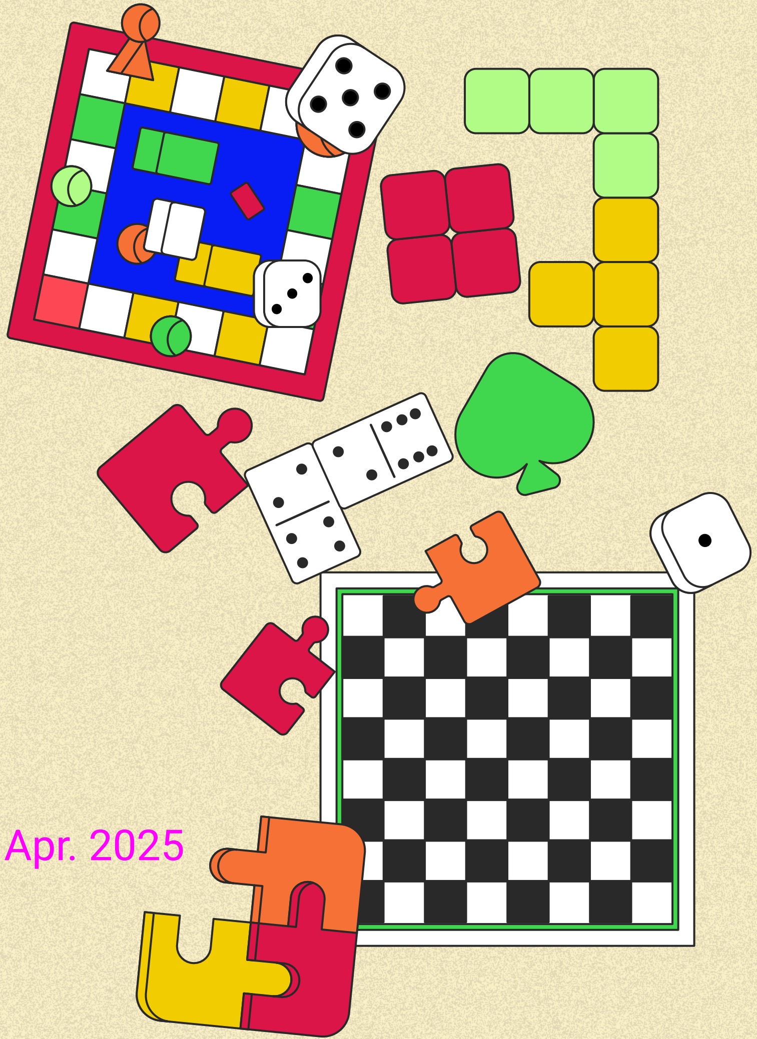
New to text-based interactive game for AI(LLM)



Author: Haonan Wang | Johns Hopkins University | Apr. 2025

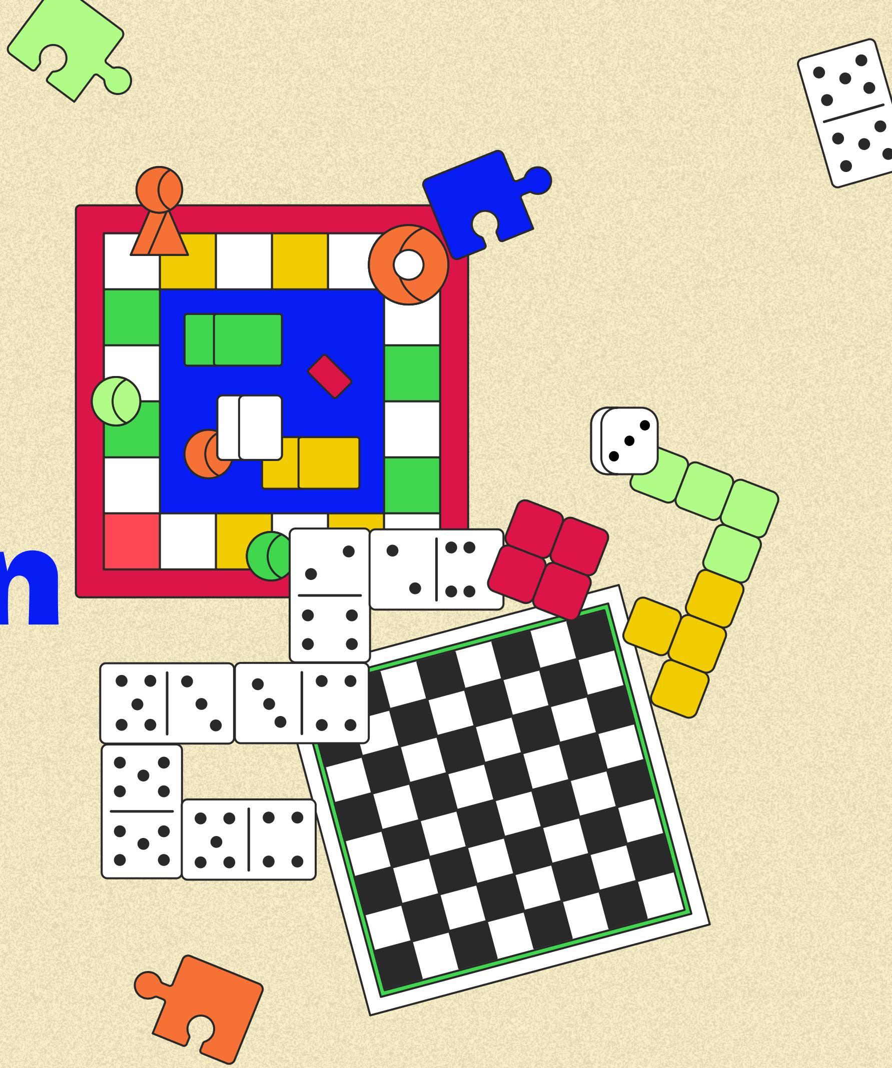
Guidance: Prof.Ziang Xiao

Table of contents

01

Introduction

Text Games: A Research Adventure!



Do you like playing these computer games?

AI(LLMs) is also worth a Text-based Virtual Games

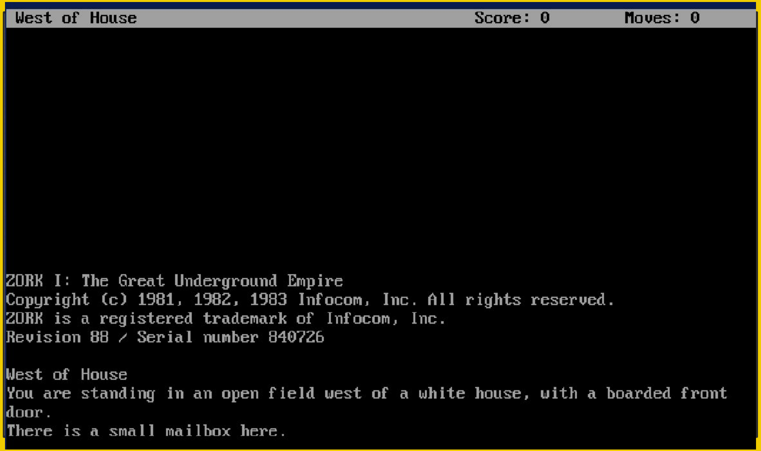
What Are Text Games? & Zork

Text Games (or, **Interactive Fiction**, or **Text-based Virtual Environments**) are interactive virtual worlds that users observe and act upon using words instead of pixels. They have a long history, with many of the earliest games (such as the ***Zork***) being text-based games.

Zork is a text adventure game **first** released in 1977 ,by developers Tim Anderson, Marc Blank, Bruce Daniels, and Dave Lebling for the PDP-10 mainframe computer.

https://classicreload.com/play/zork-i.html

Not video games, we are Text games.!!!

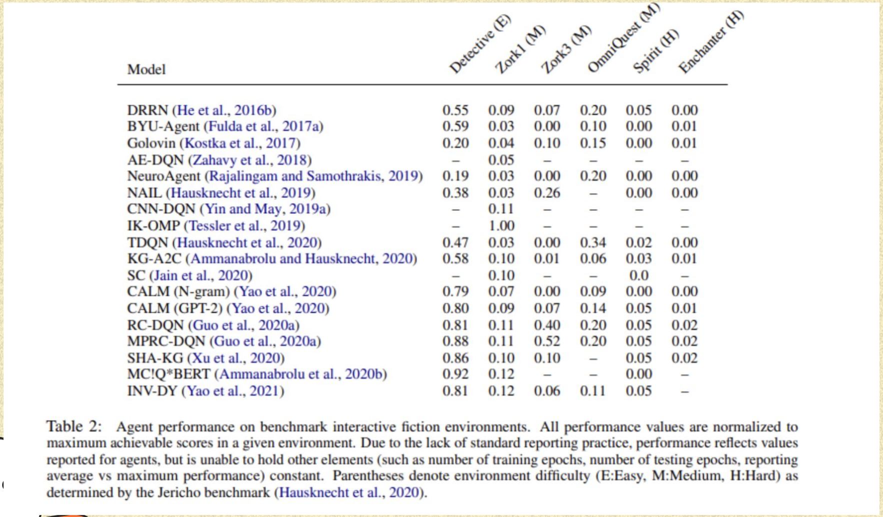


```
West of House                                       Score: 0      Moves: 0

ZORK I: The Great Underground Empire
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west of a white house, with a boarded front
door.
There is a small mailbox here.
```

