FactorSim: Generative Simulation via Factorized Representation

NeurIPS 2024

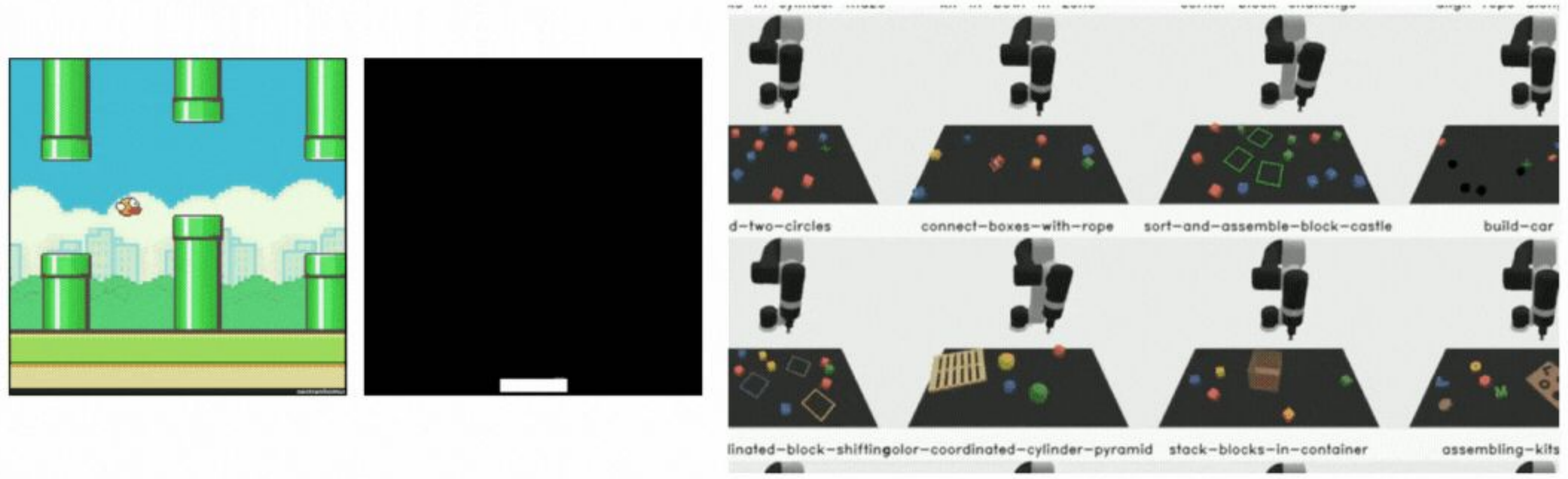
Fan-Yun Sun, S. I. Harini, Angela Yi, Yihan Zhou, Alex Zook, Jonathan Tremblay,
Logan Cross, Jiajun Wu, Nick Haber

Motivation

How can we “distill” policies from foundation models?

Large Language Model

generate simulations

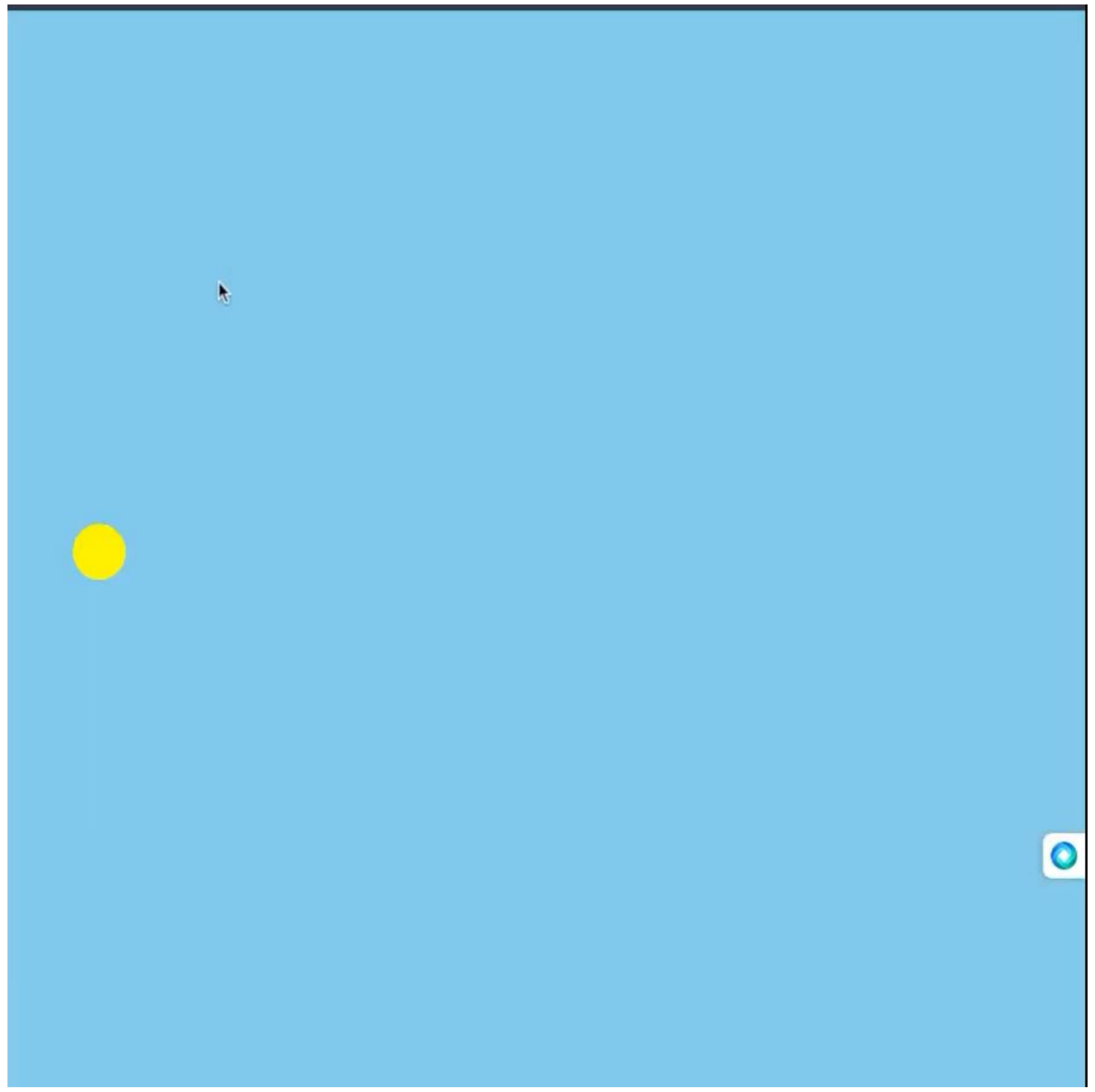


Expert Policies

Input prompt:

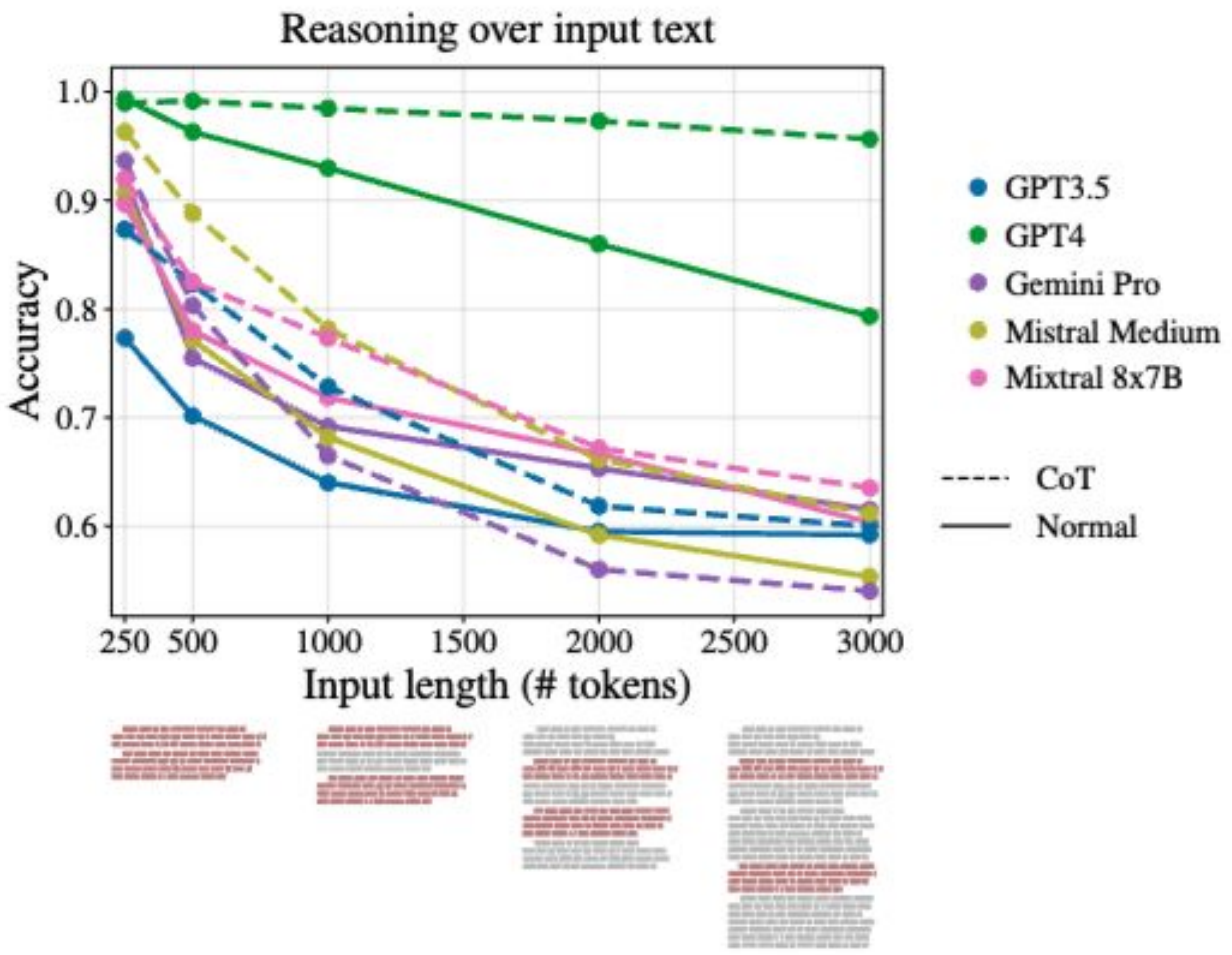
Create a bird character, visually represented as a simple rectangle within the game window. Introduce gravity, causing the bird to continuously fall slowly. Allow the bird to 'jump' or accelerate upwards in response to a player's mouse click, temporarily overcoming gravity. Periodically spawn pairs of 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!" messagea 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 $\hat{p}(\mathcal{M}'|Q_{\text{text}})$

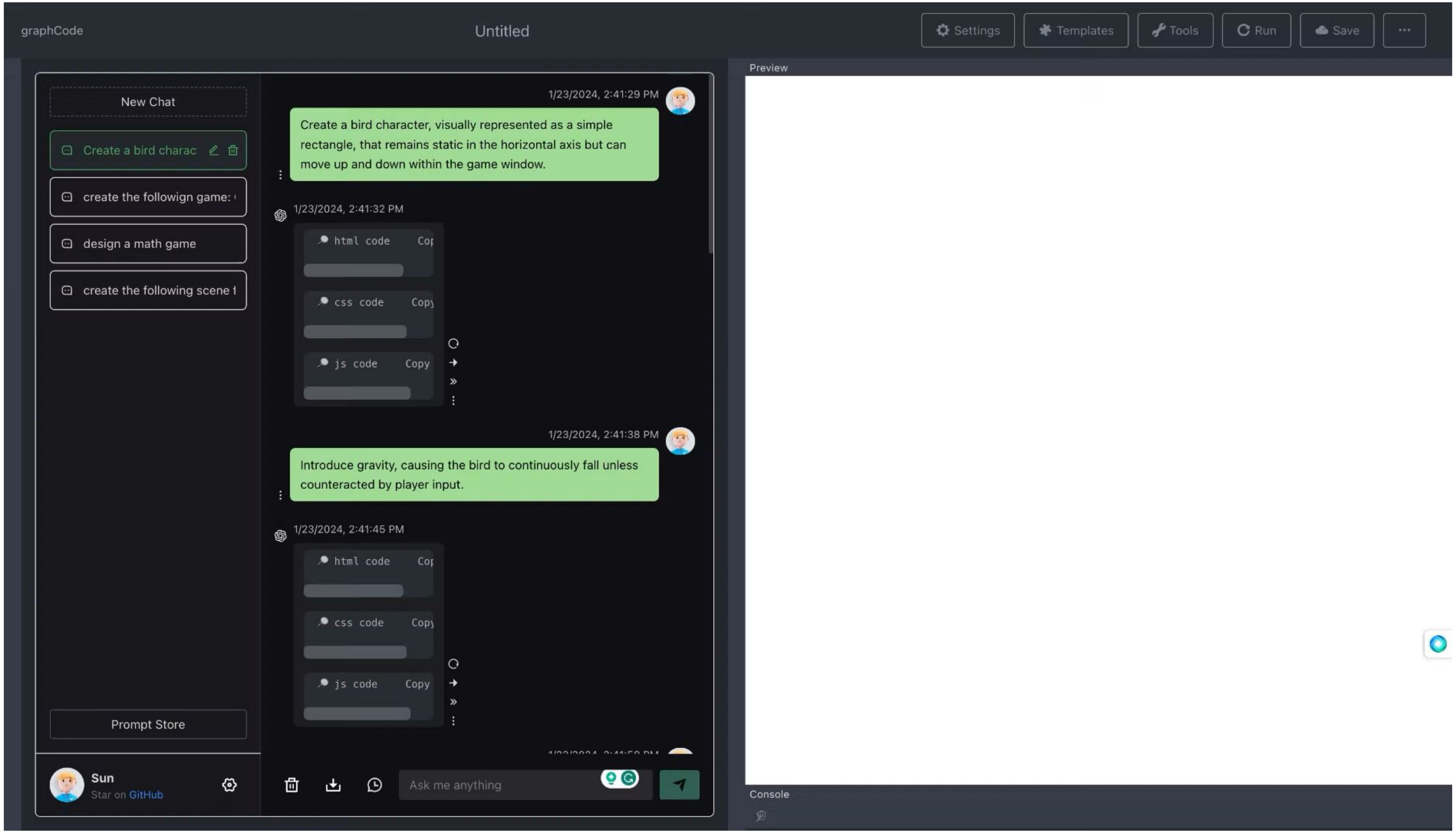
Formulating the problem

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

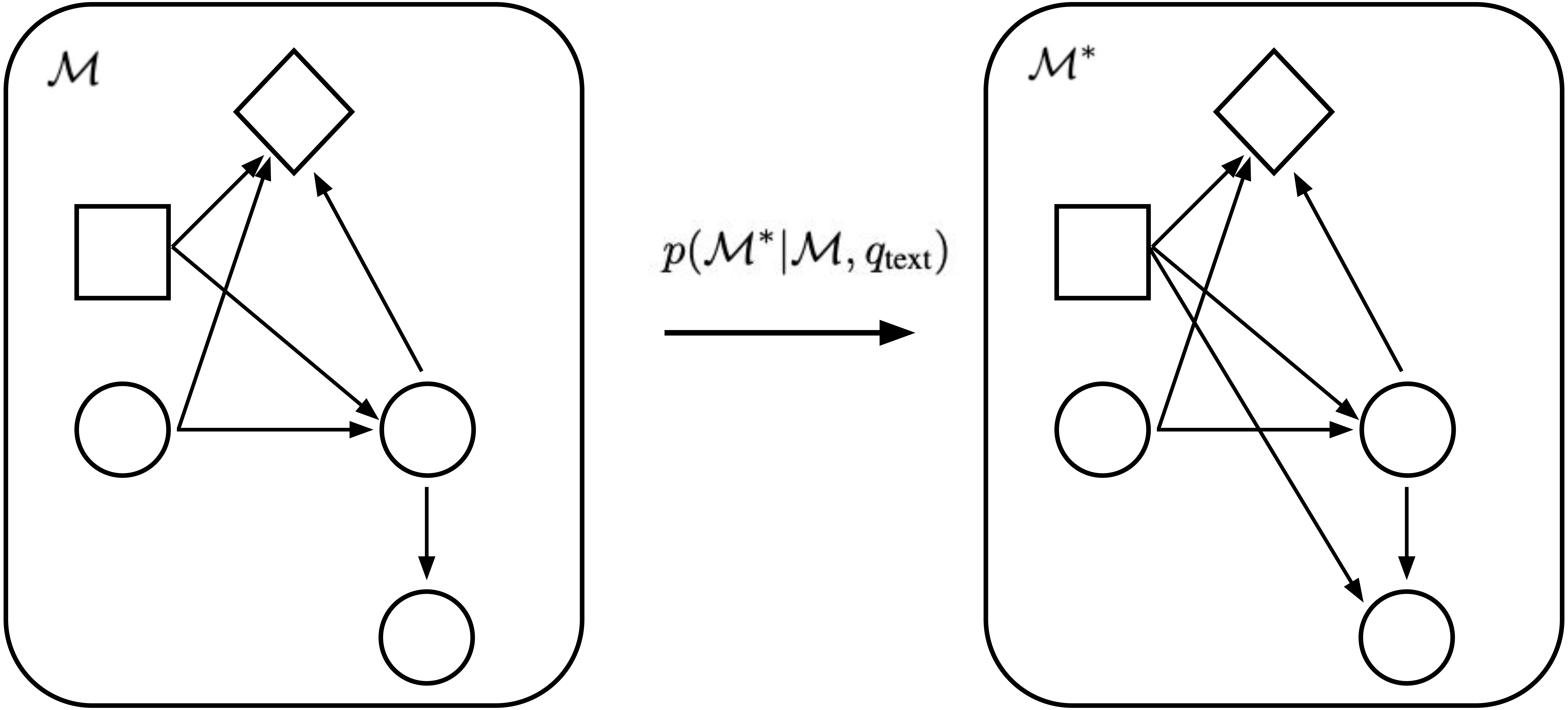
- To generate complicated simulations, we generate iteratively!

