Lecture 12:

Creating AI Agents + Simulating Virtual Worlds for Training

Visual Computing Systems
Stanford CS348K, Spring 2024

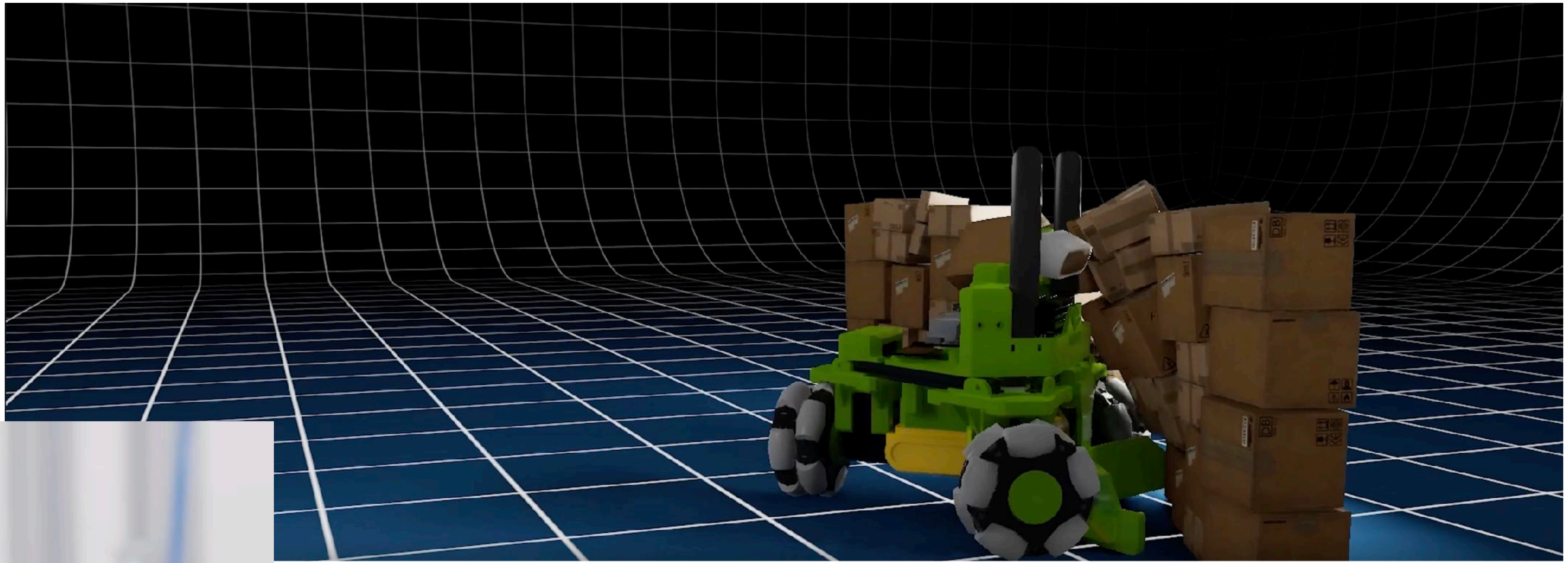
First:
resuming discussion of LLM-based agents from last time
(Generative agents + Voyager papers)

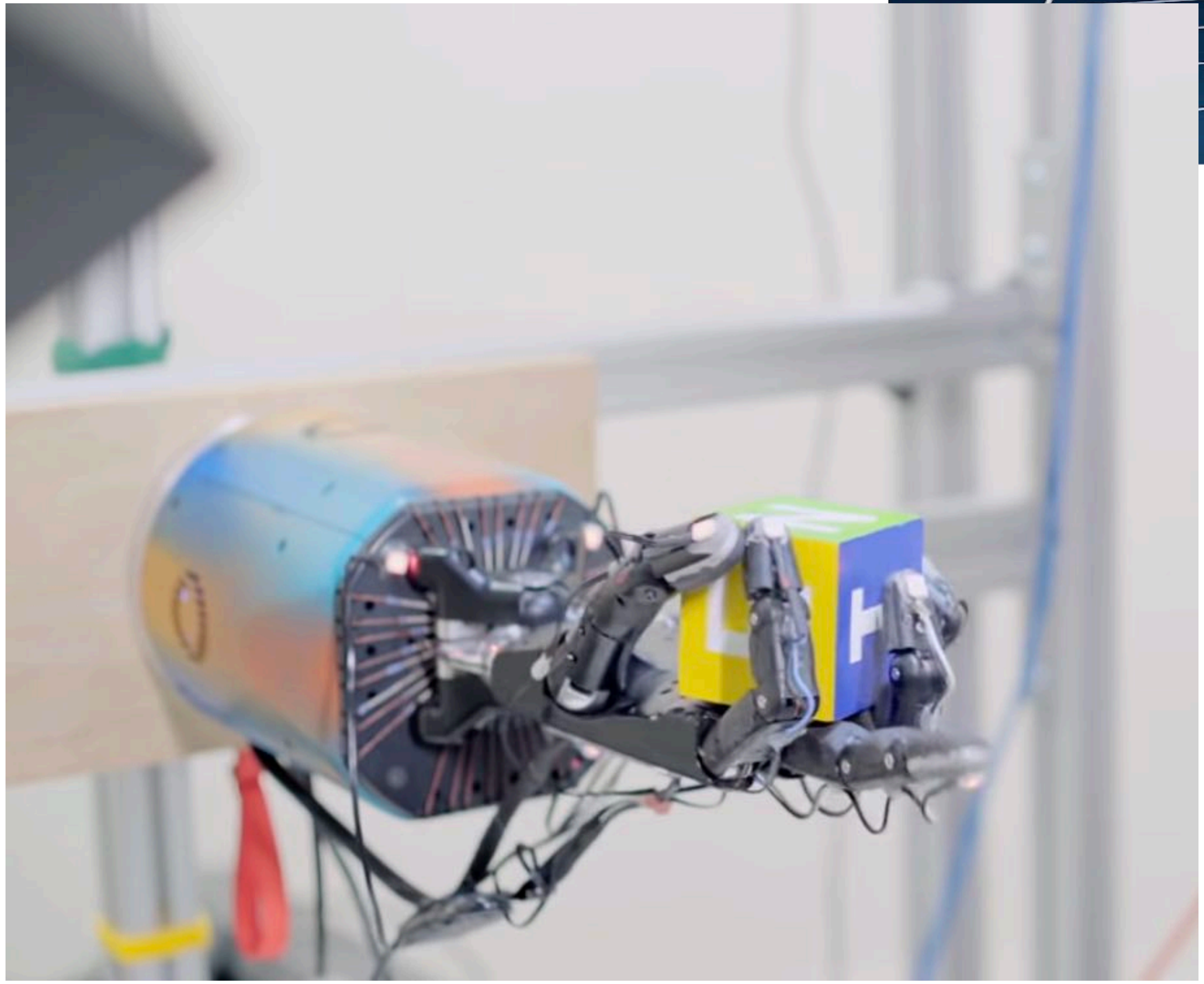
LLM-driven problem solving vs. trial and error

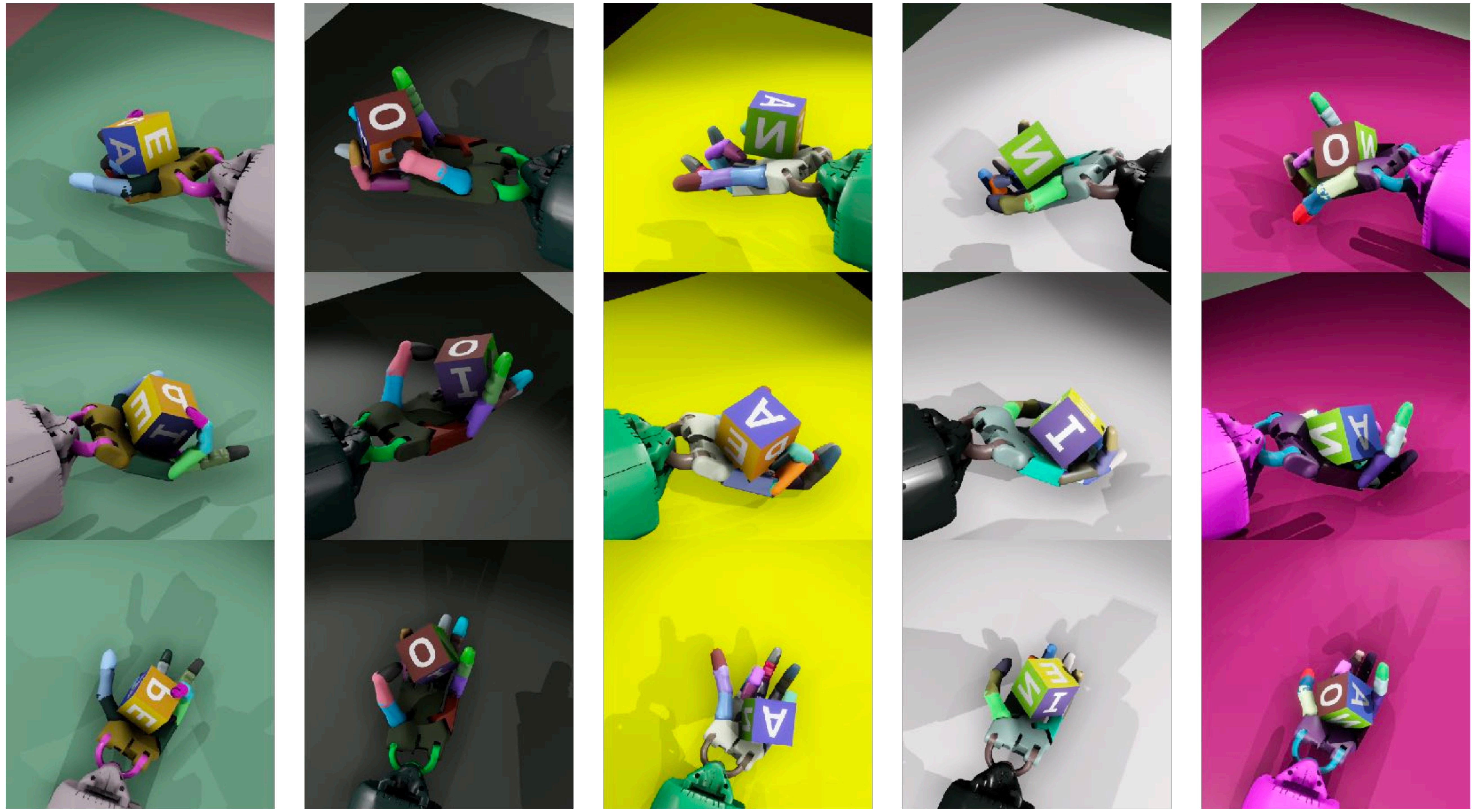
- Problem solving approach of the previous lecture and our discussion up until now:
  - State problem in plain text
  - Use LLM as a general purpose problem solver
  - LLM provides text-based (or code based) solution strategy
  - Execute strategy in a virtual world