https://www.youtube.com/watch?v=PWQDccL0aXM

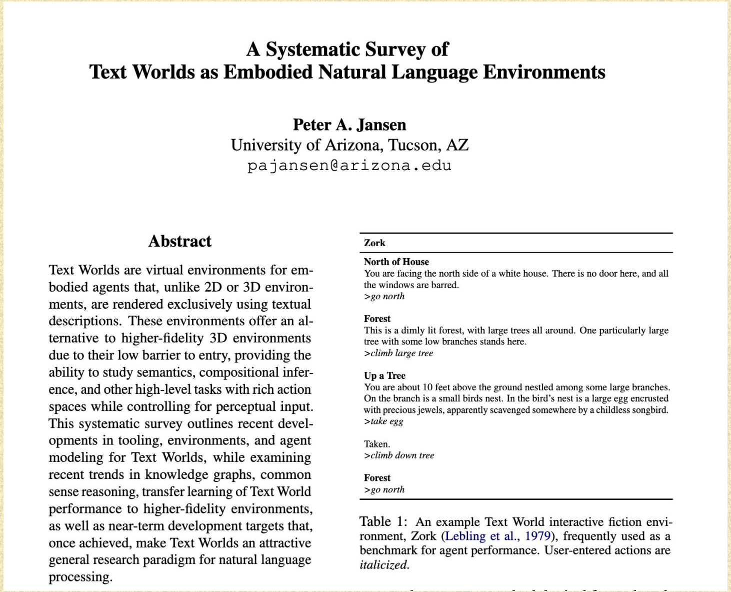
Text game & Researcher

For AI researchers, **Text Games** (or **Text-based Virtual Environments**) are an interesting text paradigm, and in that their requirement for **world knowledge** and **complex multi-step reasoning** to make AI system(like LLMs, LLM for Agent, etc) to solve.



Model	Detective (E)	Zork1 (M)	Zork3 (M)	OmniQuest (M)	Spirit (H)	Enchanter (H)
DRRN (He et al., 2016b)	0.55	0.09	0.07	0.20	0.05	0.00
BYU-Agent (Fulda et al., 2017a)	0.59	0.03	0.00	0.10	0.00	0.01
Golovin (Kostka et al., 2017)	0.20	0.04	0.10	0.15	0.00	0.01
AE-DQN (Zahavy et al., 2018)	–	0.05	–	–	–	–
NeuroAgent (Rajalingam and Samothrakis, 2019)	0.19	0.03	0.00	0.20	0.00	0.00
NAIL (Hausknecht et al., 2019)	0.38	0.03	0.26	–	0.00	0.00
CNN-DQN (Yin and May, 2019a)	–	0.11	–	–	–	–
IK-OMP (Tessler et al., 2019)	–	1.00	–	–	–	–
TDQN (Hausknecht et al., 2020)	0.47	0.03	0.00	0.34	0.02	0.00
KG-A2C (Ammanabrolu and Hausknecht, 2020)	0.58	0.10	0.01	0.06	0.03	0.01
SC (Jain et al., 2020)	–	0.10	–	–	0.0	–
CALM (N-gram) (Yao et al., 2020)	0.79	0.07	0.00	0.09	0.00	0.00
CALM (GPT-2) (Yao et al., 2020)	0.80	0.09	0.07	0.14	0.05	0.01
RC-DQN (Guo et al., 2020a)	0.81	0.11	0.40	0.20	0.05	0.02
MPRC-DQN (Guo et al., 2020a)	0.88	0.11	0.52	0.20	0.05	0.02
SHA-KG (Xu et al., 2020)	0.86	0.10	0.10	–	0.05	0.02
MC!Q*BERT (Ammanabrolu et al., 2020b)	0.92	0.12	–	–	0.00	–
INV-DY (Yao et al., 2021)	0.81	0.12	0.06	0.11	0.05	–

Table 2: Agent performance on benchmark interactive fiction environments. All performance values are normalized to maximum achievable scores in a given environment. Due to the lack of standard reporting practice, performance reflects values reported for agents, but is unable to hold other elements (such as number of training epochs, number of testing epochs, reporting average vs maximum performance) constant. Parentheses denote environment difficulty (E:Easy, M:Medium, H:Hard) as determined by the Jericho benchmark (Hausknecht et al., 2020).



A Systematic Survey of
Text Worlds as Embodied Natural Language Environments

Peter A. Jansen
University of Arizona, Tucson, AZ
pajansen@arizona.edu

Abstract

Text Worlds are virtual environments for embodied agents that, unlike 2D or 3D environments, are rendered exclusively using textual descriptions. These environments offer an alternative to higher-fidelity 3D environments due to their low barrier to entry, providing the ability to study semantics, compositional inference, and other high-level tasks with rich action spaces while controlling for perceptual input. This systematic survey outlines recent developments in tooling, environments, and agent modeling for Text Worlds, while examining recent trends in knowledge graphs, common sense reasoning, transfer learning of Text World performance to higher-fidelity environments, as well as near-term development targets that, once achieved, make Text Worlds an attractive general research paradigm for natural language processing.

Zork

North of House
You are facing the north side of a white house. There is no door here, and all the windows are barred.
>go north

Forest
This is a dimly lit forest, with large trees all around. One particularly large tree with some low branches stands here.
>climb large tree

Up a Tree
You are about 10 feet above the ground nestled among some large branches. On the branch is a small birds nest. In the bird's nest is a large egg encrusted with precious jewels, apparently scavenged somewhere by a childless songbird.
>take egg

Taken.
>climb down tree

Forest
>go north

Table 1: An example Text World interactive fiction environment, Zork (Lebling et al., 1979), frequently used as a benchmark for agent performance. User-entered actions are *italicized*.

In this 2022 paper, researchers find that: after years of active research by our AI community, **the best text game agents are only able to solve about 12% of Zork, even though it was authored in 1977**.

Why we need Text game?

Text games allow researchers to test complex reasoning, memory, planning, and language understanding — all without the **less computational burden** of 3D simulation.(They are **lightweight**, **reproducible**, and **easy** to deploy — requiring only text, not physics engines or 3D rendering.)

✅ **1.** Combine language understanding, planning, memory, and reasoning.

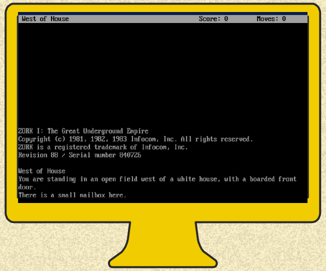
✅ **2.** Fully observable through text; no visual bias, Natural setting for reinforcement learning, LLMs learning..

✅ **3.** Text games provide an ideal balance: rich cognitive tasks, yet low operational cost.

You are in a dark cave. A torch lies here.

> take torch

You now have the torch.



```
West of House                                        Score: 0       Moves: 0

ZORK I: The Great Underground Empire
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west of a white house, with a boarded front
door.
There is a small mailbox here.
```

> go north

> pick up sword

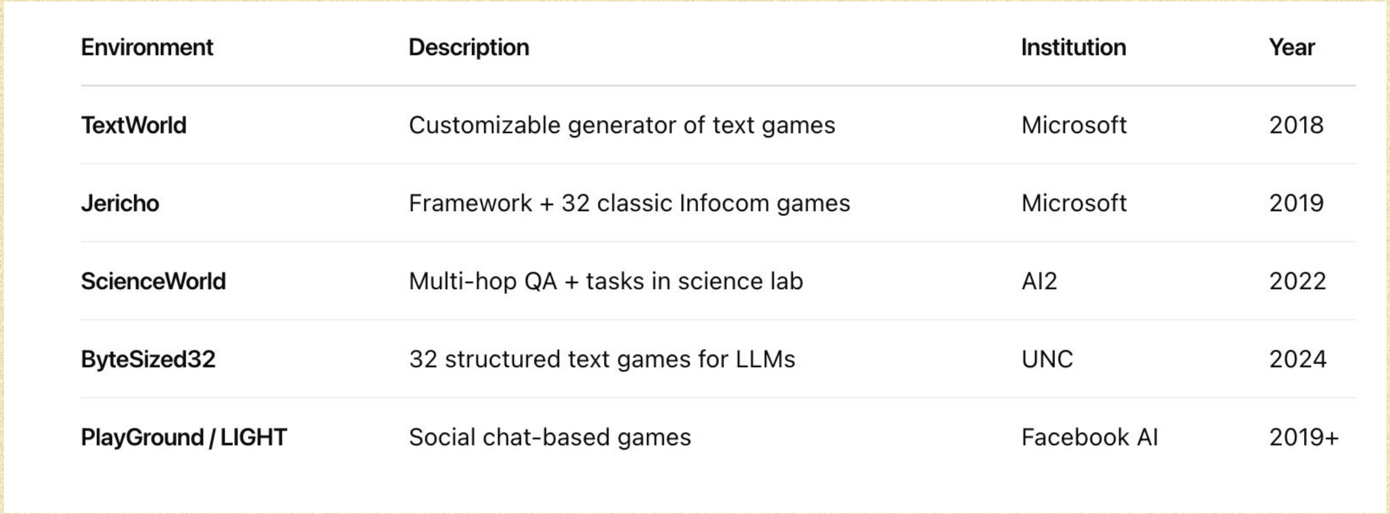
> talk to the wizard

In my view, believe that text-based environments have become a popular and important intermediate step before tackling full future embodied or real-word virtual environment agents.

How need us to do for research?

In NLP and Game theory field, human make text-based game is typically divided into two areas :

- **A. Interactive fiction environments**: which are essentially popular games from the past (such as Zork),
- **B. Purpose-built research environments**: that help teach or measure an agent's ability to perform (for example) specific kinds of common-sense or scientific reasoning.