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

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.

$$\hat{p}(\mathcal{M}'|Q_{\text{text}}) \longrightarrow \hat{p}(\mathcal{M}'|Q_{\text{text}}) = \frac{1}{N}\sum_{i=1}^{N} p(\mathcal{M}'|q_1^{(i)}, \ldots, q_K^{(i)}), \quad \text{where } (q_1^{(i)}, \ldots, q_K^{(i)}) \sim p(q_1, \ldots, q_K|Q_{text}),$$

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

// 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 $p(S_{k+1}, S[Z_k] | S_k, q_k)$ 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: $p(\Omega_{k+1} | S[Z_k], q_k)$ Eq. (13)

$\mathcal{M}_{k+1} = \langle S_{k+1}, A, O_{k+1}, T_{k+1}, \Omega_{k+1}, R_1 \rangle$ 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 $\mathcal{M}' \leftarrow \mathcal{M}_{k+1}$

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

// 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 $p(S_{k+1}, S\,[Z_k]\,|S_k, q_k)$ Eq. (9),(10)

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

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

// View component update
Observation model update: $p(\Omega_{k+1}|S\,[Z_k]\,, q_k)$ Eq. (13)

$\mathcal{M}_{k+1} = \langle S_{k+1}, A, O_{k+1}, T_{k+1}, \Omega_{k+1}, R_1 \rangle$ where O_{k+1} is the new observation space defined by S_{k+1} and Ω_{k+1}, and $T_{k+1}(s' \mid s, a) = T^{(s)}_{k+1}(s' \mid s) \cdot T^{(a)}_{k+1}(s \mid s, a)$.

end

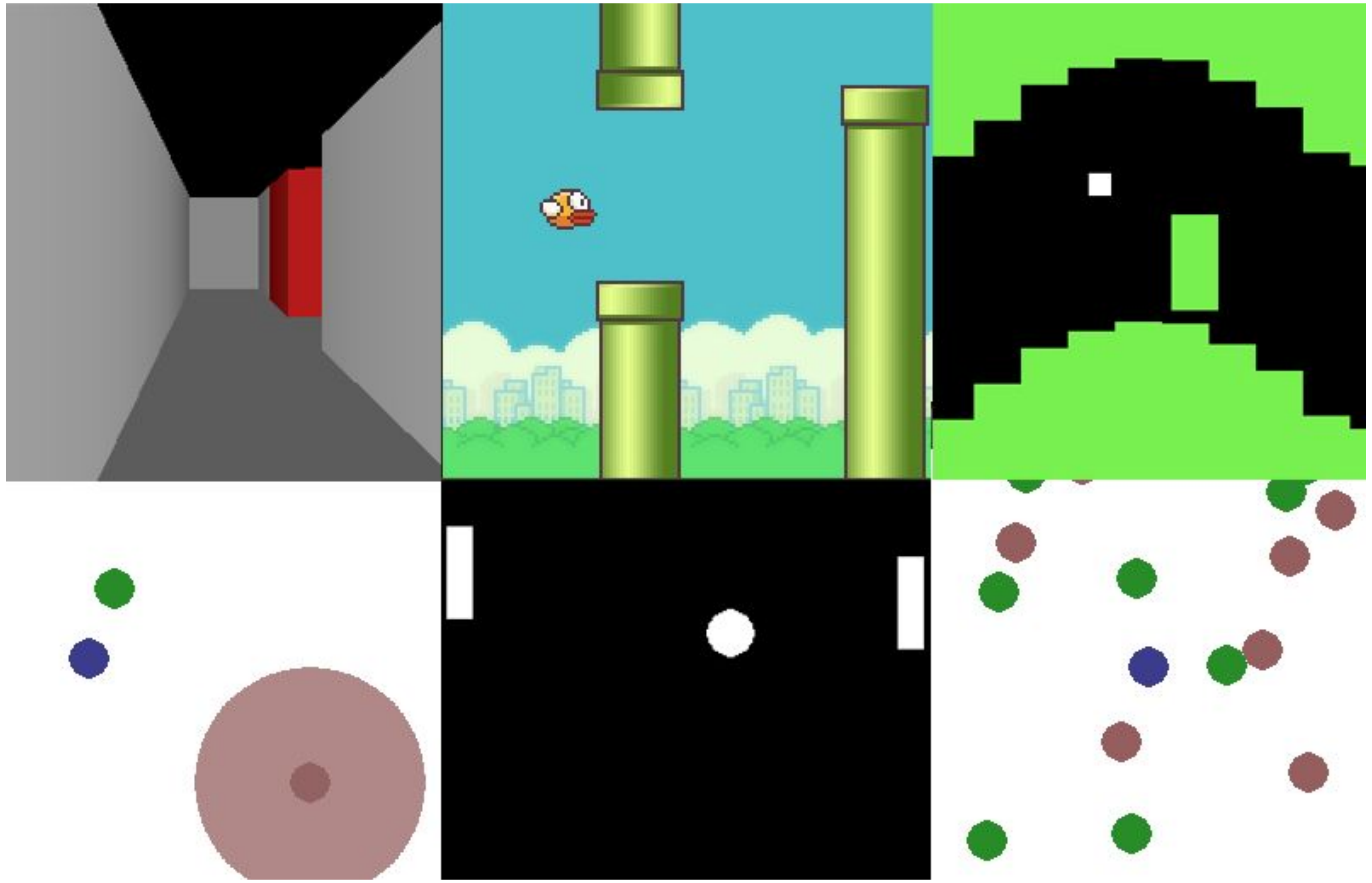
Return the final simulation $\mathcal{M}' \leftarrow \mathcal{M}_{k+1}$