- Now let's consider problems where it's less obvious how to describe the problem in text…

Dexterous manipulation







Playing an FPS game…

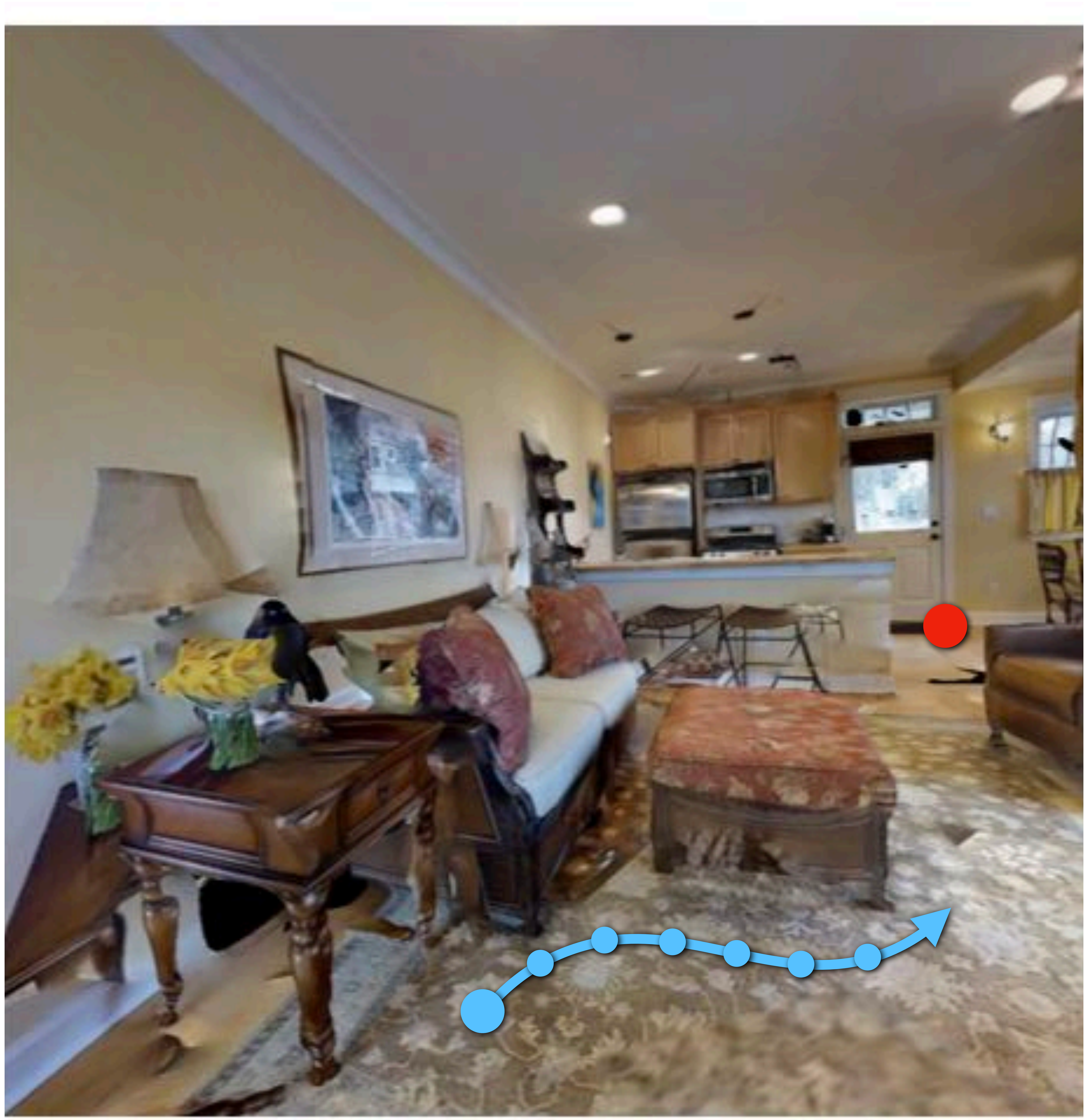

Random incoming ball + Random target

Why learning via trial and error requires a lot of simulated experience (reinforcement learning example)