Environment	Description	Institution	Year
TextWorld	Customizable generator of text games	Microsoft	2018
Jericho	Framework + 32 classic Infocom games	Microsoft	2019
ScienceWorld	Multi-hop QA + tasks in science lab	AI2	2022
ByteSized32	32 structured text games for LLMs	UNC	2024
PlayGround / LIGHT	Social chat-based games	Facebook AI	2019+

These environments allow researchers to explore LLM agents' abilities in planning, generalization, tool use, and interactive reasoning.

How need us to do for research?

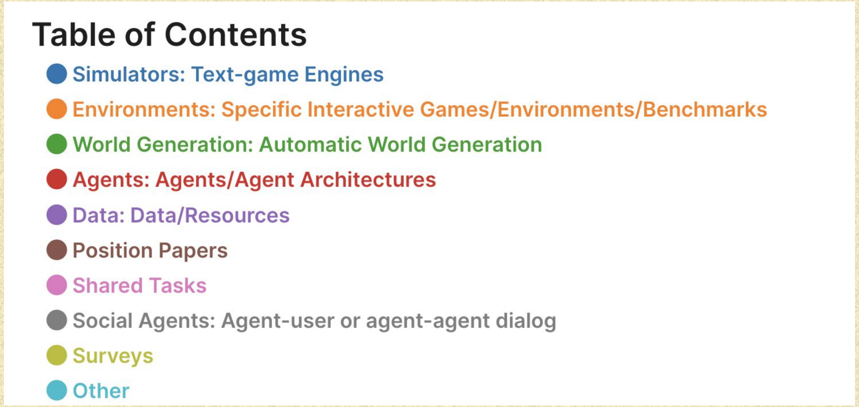


Table of Contents

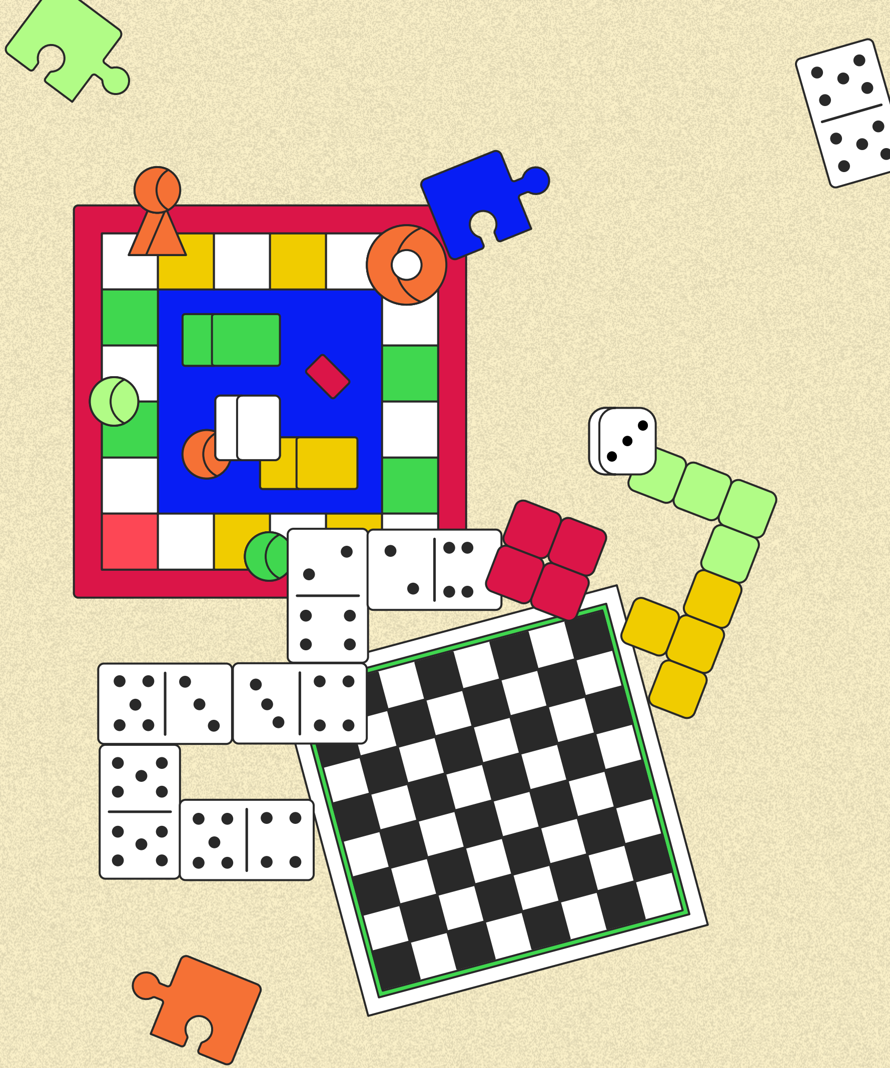
In specifically, many research directions are possible with text games.

- Some focus on building new environments or simulators to **test AI system (LLMs) ability**.
- **Creating virtual agents** that can explore, observe, reason, and interact over time to study intelligent behavior.
- 💰Although designing games takes effort, it's **still much cheaper than building 2D or 3D worlds**.

02

My work

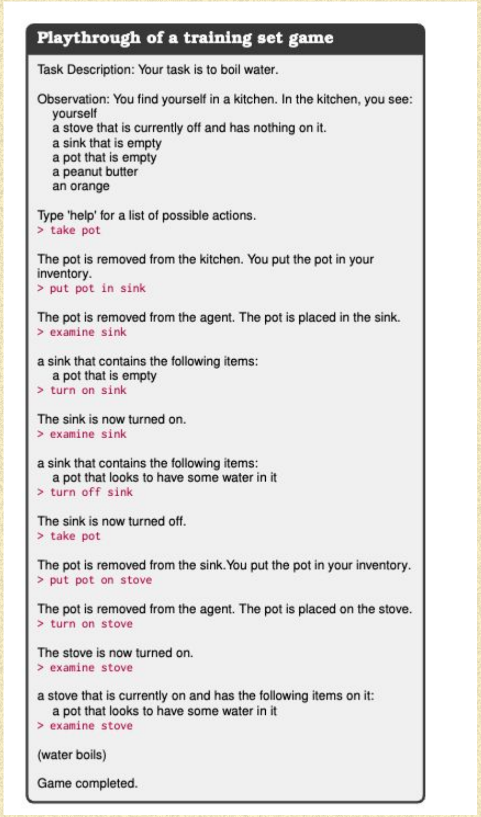
Let’s get started! Bytesized32 text game~



Previous work & Bytesized 32

Motivation:

- The ByteSized32 corpus offers 32 text games designed for **LLMs code generation**, exploring LLMs through interactive world modeling. Each game is written in Python and follows a common structure with task-specific rules and actions.
- To improve maintainability and future extensibility, we refactored the entire codebase to abstract shared components, reduce redundancy, and streamline logic across games.
- This modular redesign supports easier experimentation, faster development.



Playthrough of a training set game

Task Description: Your task is to boil water.

Observation: You find yourself in a kitchen. In the kitchen, you see:
yourself
a stove that is currently off and has nothing on it.
a sink that is empty
a pot that is empty
a peanut butter
an orange

Type 'help' for a list of possible actions.
```
> take pot
```
The pot is removed from the kitchen. You put the pot in your inventory.
```
> put pot in sink
```
The pot is removed from the agent. The pot is placed in the sink.
```
> examine sink
```
a sink that contains the following items:
a pot that is empty
```
> turn on sink
```
The sink is now turned on.
```
> examine sink
```
a sink that contains the following items:
a pot that looks to have some water in it
```
> turn off sink
```
The sink is now turned off.
```
> take pot
```
The pot is removed from the sink. You put the pot in your inventory.
```
> put pot on stove
```
The pot is removed from the agent. The pot is placed on the stove.
```
> turn on stove
```
The stove is now turned on.
```
> examine stove
```
a stove that is currently on and has the following items on it:
a pot that looks to have some water in it
```
> examine stove
```
(water boils)

Game completed.

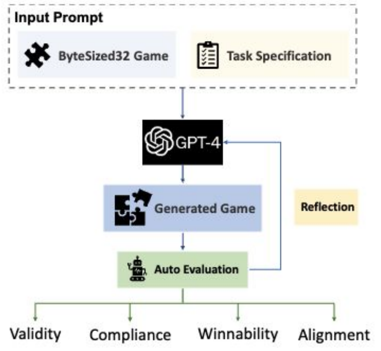
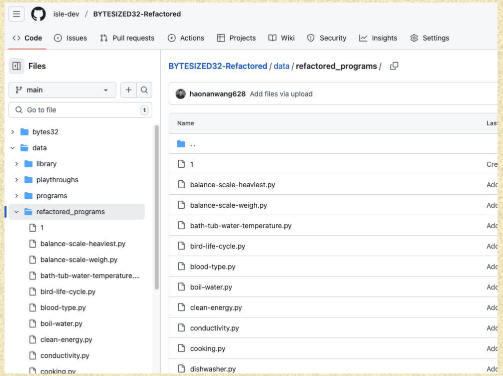


Figure 1: An overview of our text game generation and evaluation process. The model, here GPT-4, generates a game using in-context learning with a prompt consisting



bytes32
data
library
playthroughs
programs
refactored_programs
1
balance-scale-heaviest.py
balance-scale-weigh.py
bath-tub-water-temperature....
bird-life-cycle.py
blood-type.py
boil-water.py
clean-energy.py
conductivity.py
cooking.py

BYTESIZED32-Refactored / data / refactored_programs /

haonanwang628 Add files via upload

Name	Las
..	
1	Cre
balance-scale-heaviest.py	Add
balance-scale-weigh.py	Add
bath-tub-water-temperature.py	Add
bird-life-cycle.py	Add
blood-type.py	Add
boil-water.py	Add
clean-energy.py	Add
conductivity.py	Add
cooking.py	Add
dishwasher.py	Add

Code: https://github.com/isle-dev/BYTESIZED32-Refactored

Paper: https://arxiv.org/abs/2305.14879

Previous work & Difficulties

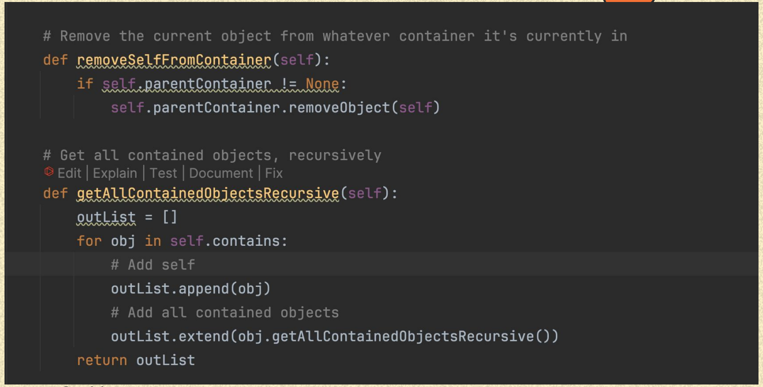
❗ Too much duplicated code

Each game redefined similar classes like `TextGame` and `GameObject`, leading to redundancy and high maintenance effort.