Experiments - Two Domains

- **2D Reinforcement Learning Game Generation**
- **Robotics Task Generation**

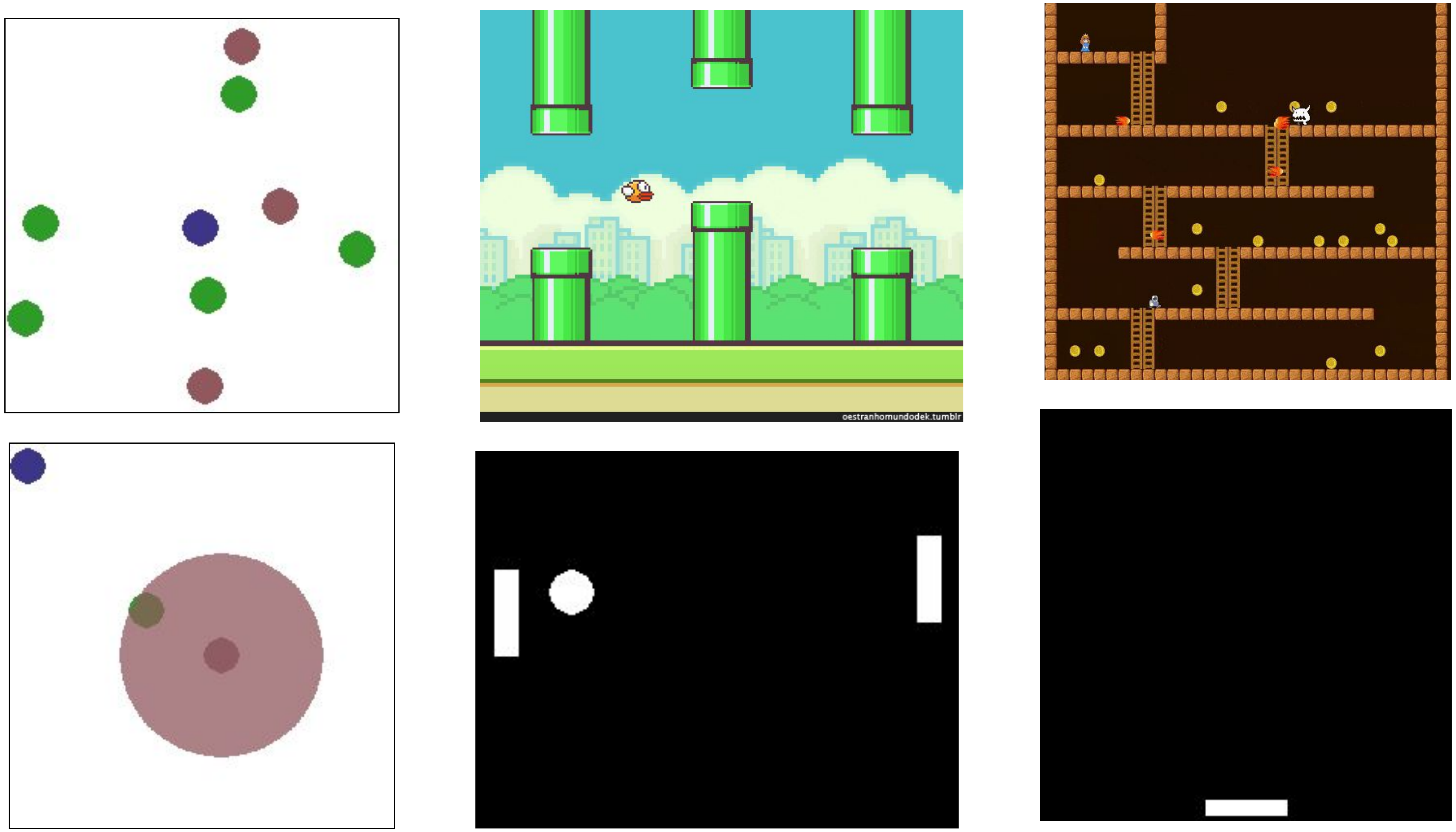
Experiment 1: Game Generation

- Pygame Learning Environment

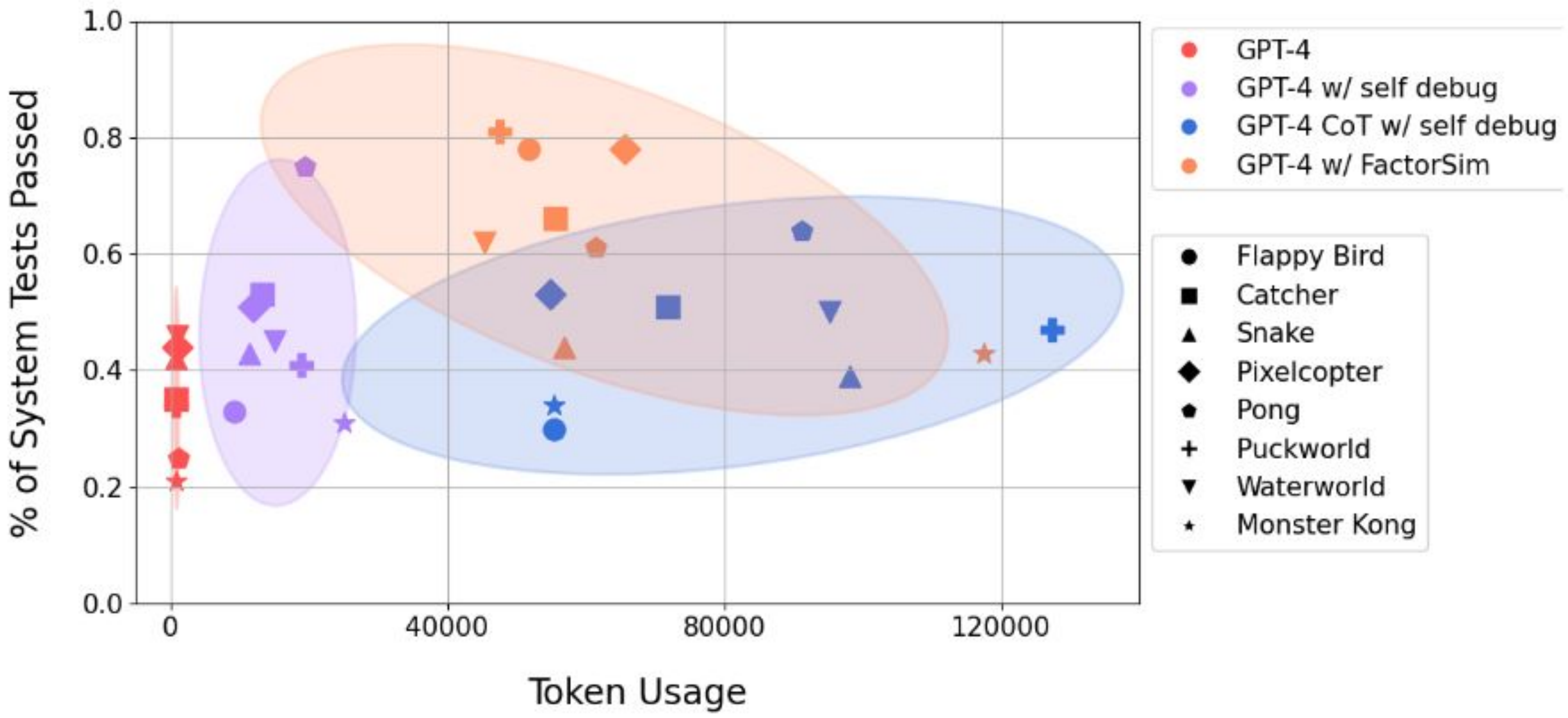


Experiment 1: Game Generation