RL in 30 seconds

Model Inference

environment observation e.g. RGB image → π_θ → **agent action**



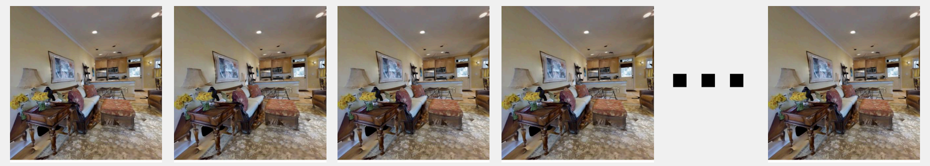
RL in 30 seconds

Model Inference

environment observation e.g. RGB image → π_θ → agent action

Model Training

sequence of observations



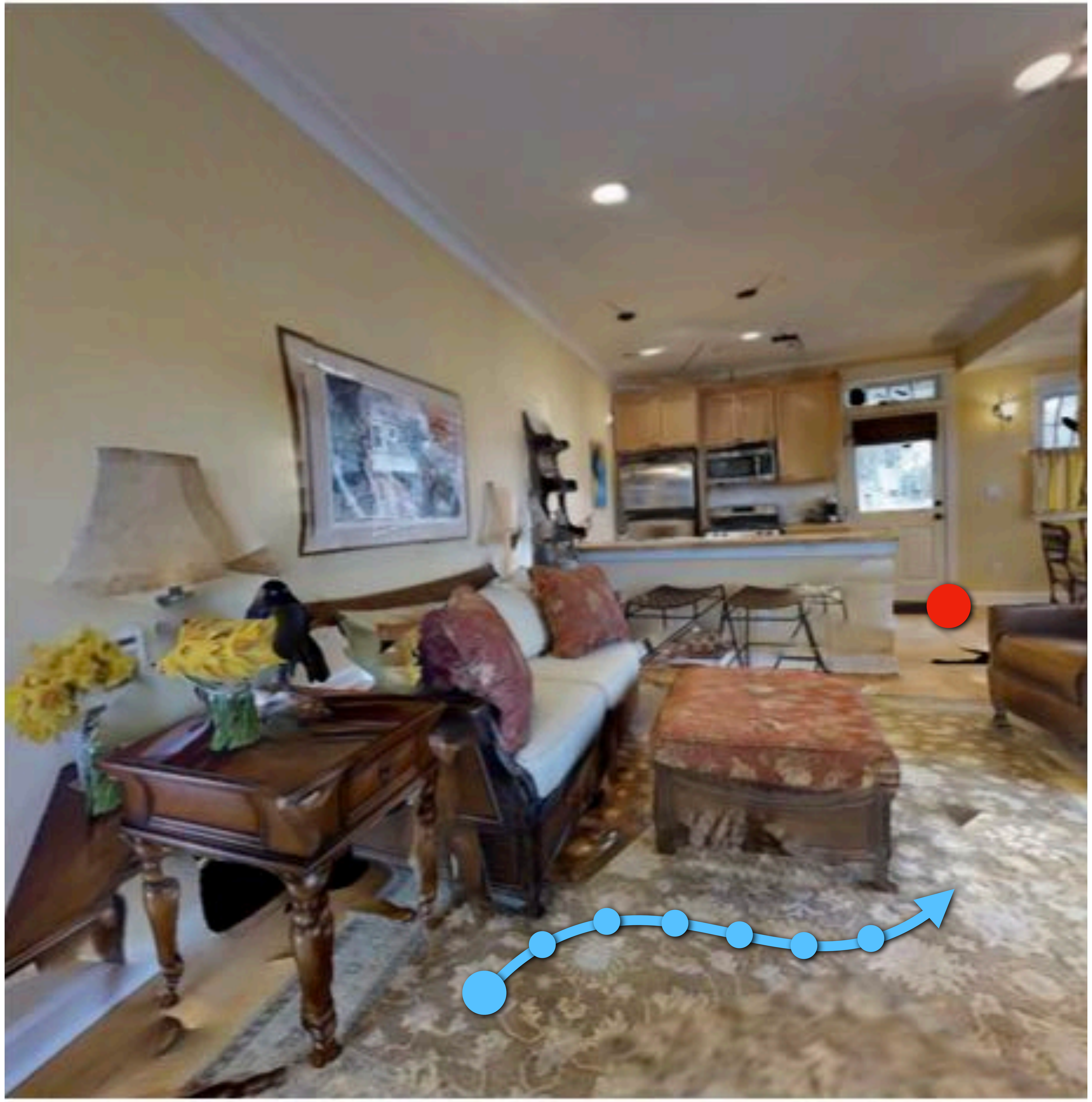
sequence of agent actions

Reward: change in distance from goal

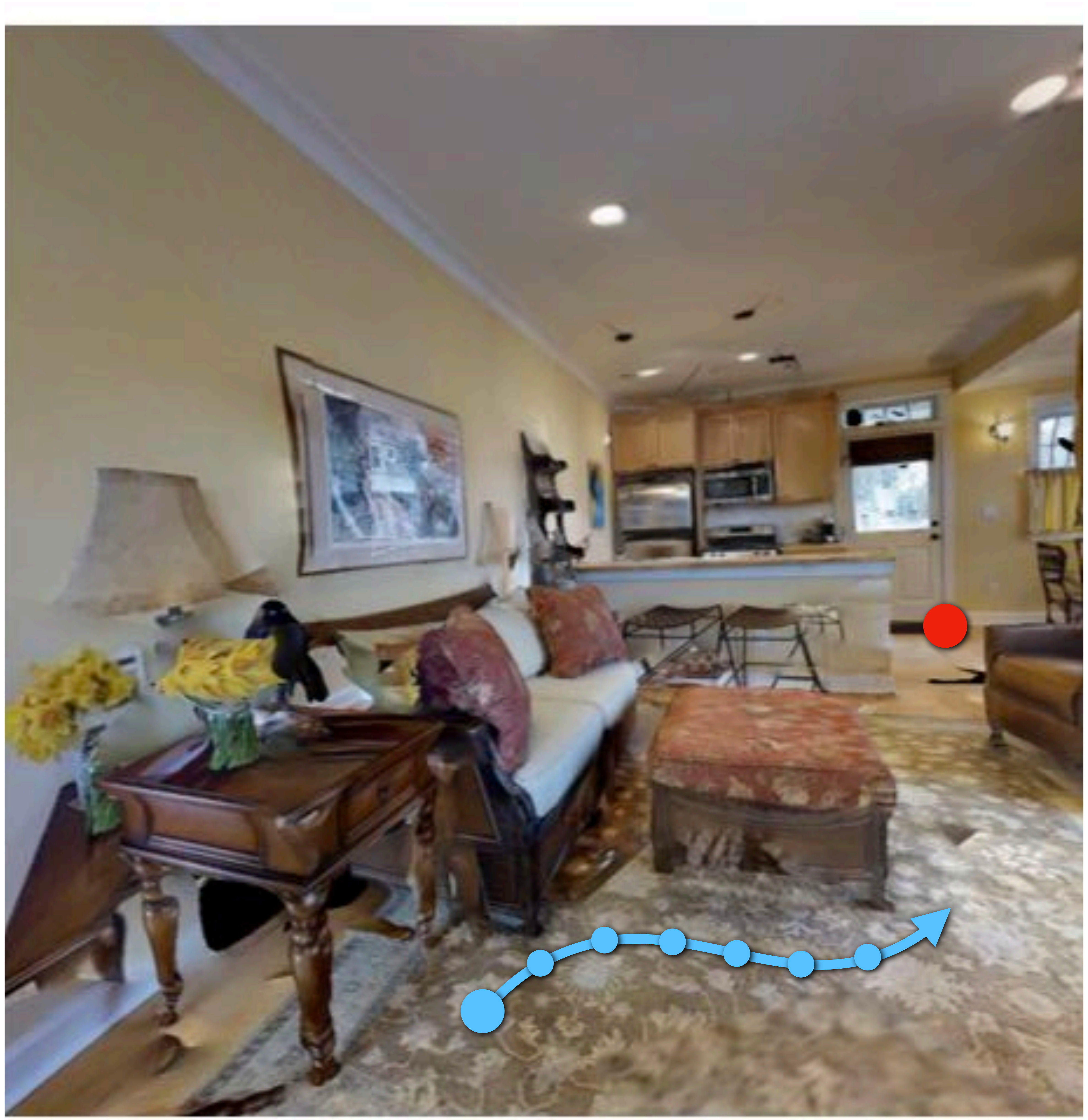
compute loss gradients

π_θ

→ update model via SGD



RL in 30 seconds



Model Inference

environment observation e.g. RGB image → π_θ → agent action