❗ Bloated if-else logic for action handling

Actions were handled with long if-elif chains, making the code hard to read, extend, and debug.

❗ Same meaning, different code



```
# Remove the current object from whatever container it's currently in
def removeSelfFromContainer(self):
    if self.parentContainer != None:
        self.parentContainer.removeObject(self)

# Get all contained objects, recursively
Edit | Explain | Test | Document | Fix
def getAllContainedObjectsRecursive(self):
    outList = []
    for obj in self.contains:
        # Add self
        outList.append(obj)
        # Add all contained objects
        outList.extend(obj.getAllContainedObjectsRecursive())
    return outList
```

Some games use different names or code structures to do the same thing. Even though the logic is similar, it looks very different, so I had to read carefully to understand it. **Example:**

One game uses `getAllObjects()` and another uses `collect_items()` — but they mean the same thing!

Code: https://github.com/isle-dev/BYTESIZED32-Refactored

Paper: https://arxiv.org/abs/2305.14879

Previous work & Results

1. Modular Design

→ Extracted shared logic (e.g., GameObject.py, Text Game.py) into a reusable library.

📦 **Result**: Simplified game code, better reuse.

2. Simplified Action Handling

→ Replaced messy `if-elif` chains with `action_map` dictionaries.

⚙️ **Result:** Easier to add or change actions.

3. Motivation: We need Optimized Recursion

→ Used list comprehensions for recursive object handling.

🔁 **Result: Cleaner and faster code.**

4. Improved Descriptions

→ Unified how objects and containers are described.

🧾 **Result: More readable and consistent output.**

5. Centralized Execution

→ All games now use the same entry point and loop.

🚀 **Result: Simplified testing and standardized execution.**

✅ **Outcome:**

💡 1. Reduced code about half of old games lines redundancy across 32 games

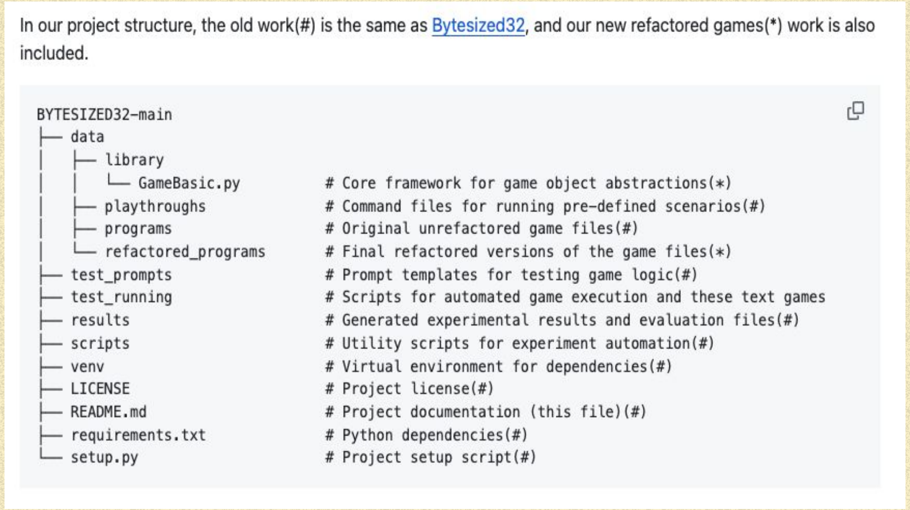
📦 2. Built a reusable library **GameBasic.py**

🎯 3. Enabled rapid prototyping and expanding of new text-based games

Code: https://github.com/isle-dev/BYTESIZED32-Refactored

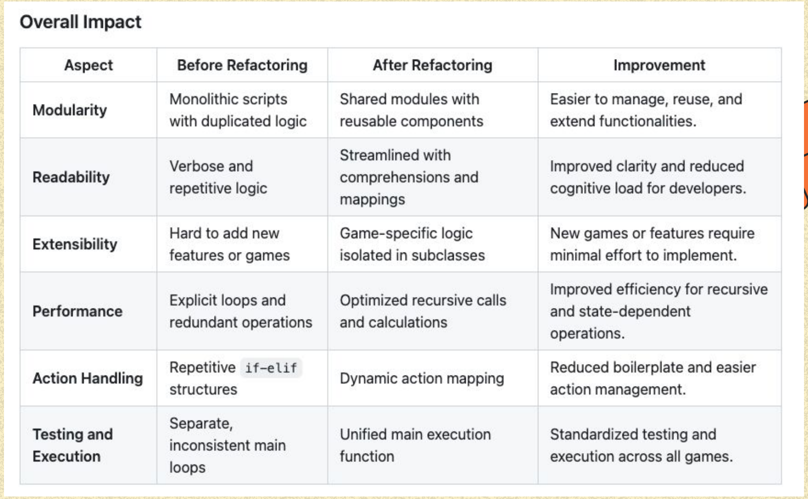
Paper: https://arxiv.org/abs/2305.14879

Previous work & Results



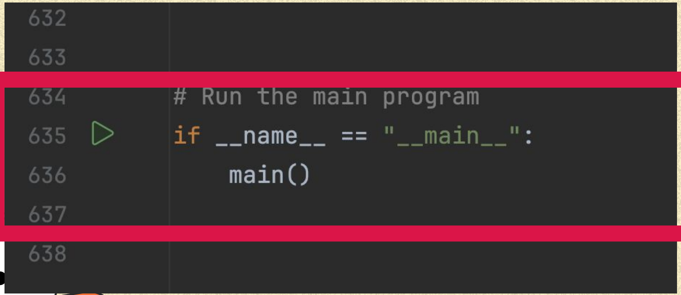
In our project structure, the old work(#) is the same as Bytesized32, and our new refactored games(*) work is also included.

```
BYTESIZED32-main
├── data
│   ├── library
│   │   └── GameBasic.py        # Core framework for game object abstractions(*)
│   ├── playthroughs            # Command files for running pre-defined scenarios(#)
│   ├── programs                # Original unrefactored game files(#)
│   └── refactored_programs     # Final refactored versions of the game files(*)
├── test_prompts                # Prompt templates for testing game logic(#)
├── test_running                # Scripts for automated game execution and these text games
├── results                     # Generated experimental results and evaluation files(#)
├── scripts                     # Utility scripts for experiment automation(#)
├── venv                        # Virtual environment for dependencies(#)
├── LICENSE                     # Project license(#)
├── README.md                   # Project documentation (this file)(#)
├── requirements.txt            # Python dependencies(#)
└── setup.py                    # Project setup script(#)
```

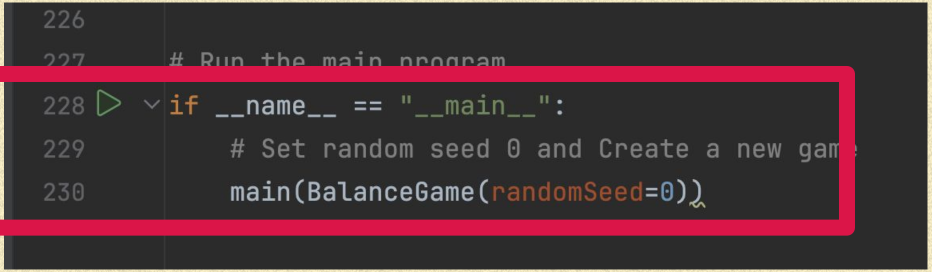


Overall Impact

Aspect	Before Refactoring	After Refactoring	Improvement
Modularity	Monolithic scripts with duplicated logic	Shared modules with reusable components	Easier to manage, reuse, and extend functionalities.
Readability	Verbose and repetitive logic	Streamlined with comprehensions and mappings	Improved clarity and reduced cognitive load for developers.
Extensibility	Hard to add new features or games	Game-specific logic isolated in subclasses	New games or features require minimal effort to implement.
Performance	Explicit loops and redundant operations	Optimized recursive calls and calculations	Improved efficiency for recursive and state-dependent operations.
Action Handling	Repetitive `if-elif` structures	Dynamic action mapping	Reduced boilerplate and easier action management.
Testing and Execution	Separate, inconsistent main loops	Unified main execution function	Standardized testing and execution across all games.



```
632
633
634  # Run the main program
635  if __name__ == "__main__":
636      main()
637
638
```



```
226
227  # Run the main program
228  if __name__ == "__main__":
229      # Set random seed 0 and Create a new gam
230      main(BalanceGame(randomSeed=0))
```