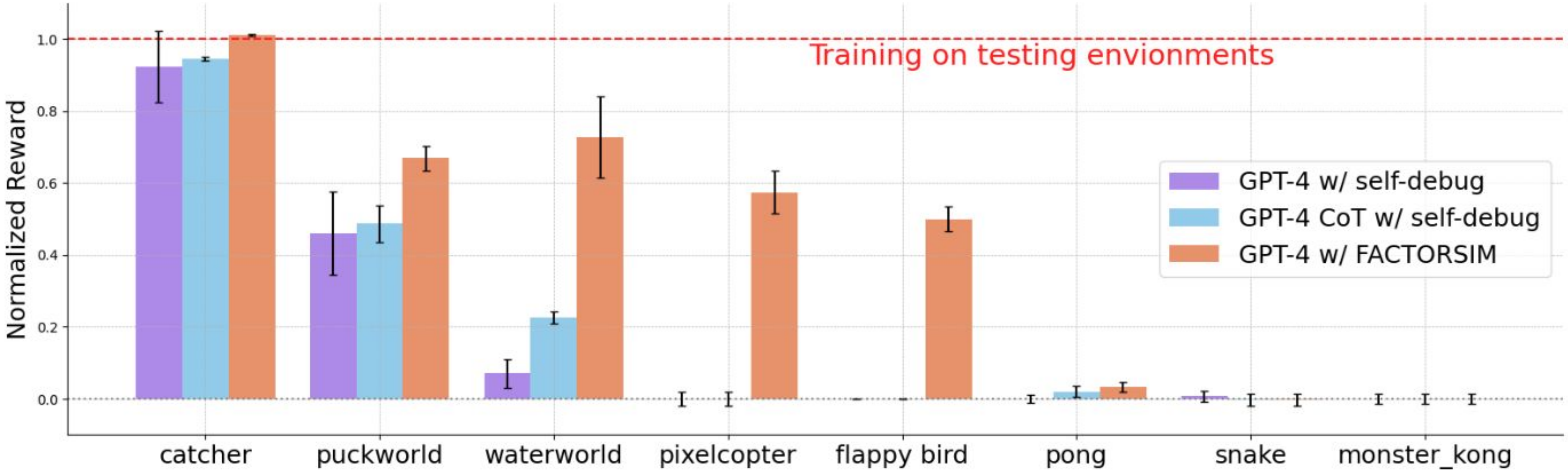
- Pygame Learning Environment



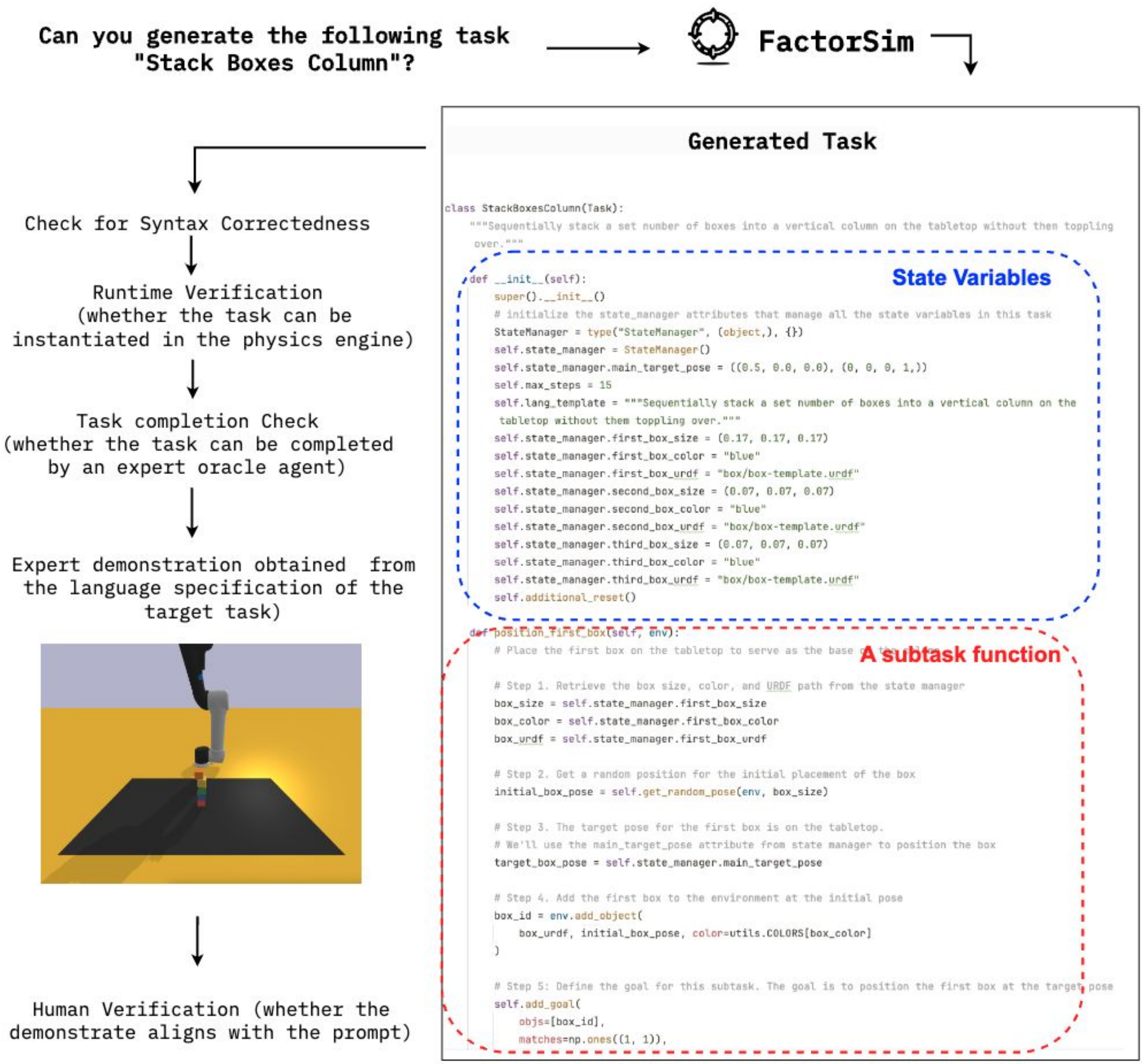
1. Better code generation accuracy



2. Zero-shot transfer results

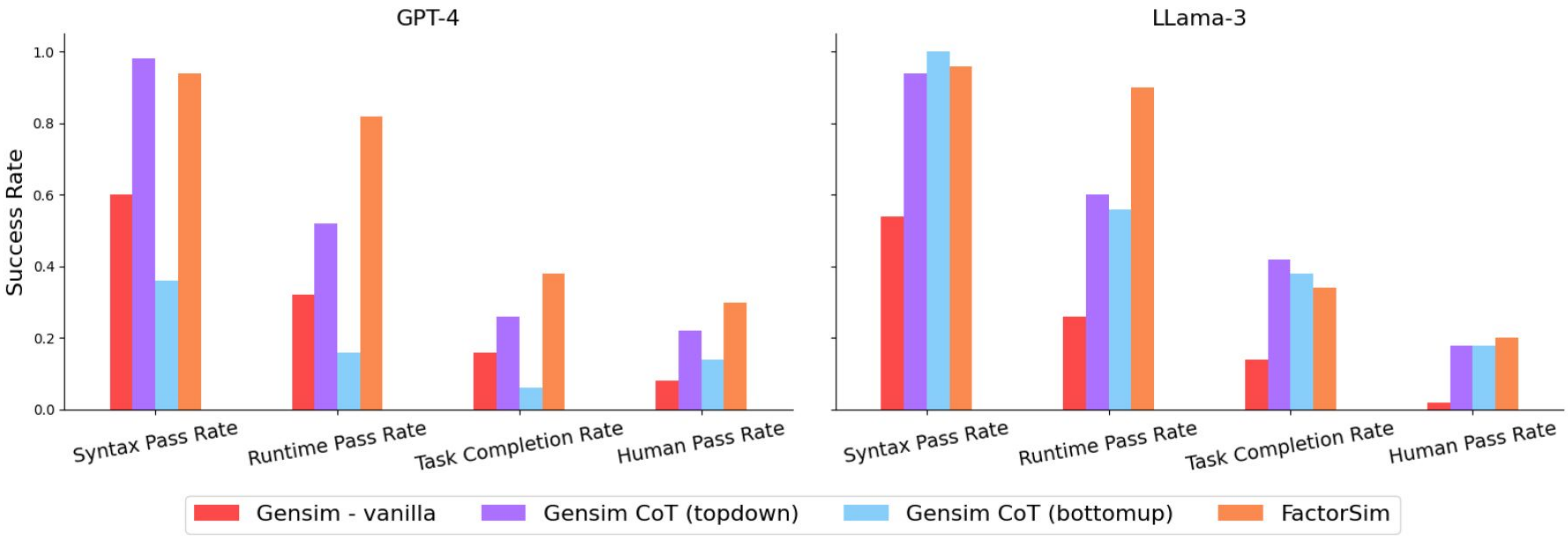


Robotics Task Generation



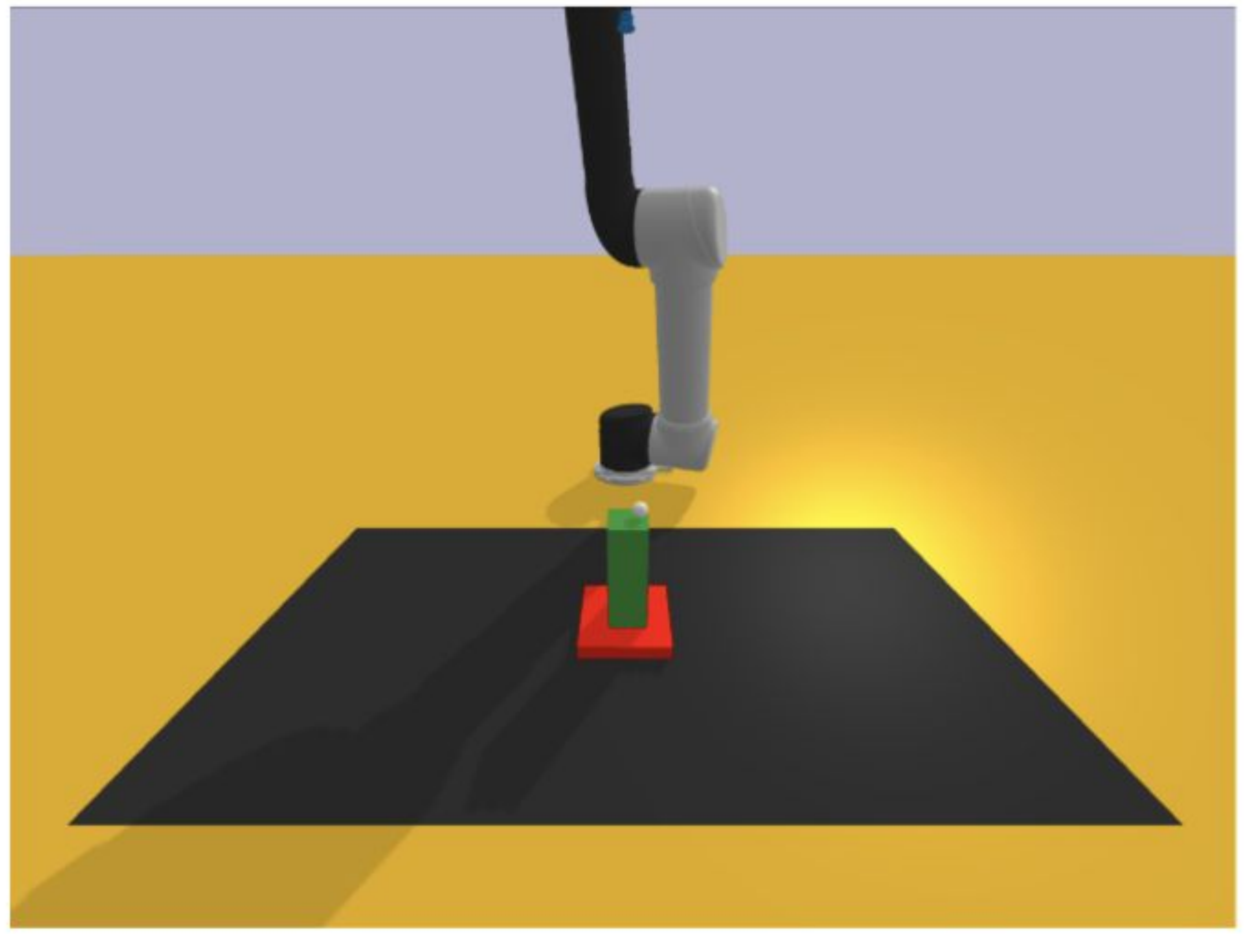