Model Training

Rollout → compute loss gradients π_θ → update model via SGD

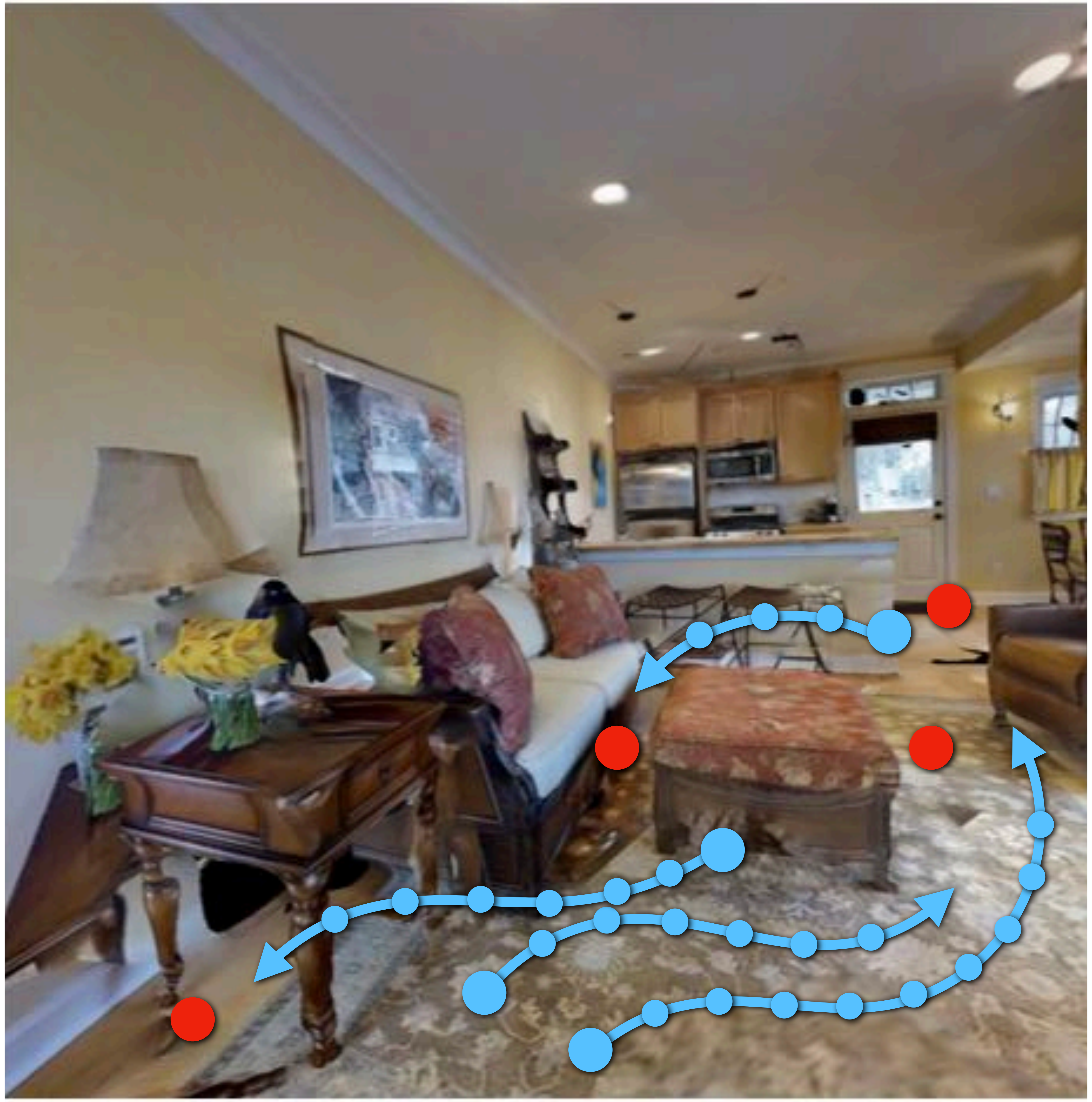
RL in 30 seconds

Many rollouts:
- **Agents independently navigating same environments**

Batch Model Training

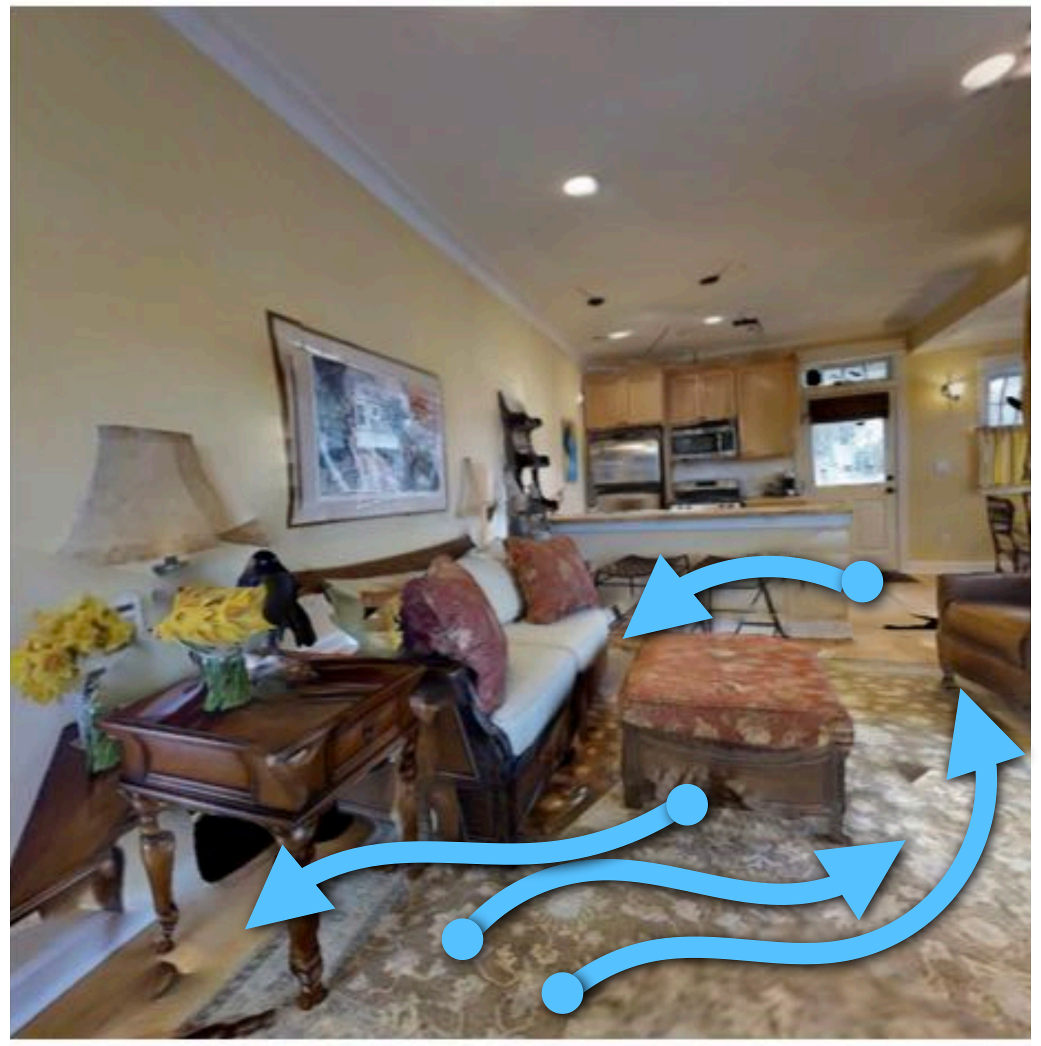
Rollout 0
Rollout 1
Rollout 2
...
Rollout N-1

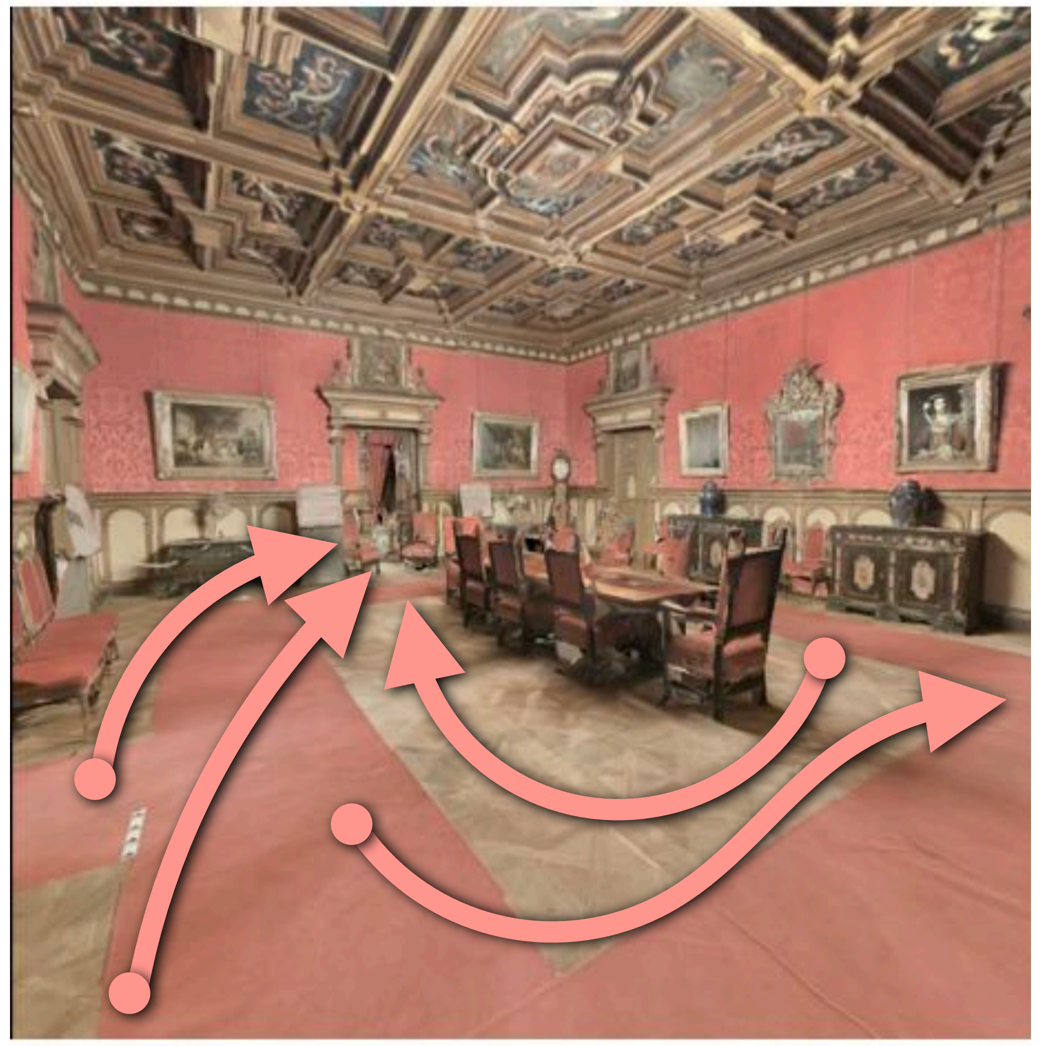
→ compute loss gradients π_θ → update model via SGD



RL in 30 seconds

Many rollouts:
- Agents independently navigating same environments
- Or different environments





