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

FactorSim: Generative Simulation via Factorized Representation NeurIPS 2024

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

Large Language Model

Input prompt:

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 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 …*

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!

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

 $p(\mathcal{M}_{k+1}|\mathcal{M}_k,q_k)$

Let's see it in action

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

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

FactorSim *— keypoints*

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

• 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.

Algorithm 1: FACTORSIM

Input: Q_{text} , 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., $T_1(s' | s, a) = \mathbf{1}[s' = s]$ $R_1(s,a,s') := s'_{\text{score}} - s_{\text{score}}$

// Chain of Thought Derive a step-by-step plan $(q_1, \ldots, q_k) \sim p(q_1, \ldots, q_k | Q_{\text{text}})$ 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: $p(T_{k+1}^{(a)}|S\left[\mathcal{Z}_{k}\right],A,q_{k})$

// 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 Ω_{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_{text} , 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., $T_1(s' | s, a) = \mathbf{1}[s' = s]$

 $- R_1(s, a, s') := s'_{score} - s_{score}$

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

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. $(9), (10)$

// 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 Ω_{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. $\left(\mathbf{I} \right)$

Eq. (13)

Experiments - Two Domains

• 2D Reinforcement Learning Game Generation

• Robotics Task Generation

Experiment 1: Game Generation - Pygame Learning Environment

Experiment 1: Game Generation - Pygame Learning Environment

1. Better code generation accuracy

Token Usage

2. Zero-shot transfer results

Robotics Task Generation

Can you generate the following task "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

- # 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()

def bosition first boxiself, env.

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

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

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

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:

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!

Score: 15 Game Over! **Press R to Restart** $1 - 11$ **Policy Training** reward on unseen testing environment over training ~ 2 200

PO Generated simulations