✅ **Outcome:**

💡 637—----230(reduce lines of code every games)

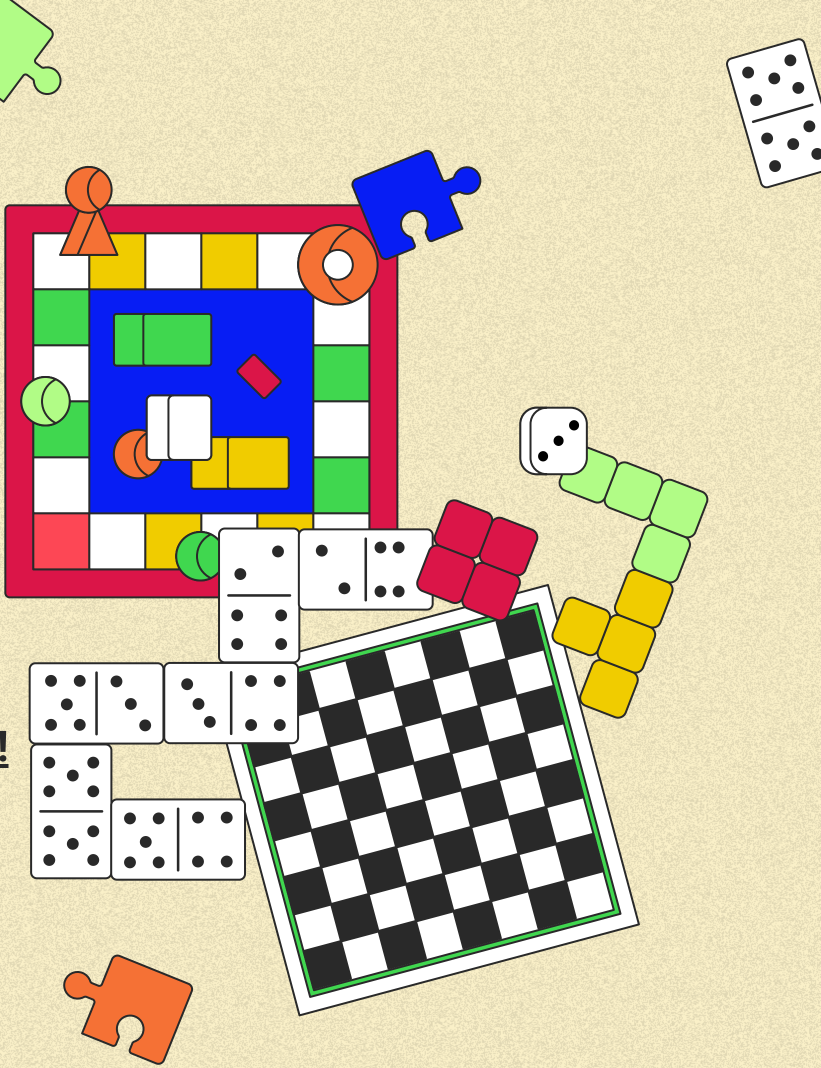
Code: https://github.com/isle-dev/BYTESIZED32-Refactored

Paper: https://arxiv.org/abs/2305.14879

03

And LLMs

A lot of my understanding and thinking with LLMs!

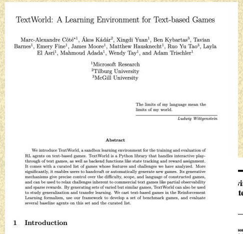


LLMs & Code generation

💡 **Idea 1:** How can LLMs generate our want own Text-based Games?

✅ :Through the process of Bytesized 32 text games project, we explore how **GPT-4 models** can generate runnable, reasoning-oriented text games.

R1. **Generate <u>specific code</u>** for AI for science in specific field?



TextWorld: A Learning Environment for Text-based Games

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Ruo Yu Tao, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler

The limits of my language mean the limits of my world.
Ludwig Wittgenstein

Abstract

1 Introduction

DiscoveryWorld: A Virtual Environment for Developing and Evaluating Automated Scientific Discovery Agents

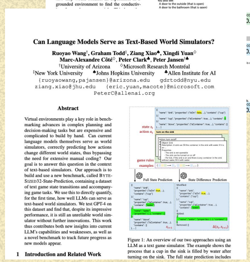
Abstract

1 Introduction

ScienceWorld: Is your Agent Smarter than a 5th Grader?

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, Prithviraj Ammanabrolu

Abstract



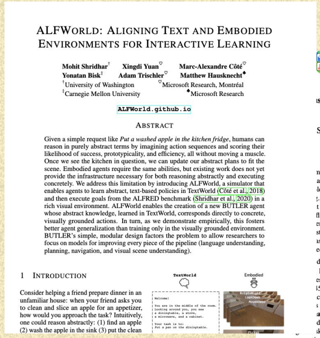
Can Language Models Serve as Text-Based World Simulators?

Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, Peter Jansen

Abstract

1 Introduction and Related Work

R2 **Understand** the <u>code structure</u>?



ALFWORLD: ALIGNING TEXT AND EMBODIED ENVIRONMENTS FOR INTERACTIVE LEARNING

Mohit Shridhar[†] Xingdi Yuan[♡] Marc-Alexandre Côté[♡]
Yonatan Bisk[‡] Adam Trischler[♡] Matthew Hausknecht[♣]
[†]University of Washington [♡]Microsoft Research, Montréal
[‡]Carnegie Mellon University [♣]Microsoft Research

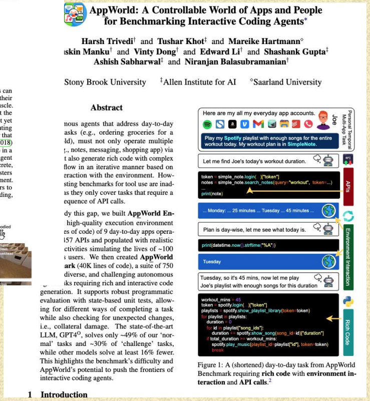
ALFWorld.github.io

ABSTRACT

Given a simple request like *Put a washed apple in the kitchen fridge*, humans can reason in purely abstract terms by imagining action sequences and scoring their likelihood of success, prototypicality, and efficiency, all without moving a muscle. Once we see the kitchen in question, we can update our abstract plans to fit the scene. Embodied agents require the same abilities, but existing work does not yet provide the infrastructure necessary for both reasoning abstractly and executing concretely. We address this limitation by introducing ALFWorld, a simulator that enables agents to learn abstract, text-based policies in TextWorld (Côté et al., 2018) and then execute goals from the ALFRED benchmark (Shridhar et al., 2020) in a rich visual environment. ALFWorld enables the creation of a new BUTLER agent whose abstract knowledge, learned in TextWorld, corresponds directly to concrete, visually grounded actions. In turn, as we demonstrate empirically, this fosters better agent generalization than training only in the visually grounded environment. BUTLER's simple, modular design factors the problem to allow researchers to focus on models for improving every piece of the pipeline (language understanding, planning, navigation, and visual scene understanding).

1 INTRODUCTION

Consider helping a friend prepare dinner in an unfamiliar house: when your friend asks you to clean and slice an apple for an appetizer, how would you approach the task? Intuitively, one could reason abstractly: (1) find an apple (2) wash the apple in the sink (3) put the clean



AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents

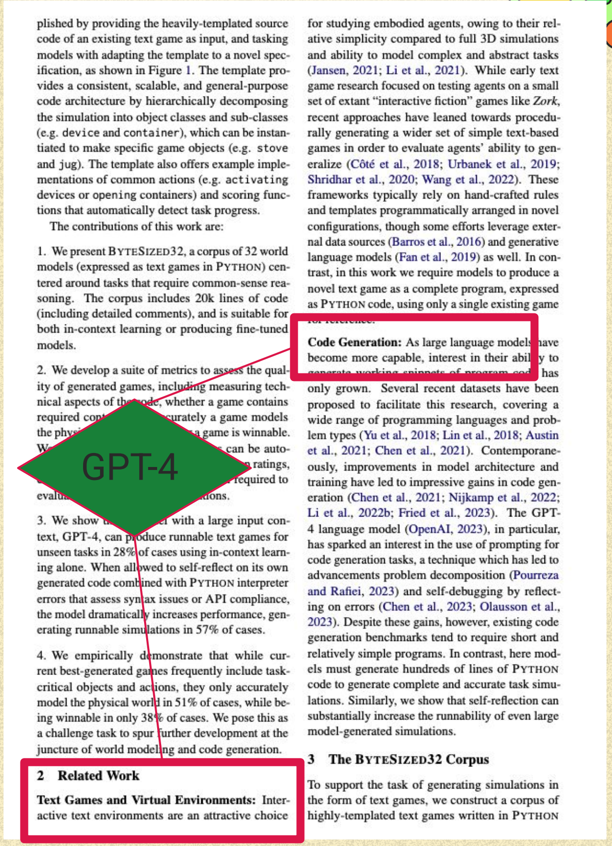
Harsh Trivedi and Tushar Khot and Mareike Hartmann
Manku and Vinty Dong and Edward Li and Shashank Gupta
Ashish Sabharwal and Niranjan Balasubramanian

Stony Brook University Allen Institute for AI Saarland University

Abstract

Figure 1: A (shortened) day-to-day task from AppWorld Benchmark requiring **rich code** with **environment interaction** and **API calls**.

1 Introduction



plished by providing the heavily-templated source code of an existing text game as input, and tasking models with adapting the template to a novel specification, as shown in Figure 1. The template provides a consistent, scalable, and general-purpose code architecture by hierarchically decomposing the simulation into object classes and sub-classes (e.g. `device` and `container`), which can be instantiated to make specific game objects (e.g. `stove` and `jug`). The template also offers example implementations of common actions (e.g. `activating` devices or `opening` containers) and scoring functions that automatically detect task progress.