Batch Model Training

Rollout 0

Rollout 1

Rollout 2

Rollout 3

Rollout 4

Rollout 5

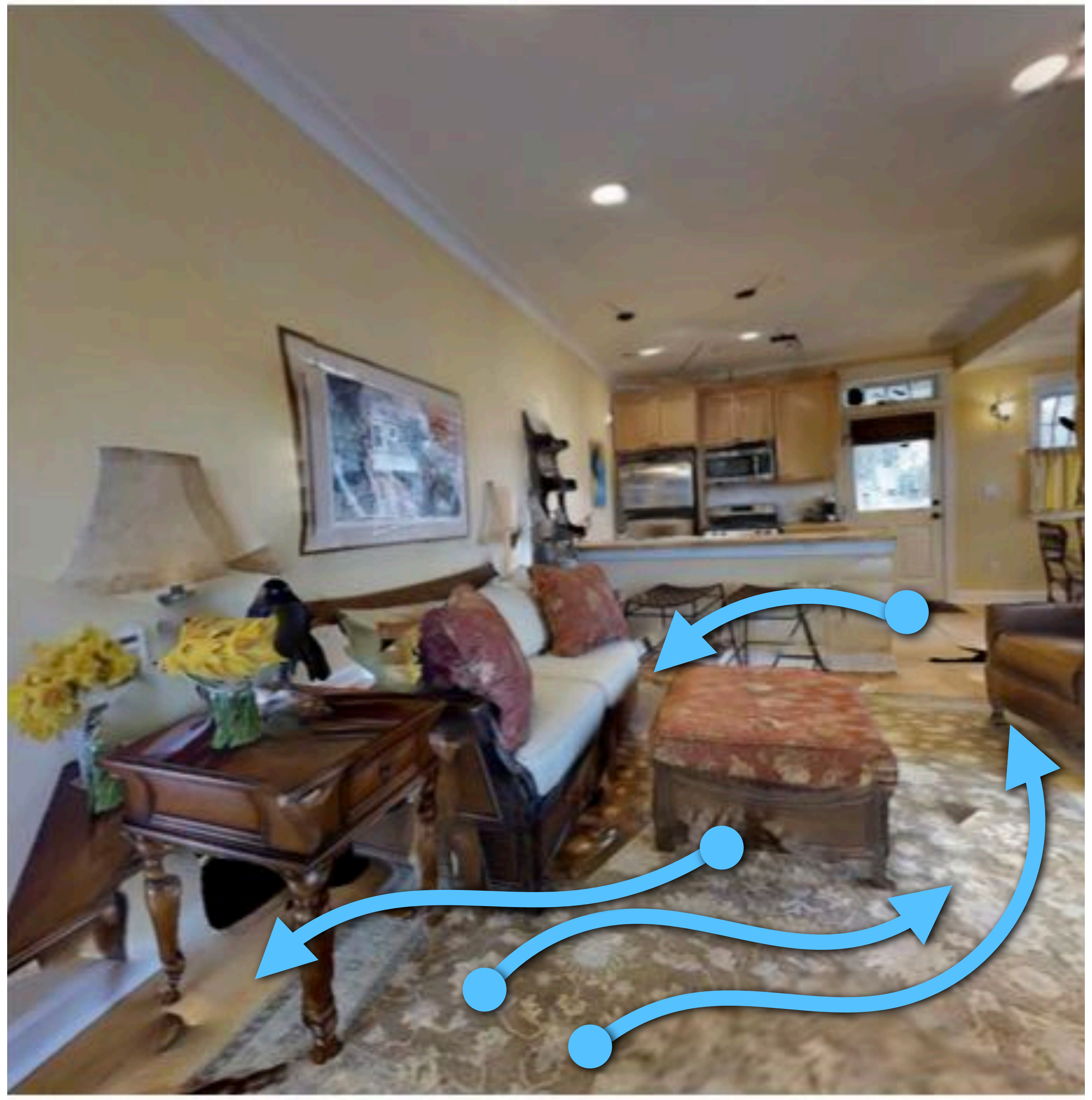
...

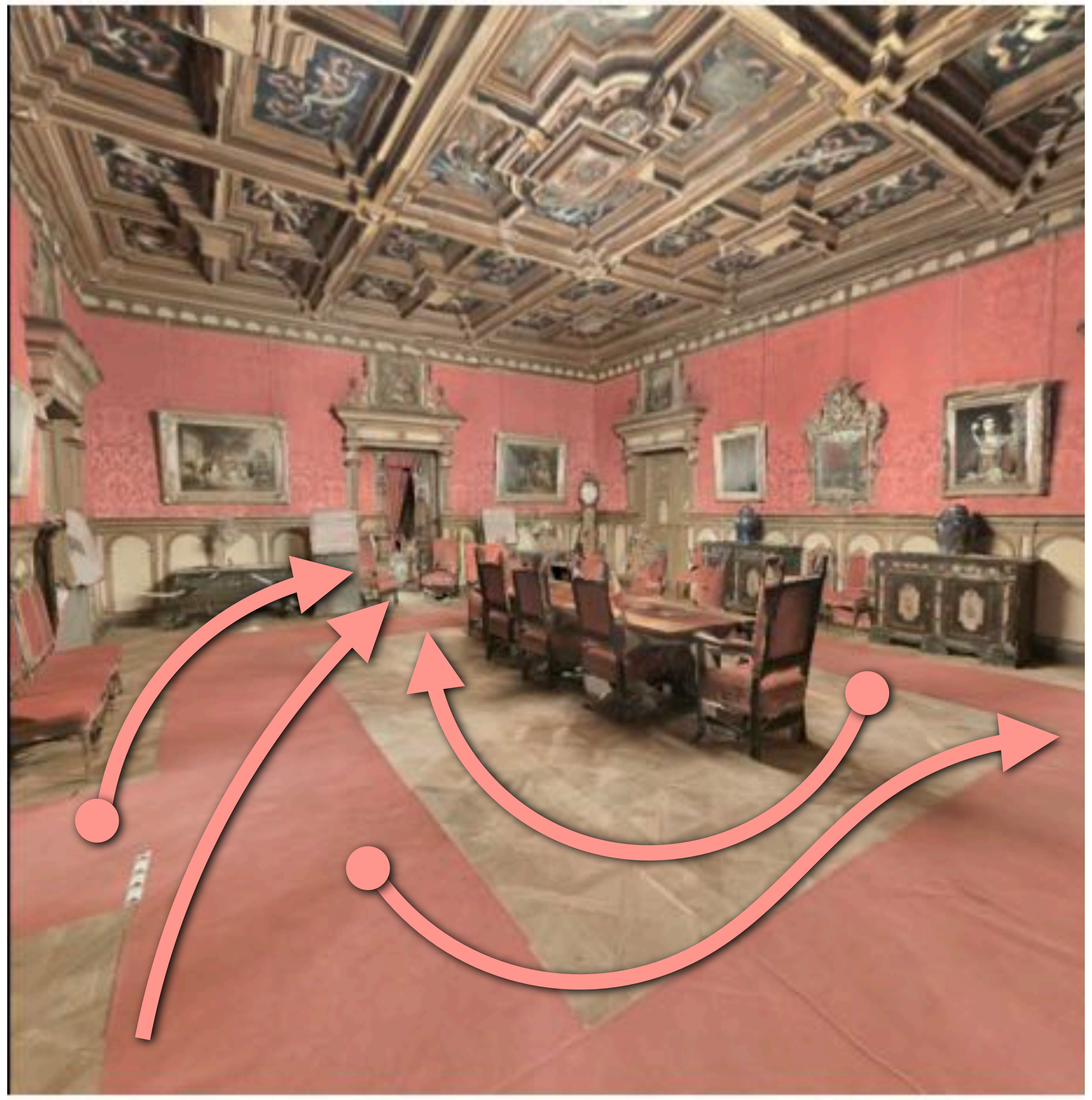
Rollout N-1

→ compute loss gradients π_θ → update model via SGD

Learning skills can require many trials (billions) of learning experience

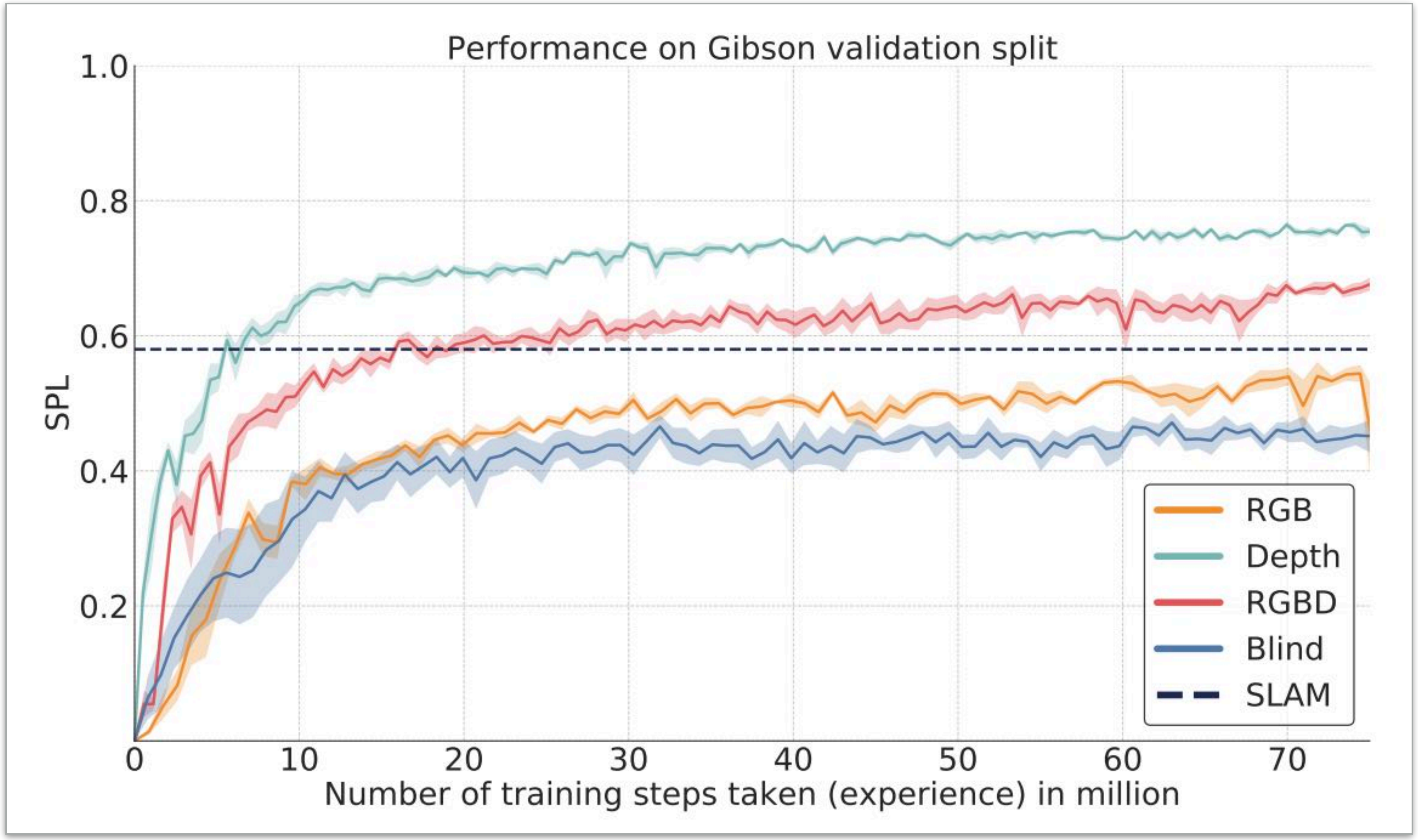
- Training in diverse set of virtual environments
- Many training trials in each environment