Wang, Lirui, et al. "Gensim: Generating robotic simulation tasks via large language models." *arXiv preprint arXiv:2310.01361* (2023).

Robotic Tasks Results

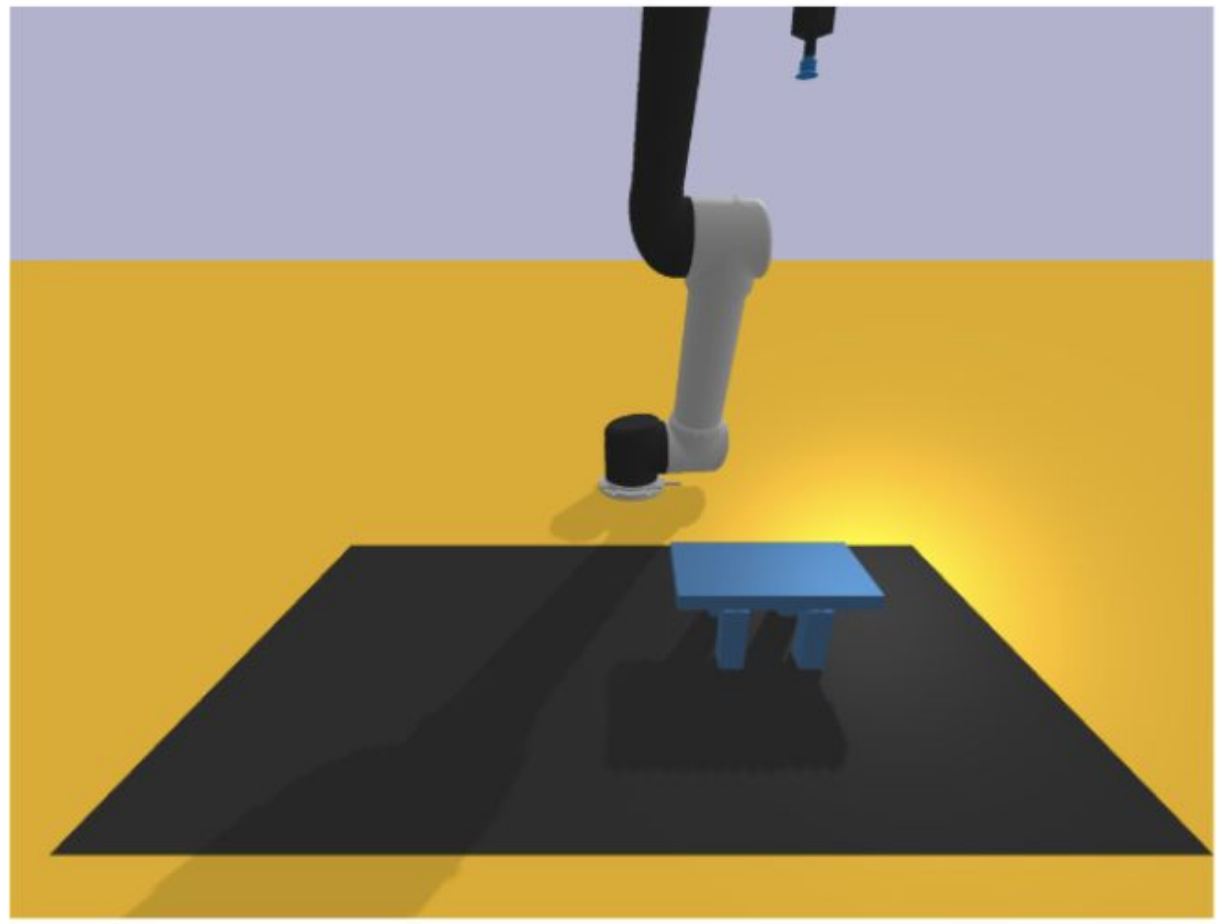


Tasks that FactorSim successfully generated but all other baselines failed.