The contributions of this work are:

1. We present BYTESIZED32, a corpus of 32 world models (expressed as text games in PYTHON) centered around tasks that require common-sense reasoning. The corpus includes 20k lines of code (including detailed comments), and is suitable for both in-context learning or producing fine-tuned models.

2. We develop a suite of metrics to assess the quality of generated games, including measuring technical aspects of the code, whether a game contains required co… …curately a game models the phys… …a game is winnable. W… …can be automa… …ratings, … …required to evalu… …ions.

3. We show … with a large input context, GPT-4, can produce runnable text games for unseen tasks in 28% of cases using in-context learning alone. When allowed to self-reflect on its own generated code combined with PYTHON interpreter errors that assess syntax issues or API compliance, the model dramatically increases performance, generating runnable simulations in 57% of cases.

4. We empirically demonstrate that while current best-generated games frequently include task-critical objects and actions, they only accurately model the physical world in 51% of cases, while being winnable in only 38% of cases. We pose this as a challenge task to spur further development at the juncture of world modeling and code generation.

2 Related Work

Text Games and Virtual Environments: Interactive text environments are an attractive choice for studying embodied agents, owing to their relative simplicity compared to full 3D simulations and ability to model complex and abstract tasks (Jansen, 2021; Li et al., 2021). While early text game research focused on testing agents on a small set of extant "interactive fiction" games like *Zork*, recent approaches have leaned towards procedurally generating a wider set of simple text-based games in order to evaluate agents' ability to generalize (Côté et al., 2018; Urbanek et al., 2019; Shridhar et al., 2020; Wang et al., 2022). These frameworks typically rely on hand-crafted rules and templates programmatically arranged in novel configurations, though some efforts leverage external data sources (Barros et al., 2016) and generative language models (Fan et al., 2019) as well. In contrast, in this work we require models to produce a novel text game as a complete program, expressed as PYTHON code, using only a single existing game for reference.

Code Generation: As large language models have become more capable, interest in their ability to generate working snippets of program code has only grown. Several recent datasets have been proposed to facilitate this research, covering a wide range of programming languages and problem types (Yu et al., 2018; Lin et al., 2018; Austin et al., 2021; Chen et al., 2021). Contemporaneously, improvements in model architecture and training have led to impressive gains in code generation (Chen et al., 2021; Nijkamp et al., 2022; Li et al., 2022b; Fried et al., 2023). The GPT-4 language model (OpenAI, 2023), in particular, has sparked an interest in the use of prompting code generation tasks, a technique which has led to advancements problem decomposition (Pourreza and Rafiei, 2023) and self-debugging by reflecting on errors (Chen et al., 2023; Olausson et al., 2023). Despite these gains, however, existing code generation benchmarks tend to require short and relatively simple programs. In contrast, here models must generate hundreds of lines of PYTHON code to generate complete and accurate task simulations. Similarly, we show that self-reflection can substantially increase the runnability of even large model-generated simulations.

3 The BYTESIZED32 Corpus

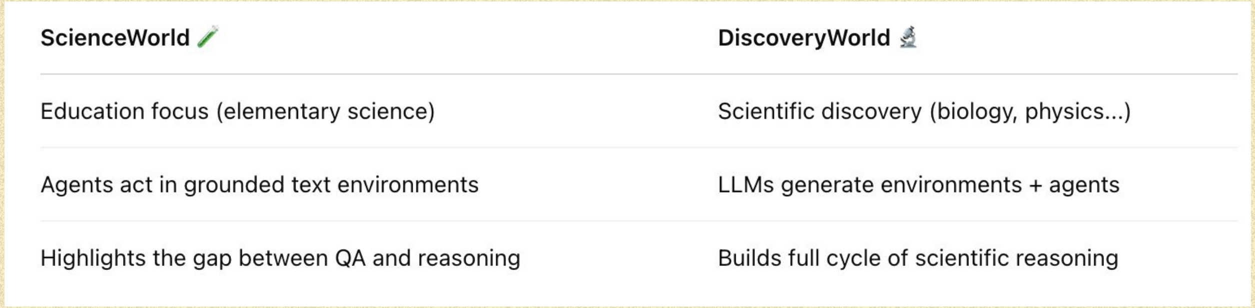
To support the task of generating simulations in the form of text games, we construct a corpus of highly-templated text games written in PYTHON

GPT-4

LLMs & Code generation

R1.**Generate <u>specific code</u> for AI for science in specific field?**

✅ **Background:** Recent research (DiscoveryWorld, ScienceWorld,etc.), it include a new future research direction to explore the LLMs generate multiple field to simulate reasoning tasks in science and education.



ScienceWorld 🧪	DiscoveryWorld 🔬
Education focus (elementary science)	Scientific discovery (biology, physics...)
Agents act in grounded text environments	LLMs generate environments + agents
Highlights the gap between QA and reasoning	Builds full cycle of scientific reasoning

💡 **My thinking**: Can we build a **generalizable code structure** for LLM-generated environments across different domains like social conversation agent or engineering field ?

🔍**My Hypothesis:**

We may need a **<u>unified, modular code structure</u>**
— a shared "skeleton" that LLMs can follow to generate environments consistently across and adapt for more and more domains.

ScienceWorld: Is your Agent Smarter than a 5th Grader?

Ruoyao Wang*‡, Peter Jansen*‡, Marc-Alexandre Côté♣, Prithviraj Ammanabrolu◇
‡University of Arizona, Tucson, AZ ♣Microsoft Research Montréal
◇Allen Institute for AI, Seattle, WA
{ruoyaowang,pajansen}@arizona.edu
macote@microsoft.com, raja@allenai.org

Abstract

We present ScienceWorld, a benchmark to test agents' scientific reasoning abilities in a new interactive text environment at the level of a standard elementary school science curriculum. Despite the transformer-based progress seen in question-answering and scientific text processing, we find that current models cannot reason about or explain learned science concepts in novel contexts. For instance, models can easily answer *what* the conductivity of a known material is but struggle when asked *how* they would conduct an experiment in a grounded environment to find the conductivity of an unknown material. This begs the question of whether current models are simply retrieving answers by way of seeing a large number of similar examples or if they have learned to reason about concepts in a reusable manner. We hypothesize that agents need to be grounded in interactive environments to achieve such reasoning capabilities. Our experiments provide empirical evidence supporting this hypothesis—showing that a 1.5 million parameter agent trained interactively for 100k steps outperforms a 11 billion parameter model statically trained for scientific question-answering and reasoning from millions of expert demonstrations.[12]

1 Introduction

Question answering (QA) has seen rapid progress recently. Standardized elementary and middle

ScienceWorld Task 3 (Test Electrical Conductivity Subtask)

Task Description
Your task is to determine if a metal fork is electrically conductive. The metal fork is located around the kitchen. First, focus on the metal fork. If conductive, place it in the blue box. If nonconductive, place it in the green box.

Agent starts in a random location, navigates to kitchen

Kitchen
This room is called the kitchen. In it, you see:
a fridge. The freezer door is closed.
a cupboard. The cupboard door is closed.
a table. On the table is: a glass cup (containing nothing).
a stove, which is turned off. On the stove is: nothing.
a sink, which is turned off. In the sink is: nothing.
a thermometer, currently reading a temperature of 10 degrees celsius
a metal fork
a counter. On the counter is: a drawer, a bowl (containing an orange, a banana, a potato, a red apple).
You also see:
A door to the outside (that is open)
A door to the bathroom (that is open)
A door to the hallway (that is open)

In room with task object — SCORE 0.1

Action: ***focus on metal fork***
You focus on the metal fork.

SCORE 0.5

Action: ***pick up metal fork***
You move the metal fork to the inventory.

Agent navigates to room with electrical components

Workshop
This room is called the workshop. In it, you see:
a electric buzzer, which is off
a electric motor, which is off
a solar panel, which is off
a table. On the table is: a blue wire, a battery, a red light bulb, which is off, a black wire, a switch, which is off, a orange wire.
a blue answer box
a green answer box
You also see:
A door to the hallway (that is open)

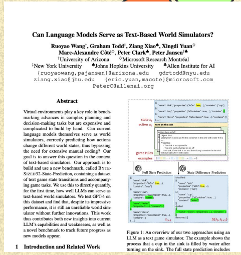
In room with components — SCORE 0.6

Action: ***connect battery cathode to orange wire terminal 1***
cathode on battery is now connected to terminal 1 on orange wire.

Action: ***connect metal fork terminal 1 to cathode in red light bulb***
terminal 1 on metal fork is now connected to cathode on red light bulb.

Agent continues wiring a circuit containing the metal fork

SCORE 0.7



Can Language Models Serve as Text-Based World Simulators?

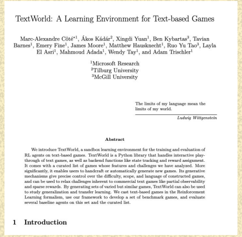
Ruoyao Wang†, Graham Todd‡, Ziang Xiao♠, Xingdi Yuan◇
Marc-Alexandre Côté◇, Peter Clark♣, Peter Jansen†♣
†University of Arizona ◇Microsoft Research Montréal
‡New York University ♠Johns Hopkins University ♣Allen Institute for AI
{ruoyaowang,pajansen}@arizona.edu gdrtodd@nyu.edu
ziang.xiao@jhu.edu {eric.yuan,macote}@microsoft.com
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 ByteSized32-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

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



TextWorld: A Learning Environment for Text-based Games

Marc-Alexandre Côté*[1], Ákos Kádár[2], Xingdi Yuan[1], Ben Kybartas[3], Tavian Barnes[1], Emery Fine[1], James Moore[1], Matthew Hausknecht[1], Ruo Yu Tao[3], Layla El Asri[1], Mahmoud Adada[1], Wendy Tay[1], and Adam Trischler[1]

[1]Microsoft Research
[2]Tilburg University
[3]McGill University

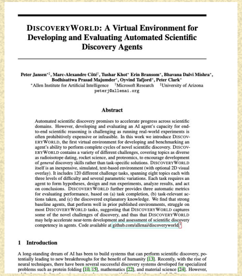
The limits of my language mean the limits of my world.

