. 2018b. Learning how not to act in text-based games.
- Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2015. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.
- Keerthiram Murugesan, Mattia Atzeni, Pavan Kanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. 2021. Text-based rl agents with commonsense knowledge: New challenges, environments and baselines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9018–9027.
- Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.
- Chen Feng Tsai, Xiaochen Zhou, Sierra S Liu, Jing Li, Mo Yu, and Hongyuan Mei. 2023. Can large language models play text games well? current state-of-the-art and open questions. *arXiv preprint arXiv:2304.02868*.
- Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.
- Shunyu Yao, Karthik Narasimhan, and Matthew Hausknecht. 2021. Reading and acting while blindfolded: The need for semantics in text game agents. *arXiv preprint arXiv:2103.13552*.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.