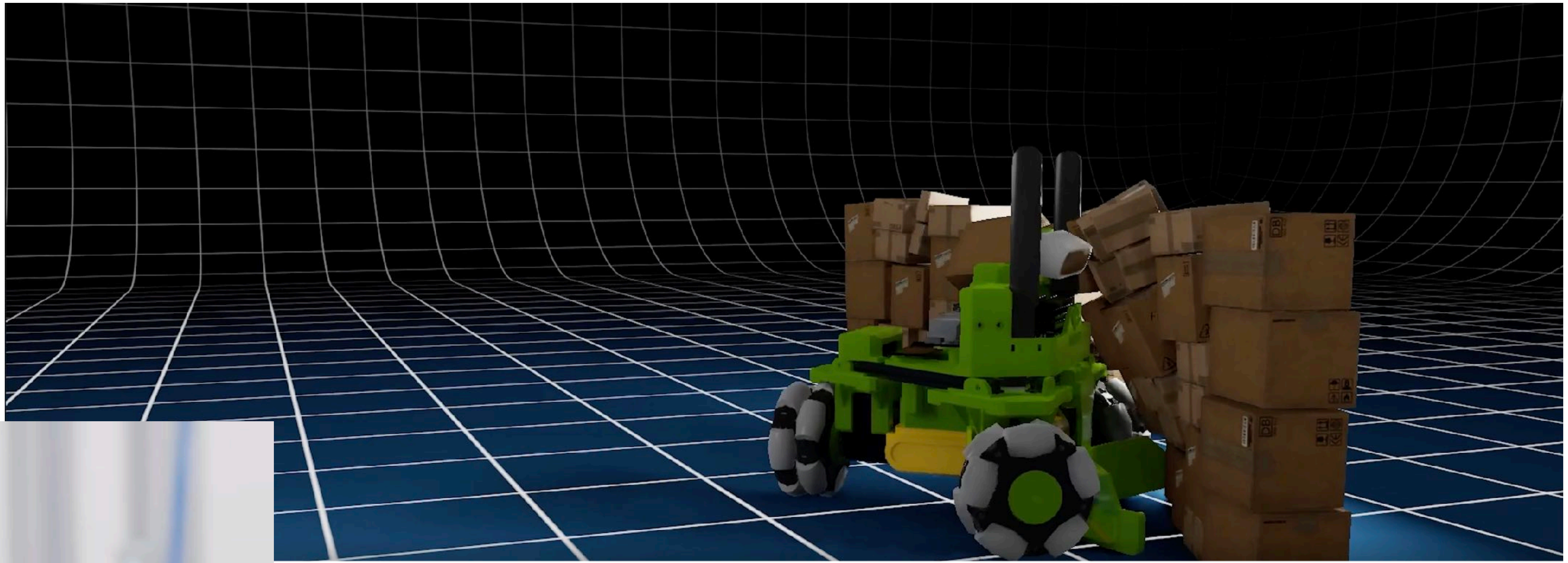


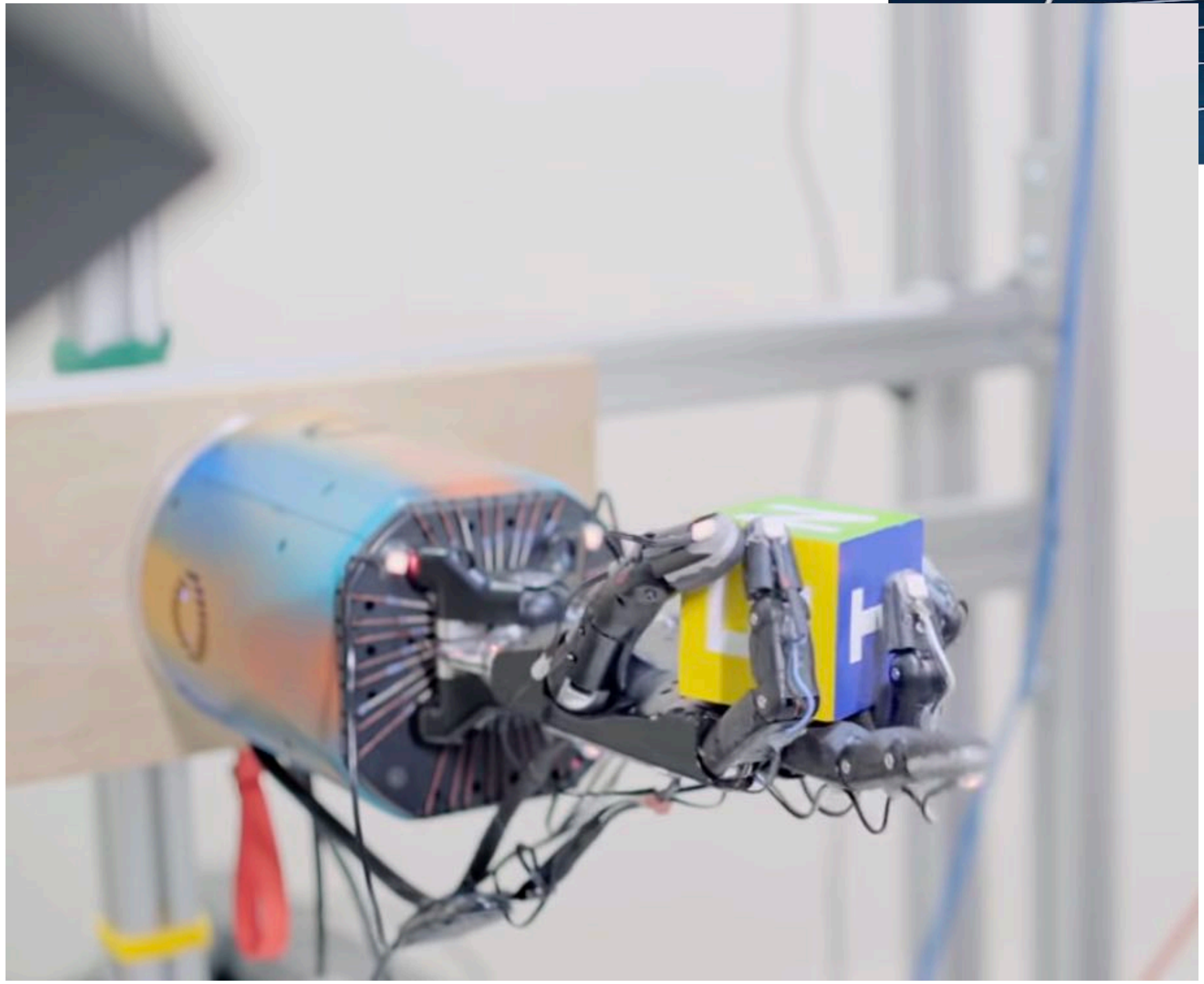
Need significant amounts of simulated expe