Ludwig Wittgenstein

Abstract

We introduce TextWorld, a sandbox learning environment for the training and evaluation of RL agents on text-based games. TextWorld is a Python library that handles interactive play-through of text games, as well as backend functions like state tracking and reward assignment. It comes with a curated list of games whose features and challenges we have analyzed. More significantly, it enables users to handcraft or automatically generate new games. Its generative mechanisms give precise control over the difficulty, scope, and language of constructed games, and can be used to relax challenges inherent to commercial text games like partial observability and sparse rewards. By generating sets of varied but similar games, TextWorld can also be used to study generalization and transfer learning. We cast text-based games in the Reinforcement Learning formalism, use our framework to develop a set of benchmark games, and evaluate several baseline agents on this set and the curated list.

1 Introduction



DiscoveryWorld: A Virtual Environment for Developing and Evaluating Automated Scientific Discovery Agents

Peter Jansen*‡, Marc-Alexandre Côté†, Tushar Khot* Erin Bransom*, Bhavana Dalvi Mishra*, Bodhisattwa Prasad Majumder*, Oyvind Tafjord*, Peter Clark*
*Allen Institute for Artificial Intelligence †Microsoft Research ‡University of Arizona
peterj@allenai.org

Abstract

Automated scientific discovery promises to accelerate progress across scientific domains. However, developing and evaluating an AI agent's capacity for end-to-end scientific reasoning is challenging as running real-world experiments is often prohibitively expensive or infeasible. In this work we introduce DiscoveryWorld, the first virtual environment for developing and benchmarking an agent's ability to perform complete cycles of novel scientific discovery. DiscoveryWorld contains a variety of different challenges, covering topics as diverse as radioisotope dating, rocket science, and proteomics, to encourage development of *general* discovery skills rather than task-specific solutions. DiscoveryWorld itself is an inexpensive, simulated, text-based environment (with optional 2D visual overlay). It includes 120 different challenge tasks, spanning eight topics each with three levels of difficulty and several parametric variations. Each task requires an agent to form hypotheses, design and run experiments, analyze results, and act on conclusions. DiscoveryWorld further provides three automatic metrics for evaluating performance, based on (a) task completion, (b) task-relevant actions taken, and (c) the discovered explanatory knowledge. We find that strong baseline agents, that perform well in prior published environments, struggle on most DiscoveryWorld tasks, suggesting that DiscoveryWorld captures some of the novel challenges of discovery, and thus that DiscoveryWorld may help accelerate near-term development and assessment of scientific discovery competency in agents. Code available at github.com/allenai/discoveryworld.[1]

1 Introduction

A long-standing dream of AI has been to build systems that can perform scientific discovery, potentially leading to new breakthroughs for the benefit of humanity [13]. Recently, with the rise of neural techniques, there have been several successful discovery systems developed for specialized problems such as protein folding [10, 15], mathematics [22], and material science [24]. However,

https://arxiv.org/pdf/2406.06769

https://arxiv.org/pdf/2305.14879

https://arxiv.org/pdf/1806.11532

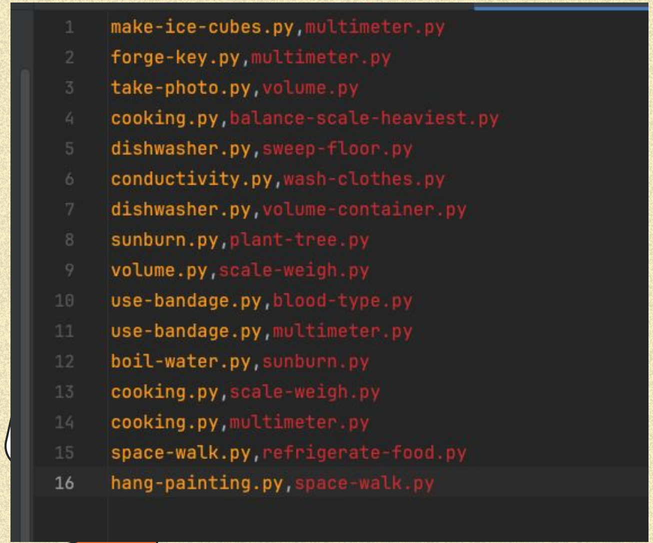
https://arxiv.org/pdf/2203.07540

LLMs & Code generation

R2. **Understand** the code structure?

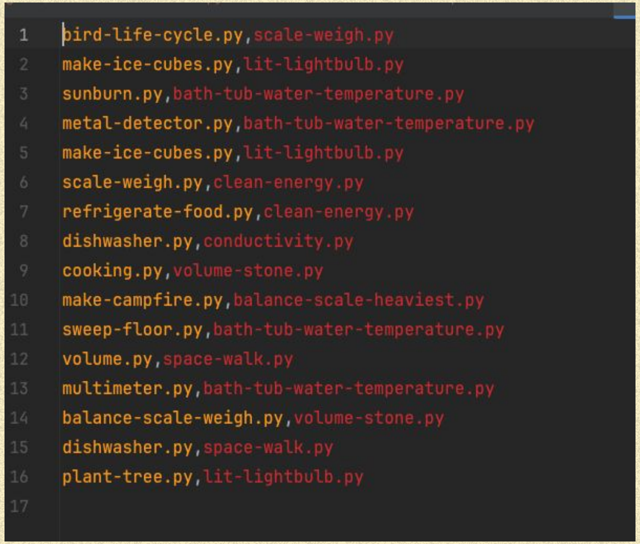
✅ **Background:** Recent research (Bytesized 32 ,etc.), it uses analogical prompt selection — pairing a target game with a semantically or functionally similar game based on shared actions. This **prompt design** does not rely on exact matches, but rather on functional or thematic similarity — helping LLMs generalize through analogical (compare,etc.) generation.

```
[target_game].py, [prompt_game].py
```



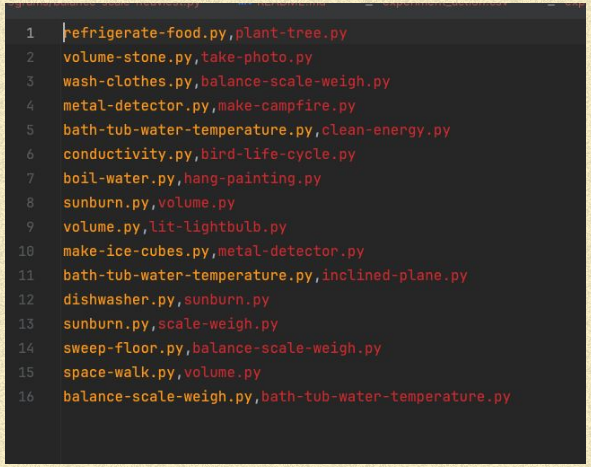
```
make-ice-cubes.py,multimeter.py
forge-key.py,multimeter.py
take-photo.py,volume.py
cooking.py,balance-scale-heaviest.py
dishwasher.py,sweep-floor.py
conductivity.py,wash-clothes.py
dishwasher.py,volume-container.py
sunburn.py,plant-tree.py
volume.py,scale-weigh.py
use-bandage.py,blood-type.py
use-bandage.py,multimeter.py
boil-water.py,sunburn.py
cooking.py,scale-weigh.py
cooking.py,multimeter.py
space-walk.py,refrigerate-food.py
hang-painting.py,space-walk.py
```

Action.csv



```
bird-life-cycle.py,scale-weigh.py
make-ice-cubes.py,lit-lightbulb.py
sunburn.py,bath-tub-water-temperature.py
metal-detector.py,bath-tub-water-temperature.py
make-ice-cubes.py,lit-lightbulb.py
scale-weigh.py,clean-energy.py
refrigerate-food.py,clean-energy.py
dishwasher.py,conductivity.py
cooking.py,volume-stone.py
make-campfire.py,balance-scale-heaviest.py
sweep-floor.py,bath-tub-water-temperature.py
volume.py,space-walk.py
multimeter.py,bath-tub-water-temperature.py
balance-scale-weigh.py,volume-stone.py
dishwasher.py,space-walk.py
plant-tree.py,lit-lightbulb.py

```

Distractor.csv



```
refrigerate-food.py,plant-tree.py
volume-stone.py,take-photo.py
wash-clothes.py,balance-scale-weigh.py
metal-detector.py,make-campfire.py
bath-tub-water-temperature.py,clean-energy.py
conductivity.py,bird-life-cycle.py
boil-water.py,hang-painting.py
sunburn.py,volume.py
volume.py,lit-lightbulb.py
make-ice-cubes.py,metal-detector.py
bath-tub-water-temperature.py,inclined-plane.py
dishwasher.py,sunburn.py
sunburn.py,scale-weigh.py
sweep-floor.py,balance-scale-weigh.py
space-walk.py,volume.py
balance-scale-weigh.py,bath-tub-water-temperature.py
```

Object.csv

LLMs & Code generation

R2.**Understand** the code structure?

