## FactorSim: Generative Simulation via Factorized Representation NeurIPS 2024

*Fan-Yun Sun*, S. I. Harini, Angela Yi, Yihan Zhou, Alex Zook, Jonathan Tremblay, *Logan Cross, Jiajun Wu, Nick Haber* 





### Motivation How can we "distill" policies from foundation models?

Large Language Model







## Input prompt:

**I**Create a bird character, visually represented as a simple rectangle within the game window. Introduce gravity, causing the bird to continuously fall slowly. Allow the bird to 'jump' or accelerate upwards in response to a player's mouse click, temporarily overcoming gravity. Periodically spawn pairs of -vertical pipes moving from right to left across the screen. Each pair should have a gap for the bird to pass through, and I their heights should vary randomly. If the bird makes contact with the ground, pipes or goes above the top of the screen the game is over. Implement the following scoring system: for each pipe it passes through it gains a positive reward of +1. Each time a terminal state is reached it receives a negative reward of -1. When the game ends, display a "Game Over!" Imessagea and stop all the motion of the game. Show the *current score in the top-left corner of the screen during* .gameplay ...



### Buggy simulations :(



### Problems when LLMs are asked to generate **complex** simulations

- Code disregards prompt specifications
- Limited context lengths
- Irrelevant context hurts performance

Levy, Mosh, Alon Jacoby, and Yoav Goldberg. "Same task, more tokens: the impact of input length on the reasoning performance of large language models." arXiv preprint arXiv:2402.14848 (2024).





### How do we generate prompt-aligned simulations?

• The distribution we want to model is  $\hat{p}(\mathcal{M}'|Q_{\text{text}})$ 

## Formulating the problem

• The distribution we want to model is

• To generate complicated simulations, we generate iteratively!

el is  $\hat{p}(\mathcal{M}'|Q_{\text{text}})$ 

ons, we generate iteratively! $p(\mathcal{M}_{k+1}|\mathcal{M}_k,q_k)$ 

## Let's see it in action

graphCode	Untitled
New Chat	1/23/2024, 2:41:29 PM
⊡ Create a bird charac 🖉 🖻	Create a bird character, visually represented as a simple rectangle, that remains static in the horizontal axis but can move up and down within the game window.
☐ create the followign game: •	: 1/23/2024, 2:41:32 PM
🖸 design a math game	> html code Cor
□ create the following scene 1	css code Copy
	O ● js code Copy → >
	: 1/23/2024, 2:41:38 PM
	Introduce gravity, causing the bird to continuously fall unless counteracted by player input.
	1/23/2024, 2:41:45 PM
	<pre>&gt; html code Cor</pre>
	🔎 css code Copy
	Ø <p< td=""></p<>
Prompt Store	
Sun A	
Star on GitHub	



## Can we do better?







### Motivation LLMs cannot generate code correctly given complex prompt logic

• The distribution we want to model is  $p(\mathcal{M}^*|q_{\text{text}})$ 

### FactorSim

### Model the generation process as expanding the state space of a POMDP.

 $\hat{p}(\mathcal{M}'|Q_{\text{text}}) \longrightarrow \hat{p}(\mathcal{M}'|Q_{\text{text}}) = \frac{1}{N}$ 

$$\frac{1}{N}\sum_{i=1}^{N} p(\mathcal{M}'|q_1^{(i)}, \dots, q_K^{(i)}), \text{ where } (q_1^{(i)}, \dots, q_K^{(i)}) \sim p(q_1, \dots, q_K|Q_1)$$



## **FactorSim** - keypoints

• A chain of thought processes that exploits the mathematical structure of POMDP using the model-view-controller design pattern.

Use LLMs to perform contextual retrieval of state variables.

### Model the generation process as expanding the state space of a POMDP.

#### Algorithm 1: FACTORSIM

Input: Q<sub>text</sub>, a natural language description of the simulation, and an LLM **Output:** a turing-computable simulation represented as a POMDP  $\mathcal{M}' = \langle S, A, O, T, \Omega, R \rangle$ 

Initialize a Factored POMDP  $\mathcal{M}_1 \leftarrow \langle S_1, A, \emptyset, T_1, \emptyset, R_1 \rangle$  where  $-S_1 := \{s_{\text{score}}\}$ - A is the set of all keyboard inputs -  $T_1$  is an identity function, i.e.,  $T_1(s' \mid s, a) = \mathbf{1}[s' = s]$ -  $R_1(s, a, s') := s'_{\text{score}} - s_{\text{score}}$ 

// Chain of Thought Derive a step-by-step plan  $(q_1, \ldots, q_k) \sim p(q_1, \ldots, q_k \mid Q_{\text{text}})$ Eq. (1)

for each step, or module  $q_k$  do State space update & context selection  $p(S_{k+1}, S[Z_k] | S_k, q_k)$ 

// Controller component update

Action-dependent state transition model update:  $p(T_{k+1}^{(a)}|S[Z_k], A, q_k)$ 

// Model component update

Action-independent state transition model update:  $p(T_{k+1}^{(s)}|T[Z_k], S[Z_k], q_k)$ // View component update

**Observation model update**:  $p(\Omega_{k+1}|S[Z_k], q_k)$ 

 $\mathcal{M}_{k+1} = \langle S_{k+1}, A, O_{k+1}, T_{k+1}, \Omega_{k+1}, R_1 \rangle$  where  $O_{k+1}$  is the new observation space defined by  $S_{k+1}$  and  $\Omega_{k+1}$ , and  $T_{k+1}(s' \mid s, a) = T_{k+1}^{(s)}(s' \mid s) \cdot T_{k+1}^{(a)}(s \mid s, a)$ .