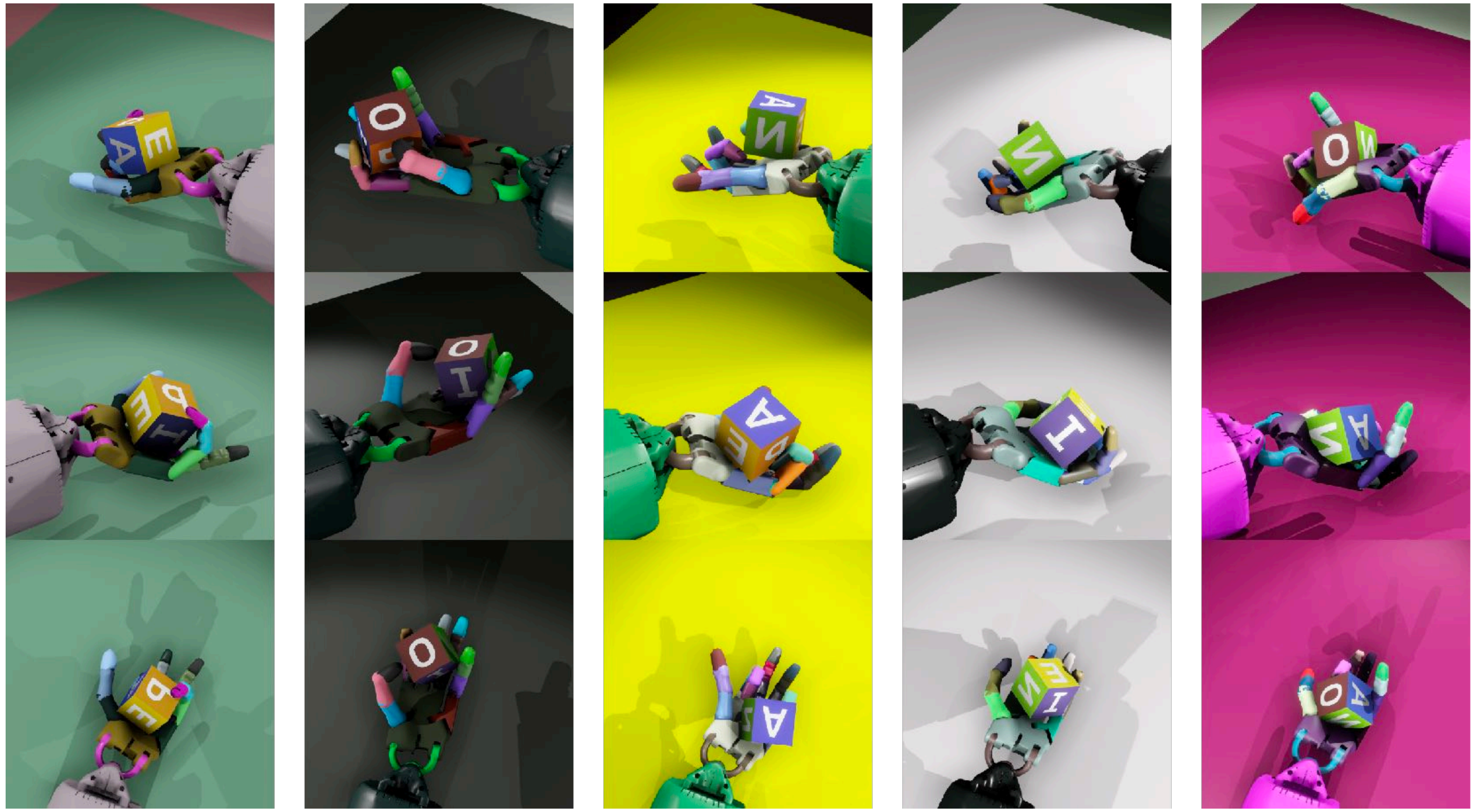
Example: even for simple PointGoal navigation task: need billions of steps of "experience" to exceed traditional non-learned approaches