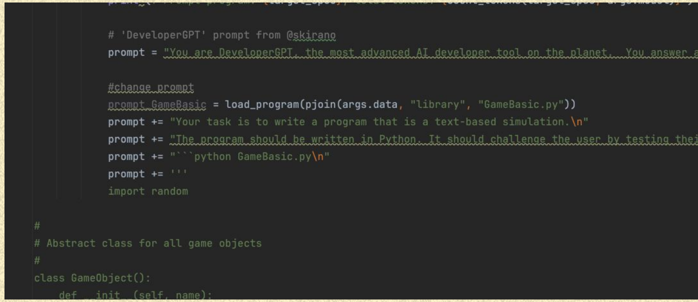
In conclusion, we can define Bytesized 32 method named **Compare Prompt Selection: Teaching by Similarity**

💡 **My thinking**:

1.Can we provide better **structured prompts** design to guide LLMs?(Bytesized 32 has successful build `GameBasic.py`)

Key idea:

- Define a general skeleton for games: `Environment`, `Task`, `Agent`, etc.
- Provide examples that follow the same structure
- Help LLMs learn "how to build a game" in a modular, reusable way



```
        # 'DeveloperGPT' prompt from @skirano
        prompt = "You are DeveloperGPT, the most advanced AI developer tool on the planet.  You answer a

        #change prompt
        prompt_GameBasic = load_program(pjoin(args.data, "library", "GameBasic.py"))
        prompt += "Your task is to write a program that is a text-based simulation.\n"
        prompt += "The program should be written in Python. It should challenge the user by testing thei
        prompt += "```python GameBasic.py\n"
        prompt += '''
        import random

#
# Abstract class for all game objects
#
class GameObject():
    def __init__(self, name):
```

2.Can we create **an evaluation framework for code structure quality**?

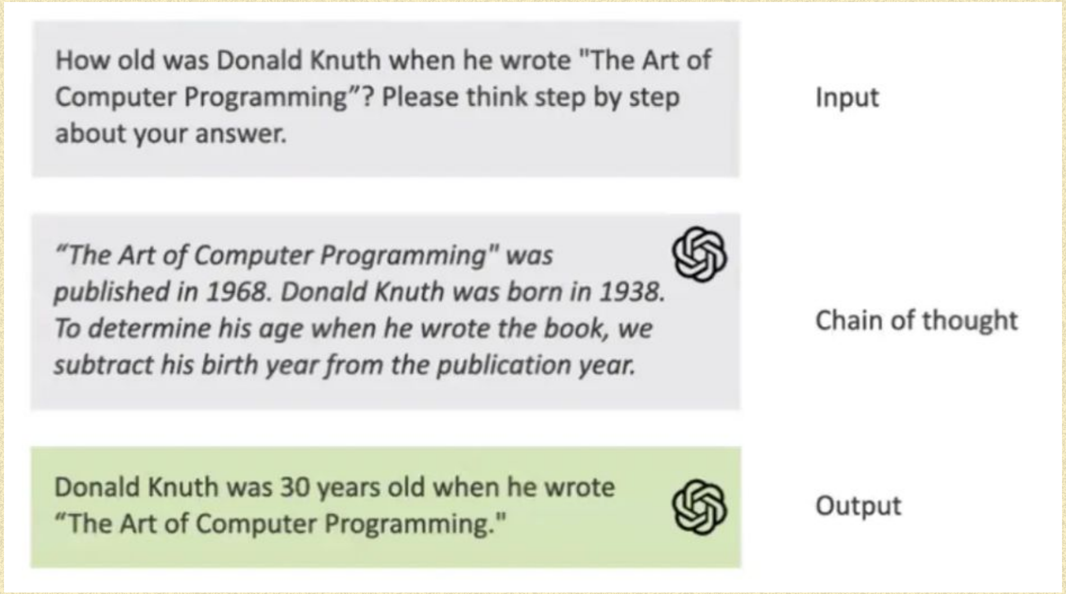
Key idea:

- Not just test if the code "runs," but assess:
  ✅ **Modularity** ✅ **Separation of concerns** ✅ **Reusability & clarity**
- Inspired by **software engineering principles** + **educational programming rubrics**

In my conclusion, these two ideas aim to **not only help LLMs generate better code**, **but also give us a way to measure what "better" means**, believe that **combining structured prompts with structural evaluation** can push the development of code-based reasoning environments with LLMs.

LLMs & Reasoning

Reasoning with LLMs



How old was Donald Knuth when he wrote "The Art of Computer Programming"? Please think step by step about your answer.

Input

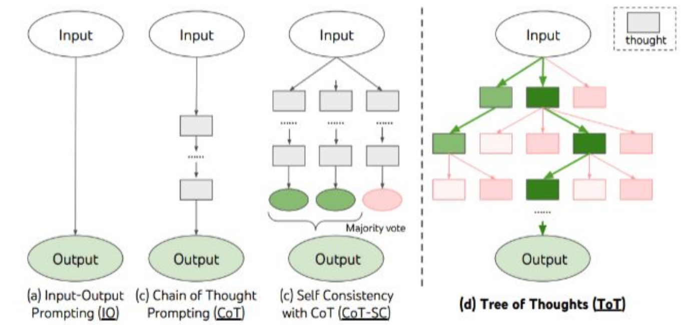
"The Art of Computer Programming" was published in 1968. Donald Knuth was born in 1938. To determine his age when he wrote the book, we subtract his birth year from the publication year.

Chain of thought

Donald Knuth was 30 years old when he wrote "The Art of Computer Programming."

Output

Large Language Model Reasoning


Input
Output
(a) Input-Output Prompting (IO)
Input
Output
(c) Chain of Thought Prompting (CoT)
Input
Majority vote
Output
(c) Self Consistency with CoT (CoT-SC)
Input
thought
Output
(d) Tree of Thoughts (ToT)

Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al. 2023)

Chain of Thought Prompting Elicits Reasoning in Large Language Models (Wei et al. 2022)

LLMs & Reasoning

For exploring the **LLMs reasoning with Text-based games: "Two Perspectives"**

Reasoning with Interactive language

Reasoning with Evaluation

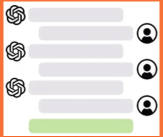
LLMs & Reasoning

Reasoning with <u>Interactive language</u>

✅ **In my thinking, I hope Reasoning with Interactive Language** means reasoning that happens during a multi-step dialogue or instruction-following process in a dynamic text based game environment.

You are in a kitchen. There is a fridge, a table with apples and bananas, and a drawer.

NPC says: "Someone is coming to take the red apple. You better <u>hide</u> it."



> Step 1: Find the red apple.

> Step 2: Think: Where would be a good place to <u>hide</u> it?

> Step 3: Action: Put the red apple inside the drawer.

<u>**How to test the LLMs for Human-like reasoning chain?**</u>

✅**Ability 1: Language understanding + motive inference :**The NPC didn't give a direct order — the model needs to understand the *implied* intent to hide something.

✅**Ability 2: Context modeling + environment evaluation**
→ The model must evaluate possible locations (fridge, table, drawer) and choose the most suitable one.

✅**Ability 3: ✅ Planning + multi-step action execution**
→ The model needs to first find the object → choose where to put it → and perform multiple actions in sequence.

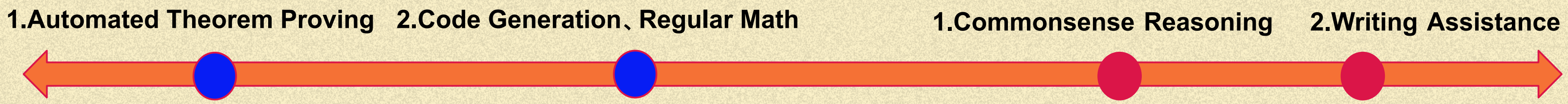
LLMs & Reasoning

Reasoning with <u>Interactive language</u>

✅ **In my future work,** reasoning with interactive language should go beyond parsing sentences.
It requires LLMs to simulate human-like reasoning — integrating motivation, dialogue history, and future planning.

🧠 1. LLMs should learn not just "what to do," but also "why to do it" in text based tasks.
🤝 2. Text-game action dialogue history and implied goals are essential to real reasoning.
🧩 3. Text games are a great testbed for this kind of cognitive emulation.

✅ **how to identify the interactive language to create reasoning text-based game?**



Formal reasoning:
the more the model needs to follow strict rules, remember logic, and avoid mistakes

Goal: human-level reasoning!!!

Informal reasoning:
the more the model needs to use experience, understand context, and guess meaning from unclear language.

LLMs & Reasoning

Reasoning with <u>Evaluation</u>

✅ **In my thinking, I hope Reasoning with Evaluation in text games** means use LLMs to generate the **text games** benchmark, and also use LLMs or RL-agent to solve it.

1. What's doing:

Use LLMs to generate text-games with natural language rules.

2.What are we trying to evaluate?

Check how well LLMs or agents can understand these games, and whether they can play them by reasoning.

3. What is my final goal?

To build a high-quality, automatically generated reasoning benchmark.

- How well LLMs or RL-trained agents can understand these natural language game descriptions?
- How effectively they can reason, act, and win within these environments?
- Whether LLMs can function as both game designers and agents?

04

Discussion

Please share your interesting idea!

Discussion & Question

I've shared how we can use LLMs to generate and explore text-based games.
Now I'd love to hear from you

Open Question 1: If you were a developer, what kind of text game would you design?
What's the setting, the goal, the challenge?
Would it be realistic, imaginative, emotional, or social?

Open Question 2: What could this text game contribute to the AI community?
Could it reveal LLMs model weaknesses? AI Safety? AI reasoning? Build human-AI?

Open Question 3:What would you try to explore through this text game?

🎮 In the future of generative AI, games are more than play —
they are tools for thought, mirrors for intelligence, and windows into what we value.