end

Return the final simulation  $\mathcal{M}' \leftarrow \mathcal{M}_{k+1}$ 

Eq. (9),(10)

Eq. (13)

### **Algorithm 1:** FACTORSIM

**Input:** Q<sub>text</sub>, a natural language description of the simulation, and an LLM **Output:** a turing-computable simulation represented as a POMDP  $\mathcal{M}' = \langle S, A, O, T, \Omega, R \rangle$ 

Initialize a Factored POMDP  $\mathcal{M}_1 \leftarrow \langle S_1, A, \emptyset, T_1, \emptyset, R_1 \rangle$  where  $-S_1 := \{s_{score}\}$ - A is the set of all keyboard inputs -  $T_1$  is an identity function, i.e.,  $T_1(s' \mid s, a) = \mathbf{1}[s' = s]$ -  $R_1(s, a, s') := s'_{\text{score}} - s_{\text{score}}$ 

// Chain of Thought Derive a step-by-step plan  $(q_1, \ldots, q_k) \sim p(q_1, \ldots, q_k \mid Q_{\text{text}})$ 

for each step, or module  $q_k$  do State space update & context selection  $p(S_{k+1}, S[Z_k]|S_k, q_k)$ Eq. (9),(11)

// Controller component update

Action-dependent state transition model update:  $p(T_{k+1}^{(a)}|S[Z_k], A, q_k)$ // Model component update

Action-independent state transition model update:  $p(T_{k+1}^{(s)}|T[Z_k], S[Z_k], q_k)$ // View component update

**Observation model update**:  $p(\Omega_{k+1}|S[Z_k], q_k)$ 

 $\mathcal{M}_{k+1} = \langle S_{k+1}, A, O_{k+1}, T_{k+1}, \Omega_{k+1}, R_1 \rangle$  where  $O_{k+1}$  is the new observation space defined by  $S_{k+1}$  and  $\Omega_{k+1}$ , and  $T_{k+1}(s' \mid s, a) = T_{k+1}^{(s)}(s' \mid s) \cdot T_{k+1}^{(a)}(s \mid s, a)$ .

### end

.

Return the final simulation  $\mathcal{M}' \leftarrow \mathcal{M}_{k+1}$ 

Eq. 🚺)

Eq. (13)

## **Experiments - Two Domains**

### 2D Reinforcement Learning Game Generation

### Robotics Task Generation

## **Experiment 1: Game Generation** - Pygame Learning Environment



# - Pygame Learning Environment













## 1. Better code generation accuracy



Token Usage

## 2. Zero-shot transfer results



## **Robotics Task Generation**

Can you generate the following task FactorSim -"Stack Boxes Column"?



#### **Generated** Task

lass StackBoxesColumn(Task)

"""Sequentially stack a set number of boxes into a vertical column on the tabletop without them toppling

#### State Variables

super().\_\_init\_\_() # initialize the state\_manager attributes that manage all the state variables in this task StateManager = type("StateManager", (object,), {})

self.state\_manager = StateManager()

self.state\_manager.main\_target\_pose = ((0.5, 0.0, 0.0), (0, 0, 0, 1,))

self.max\_steps = 15 self.lang\_template = """Sequentially stack a set number of boxes into a vertical column on the tabletop without them toppling over."""

self.state\_manager.first\_box\_size = (0.17, 0.17, 0.17)

self.state\_manager.first\_box\_color = "blue"

self.state\_manager.first\_box\_urdf = "box/box-template.urdf"

self.state\_manager.second\_box\_size = (0.07, 0.07, 0.07)

self.state manager.second box color = "blue

self.state\_manager.second\_box\_urdf = "box/box-template.urdf"

self.state\_manager.third\_box\_size = (0.07, 0.07, 0.07)

self.state\_manager.third\_box\_color = "blue"

self.state\_manager.third\_box\_urdf = "box/box-template.urdf" self.additional reset()

matches=np.ones((1, 1)),

# Place the first box on the tabletop to serve as the base A subtask function

# Step 1. Retrieve the box size, color, and URDF path from the state manager box\_size = self.state\_manager.first\_box\_size box\_color = self.state\_manager.first\_box\_color

box\_urdf = self.state\_manager.first\_box\_urdf

# Step 2. Get a random position for the initial placement of the box initial\_box\_pose = self.get\_random\_pose(env, box\_size)

# Step 3. The target pose for the first box is on the tabletop. # We'll use the main\_target\_pose attribute from state manager to position the box target\_box\_pose = self.state\_manager.main\_target\_pose

matches=np.ones((1, 1)),

# Step 4. Add the first box to the environment at the initial pose box\_id = env.add\_object(

box\_urdf, initial\_box\_pose, color=utils.COLORS[box\_color]

# Step 5: Define the goal for this subtask. The goal is to position the first box at the target pose self.add\_goal( objs=[box\_id],

Wang, Lirui, et al. "Gensim: Generating robotic simulation tasks via large language models." *arXiv preprint arXiv:2310.01361* (2023).

## **Robotic Tasks Results**



### Tasks that FactorSim successfully generated but all other baselines failed.

Build Lamp Post



Insert Ell Along Squre Path



Build Picnic Table



Color Coordinated Cylinder Tower



## Some extensions

input image



generated task:



Pick up the various food items and place them into respective zones based on their category: cans, boxes, and bottles





generated task:



#### CategorySorting

Pick up the different food items (cookies, macaroni and cheese, mustard, raisins, and cherries) and sort them into separate zones marked on the tabletop base on their type.

## Thank you!





#### **Generated** simulations P