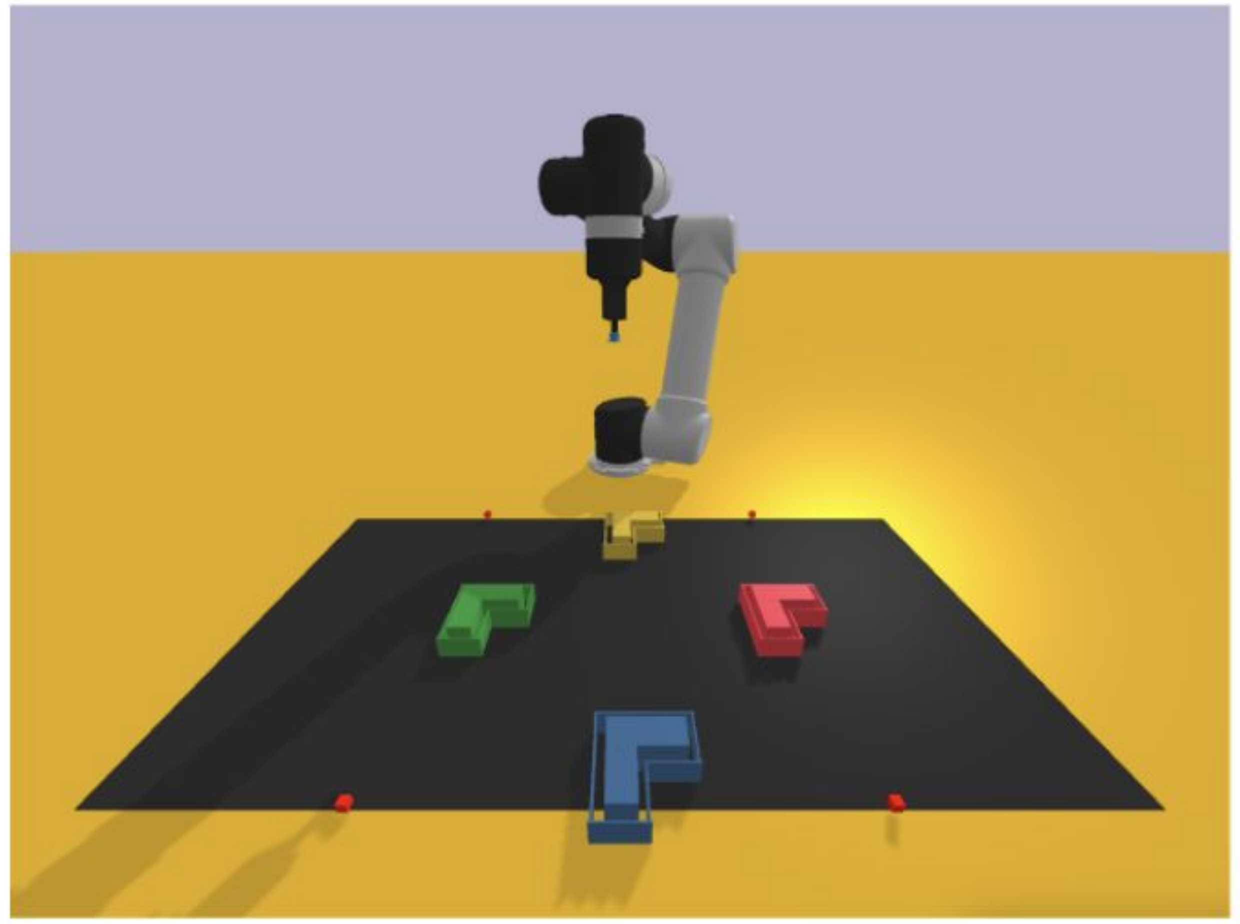
Build Lamp Post



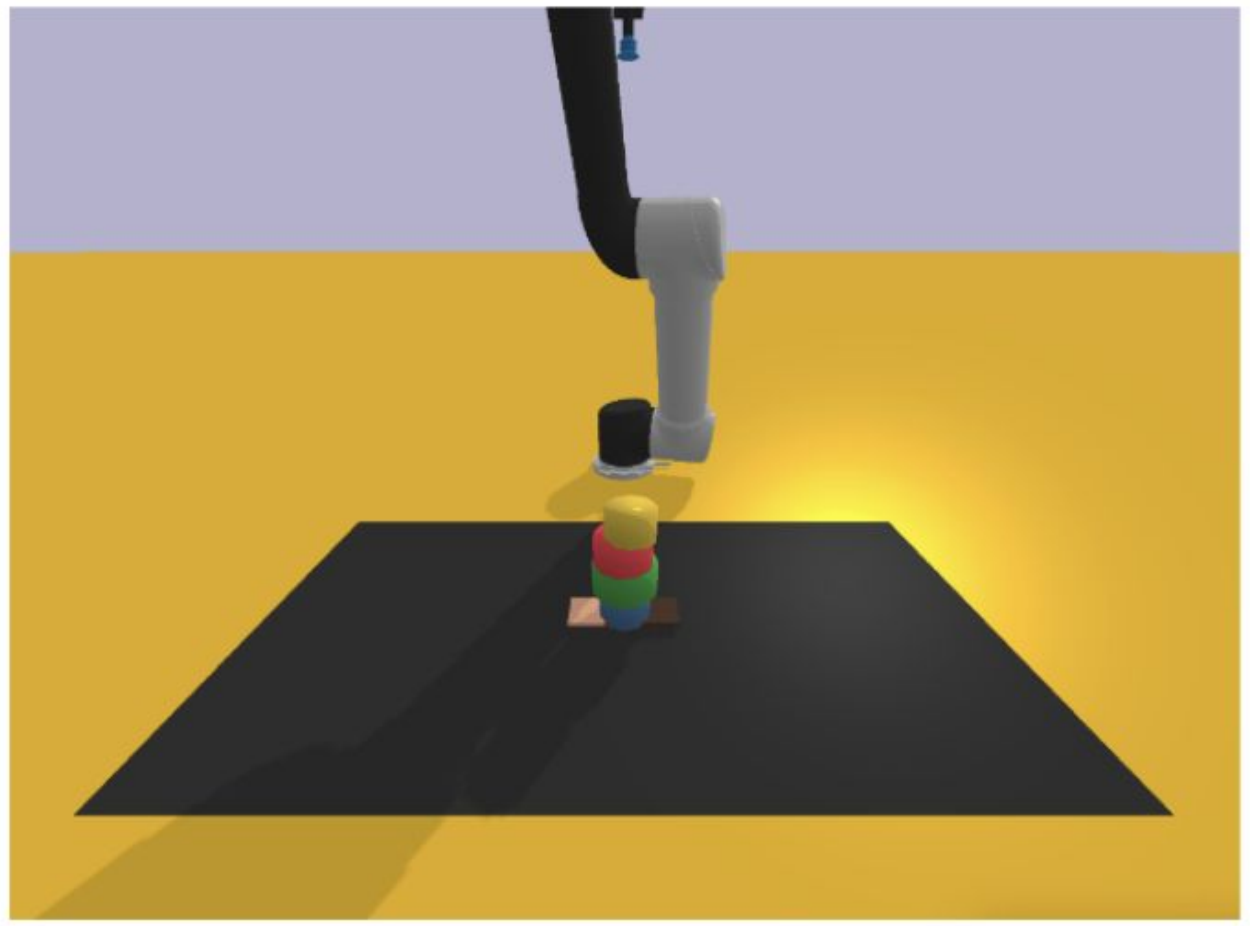
Build Picnic Table



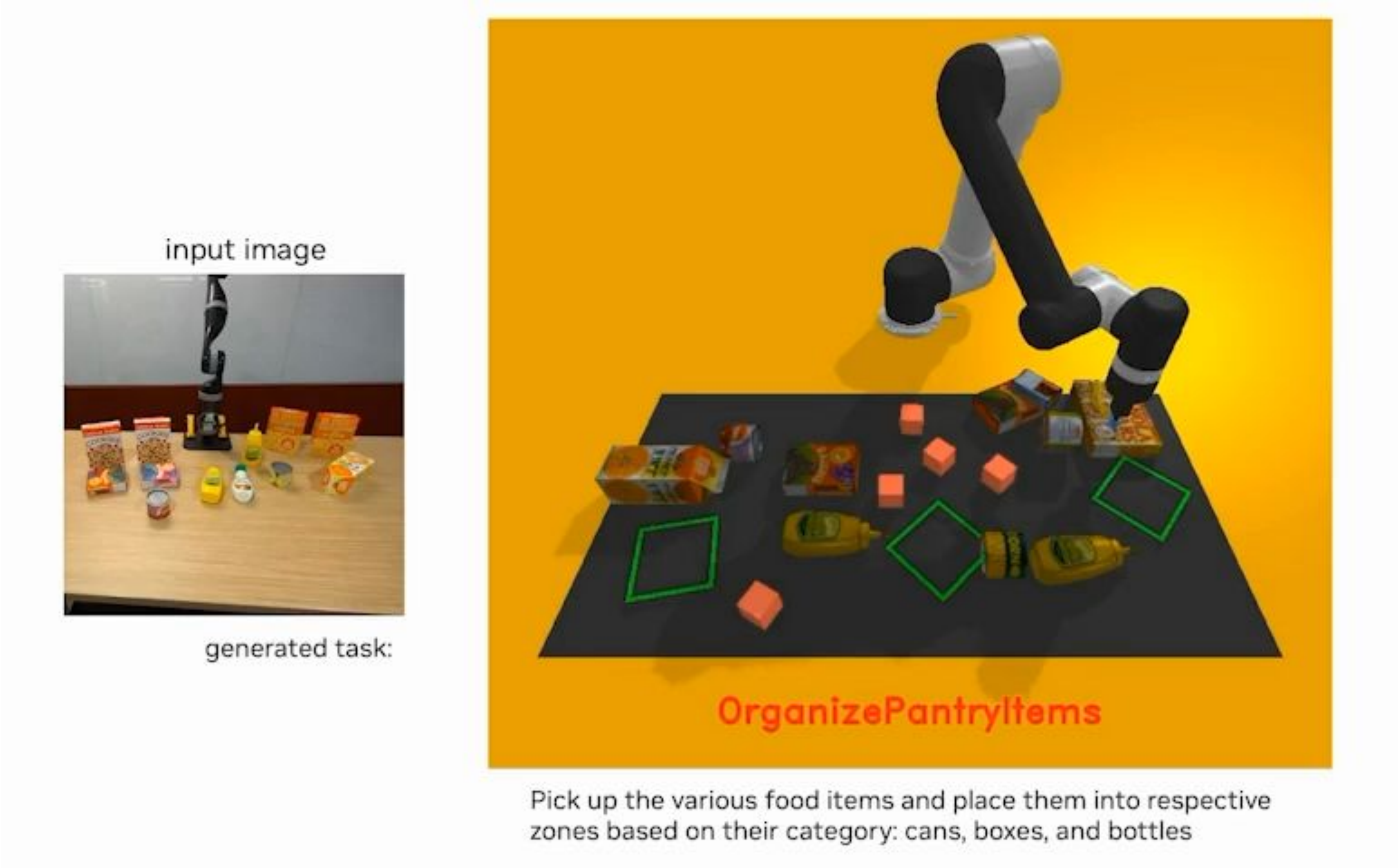
Insert Ell Along Squre Path

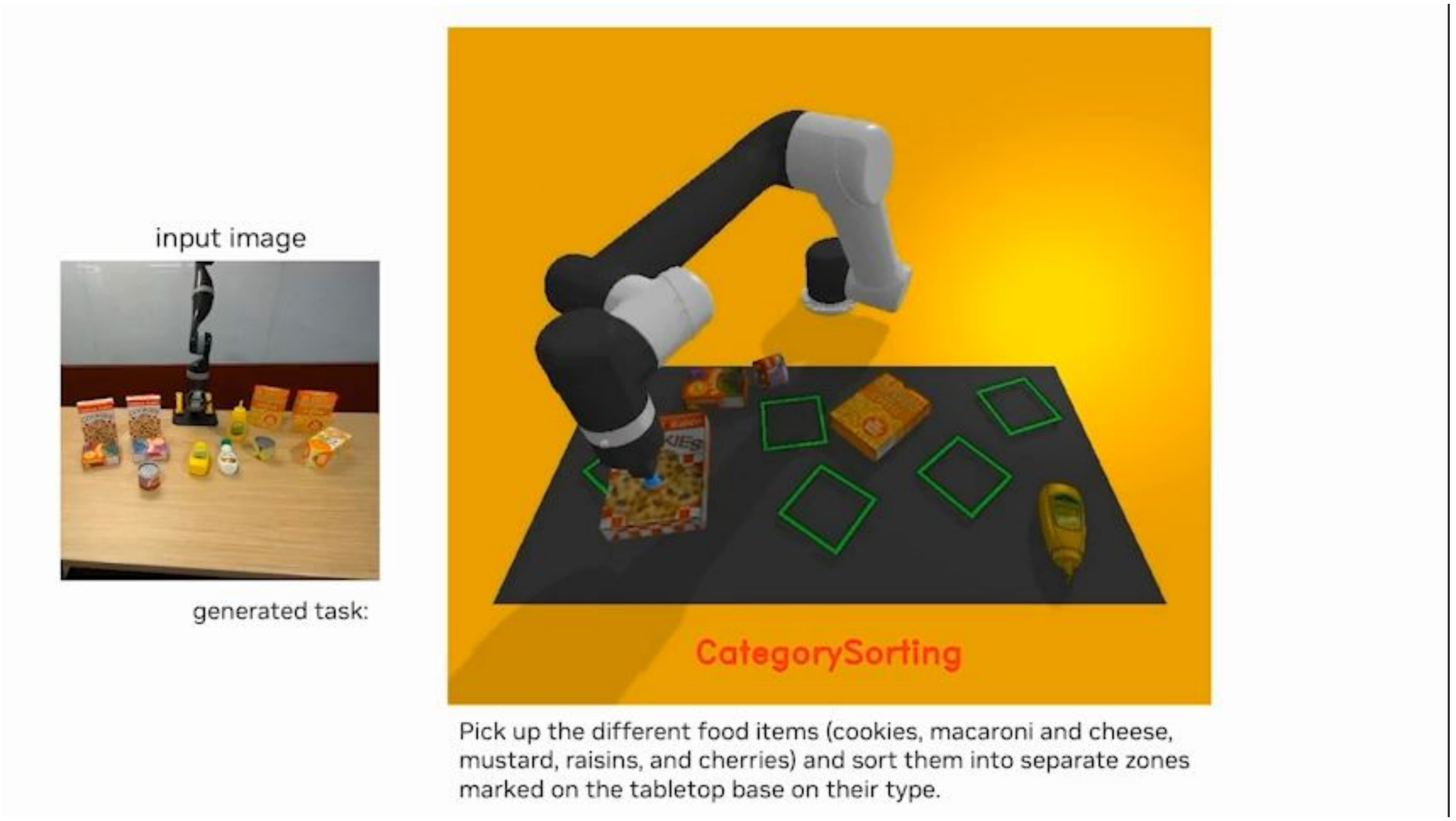


Color Coordinated Cylinder Tower

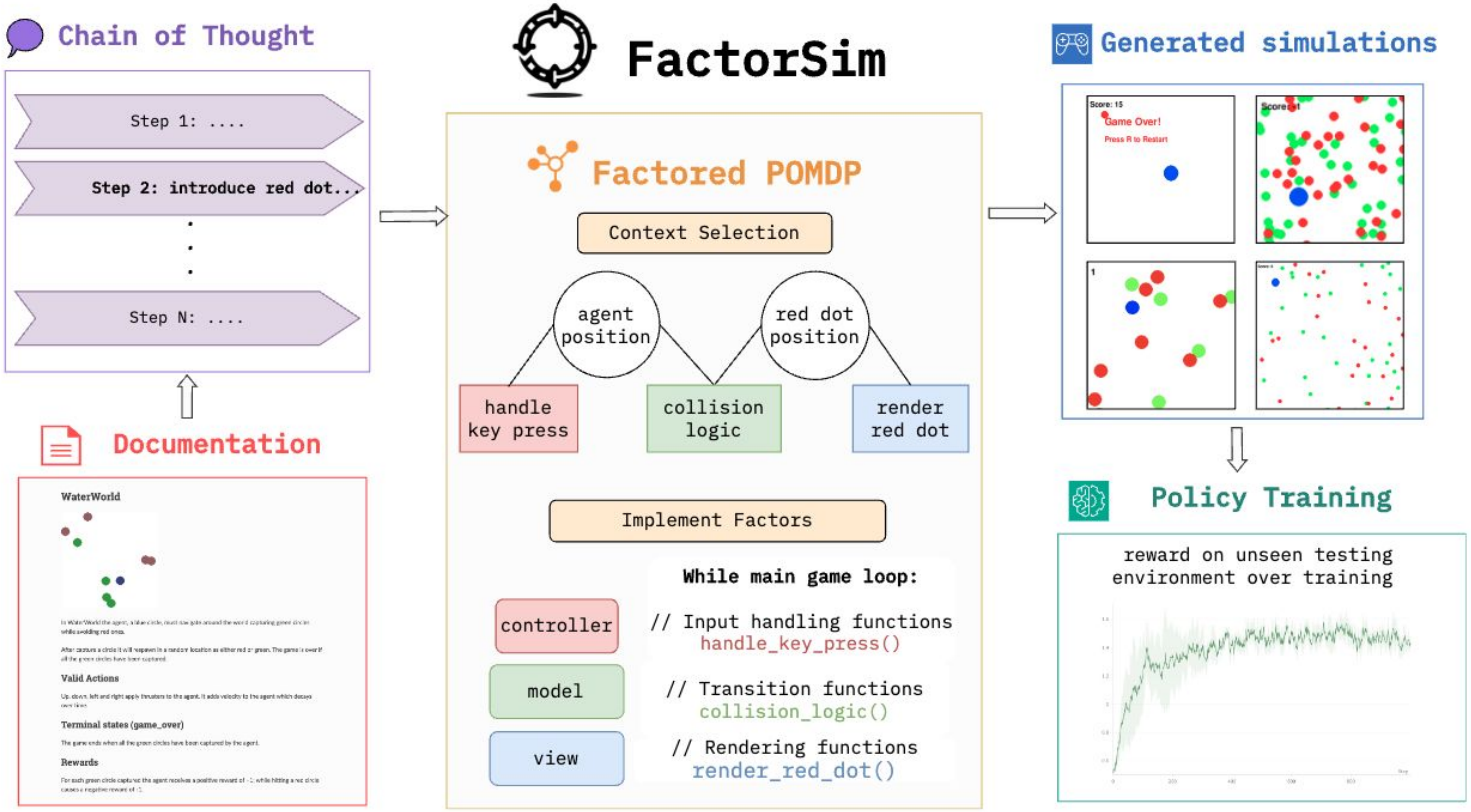


Some extensions





Thank you!



Chain of Thought
Step 1: ....
Step 2: introduce red dot...
Step N: ....
Documentation
WaterWorld
Valid Actions
Terminal states (game_over)
Rewards
FactorSim
Factored POMDP
Context Selection
agent position
red dot position
handle key press
collision logic
render red dot
Implement Factors
While main game loop:
controller
// Input handling functions
handle_key_press()
model
// Transition functions
collision_logic()
view
// Rendering functions
render_red_dot()
Generated simulations
Game Over!
Press R to Restart
Policy Training
reward on unseen testing environment over training