Accurate
Physics simulation








Random incoming ball + Random target
Practice in simulation…


Video Credit: OpenAI

Many interactive virtual home environments

Navigate to a location

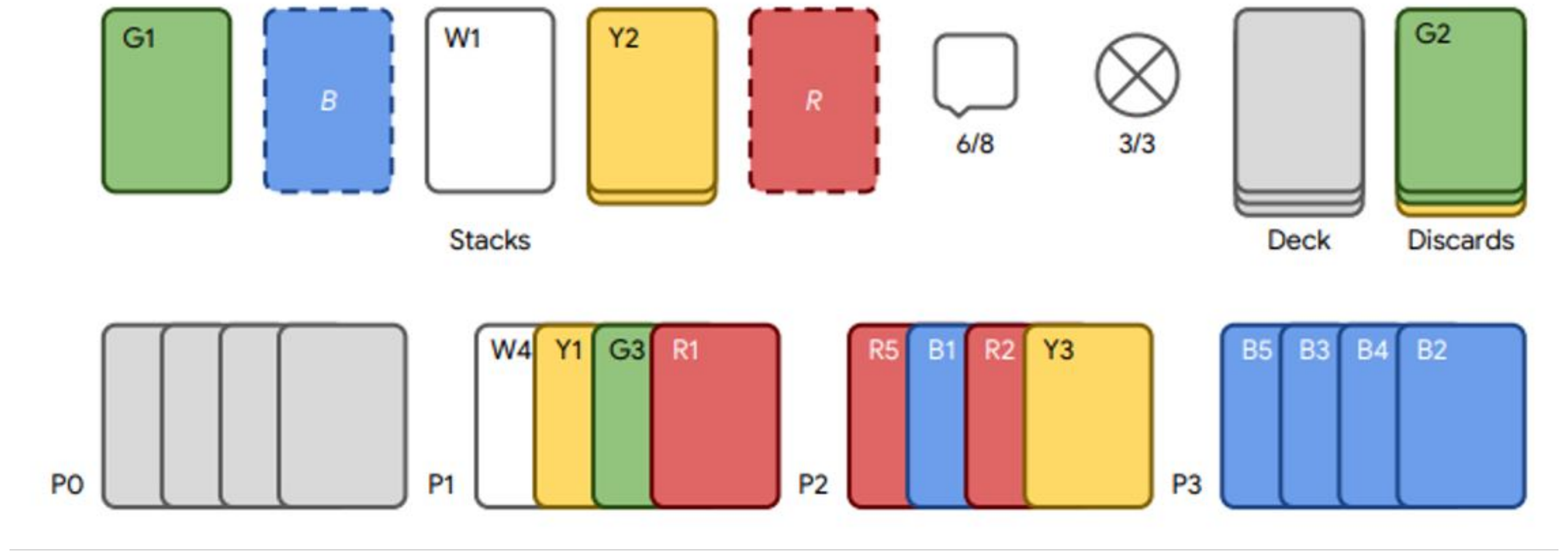
Find an object

Rearrange the room so objects are in desired locations

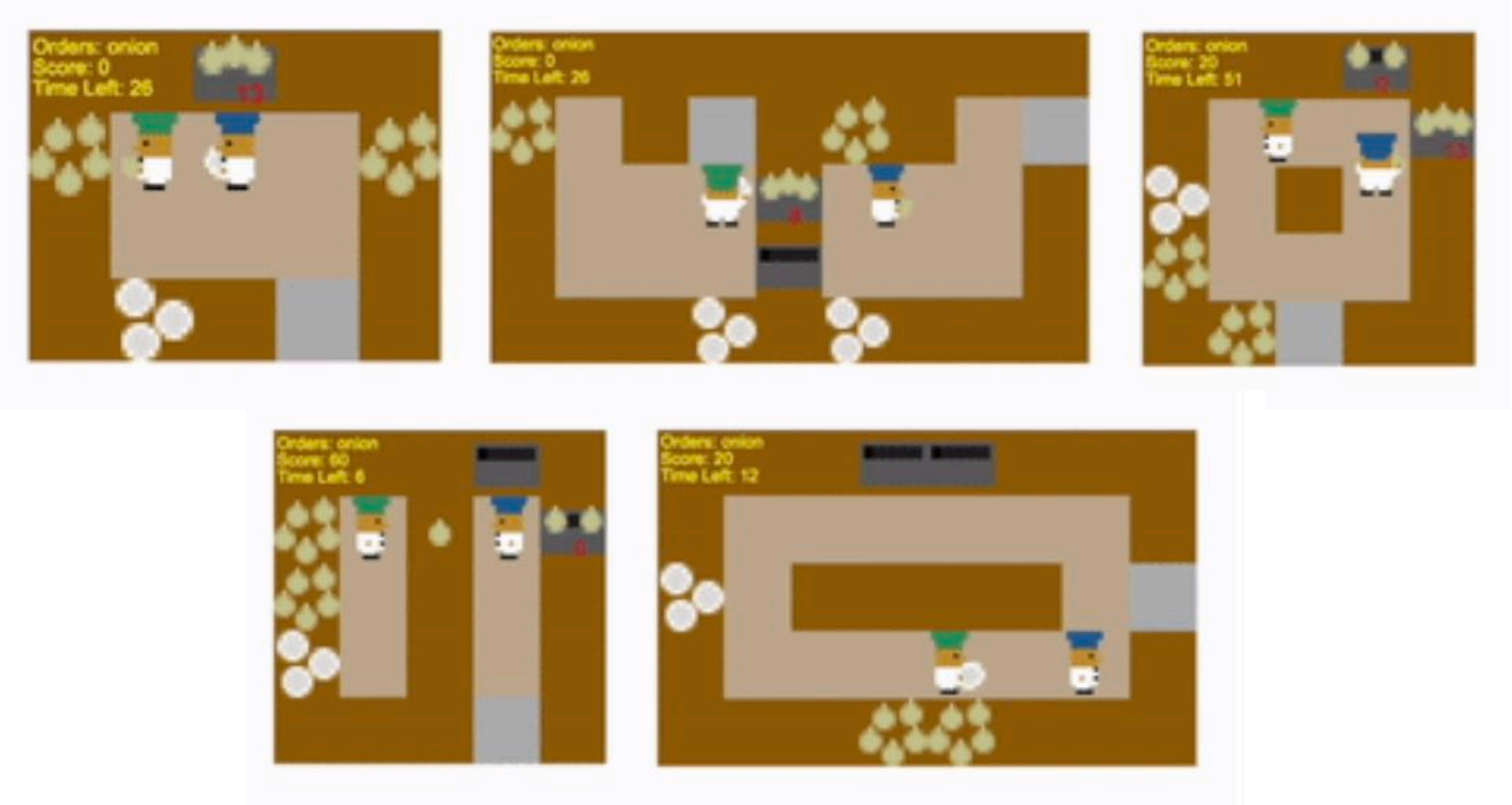
Pour oneself a glass of milk

Multi-agent games

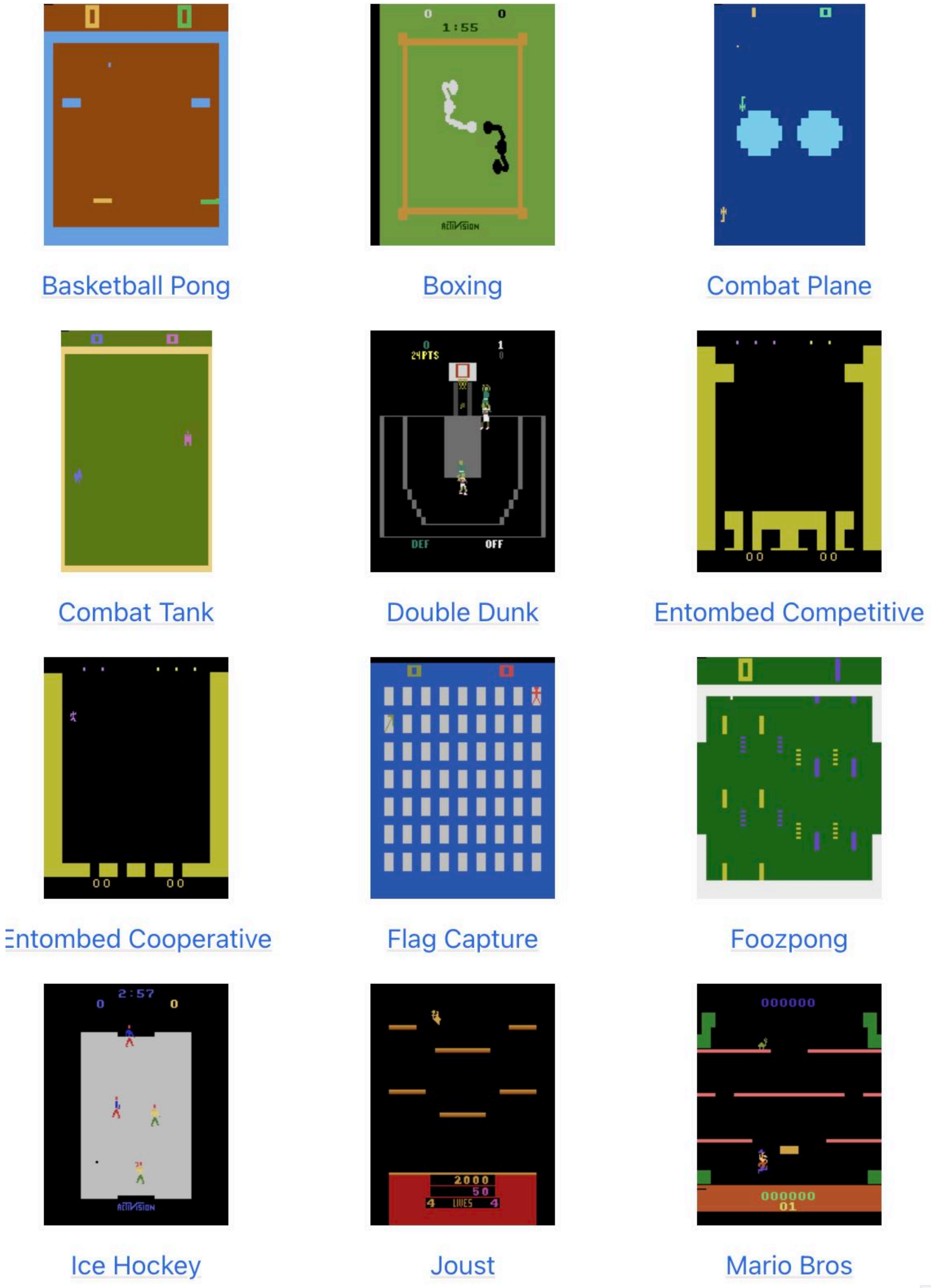
Hanabi (Card Game)



Overcooked (Sims-Like Env)



Atari Games

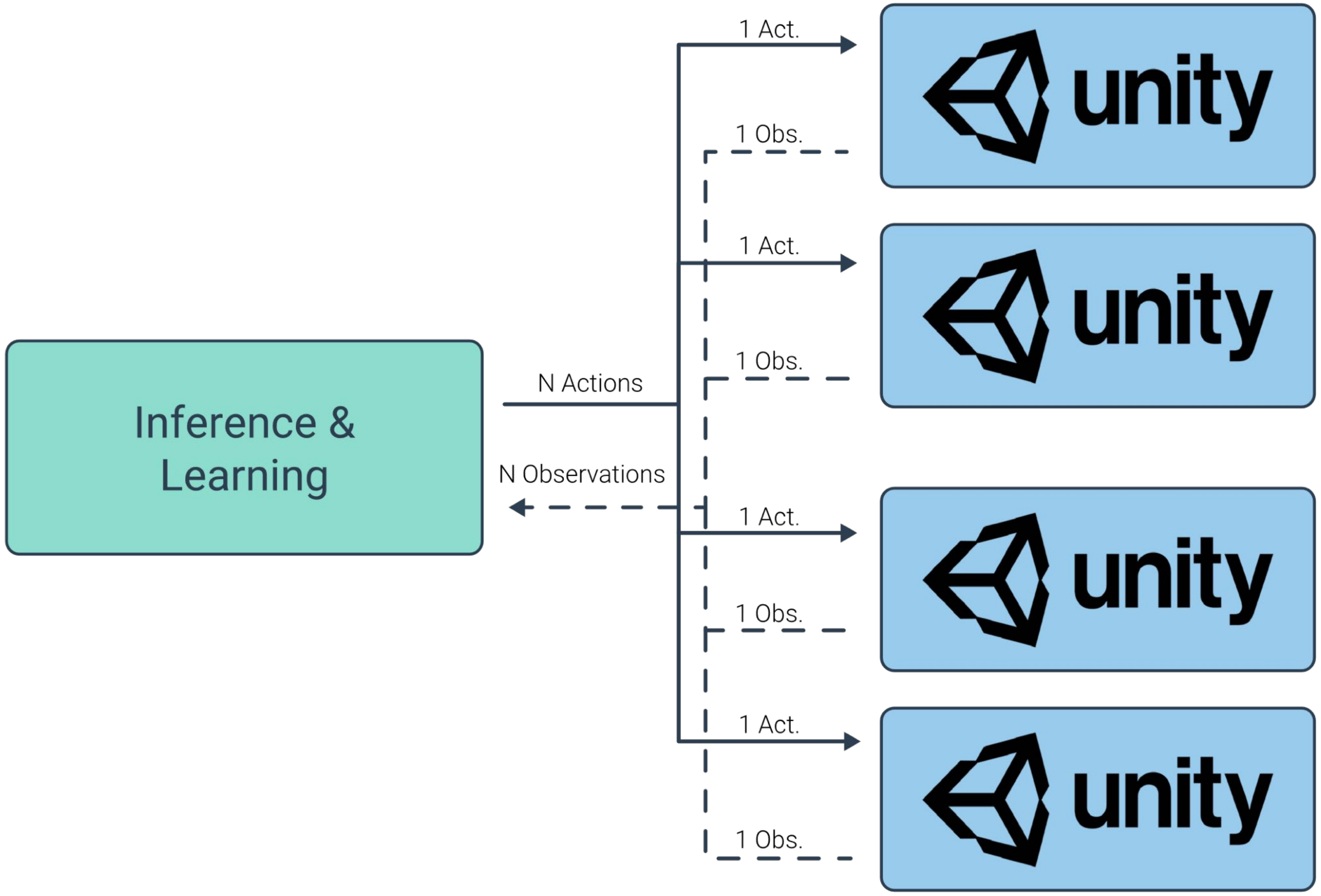


RL workload summary

- Within a rollout
  - For each step of a rollout:
  - Render → Execute policy inference → simulate next world state

- Across many independent rollouts
  - Simulated agents may (or may not) share scene state
  - Diversity in scenes in a batch of rollouts is desirable to avoid overfitting, sample efficiency of learning

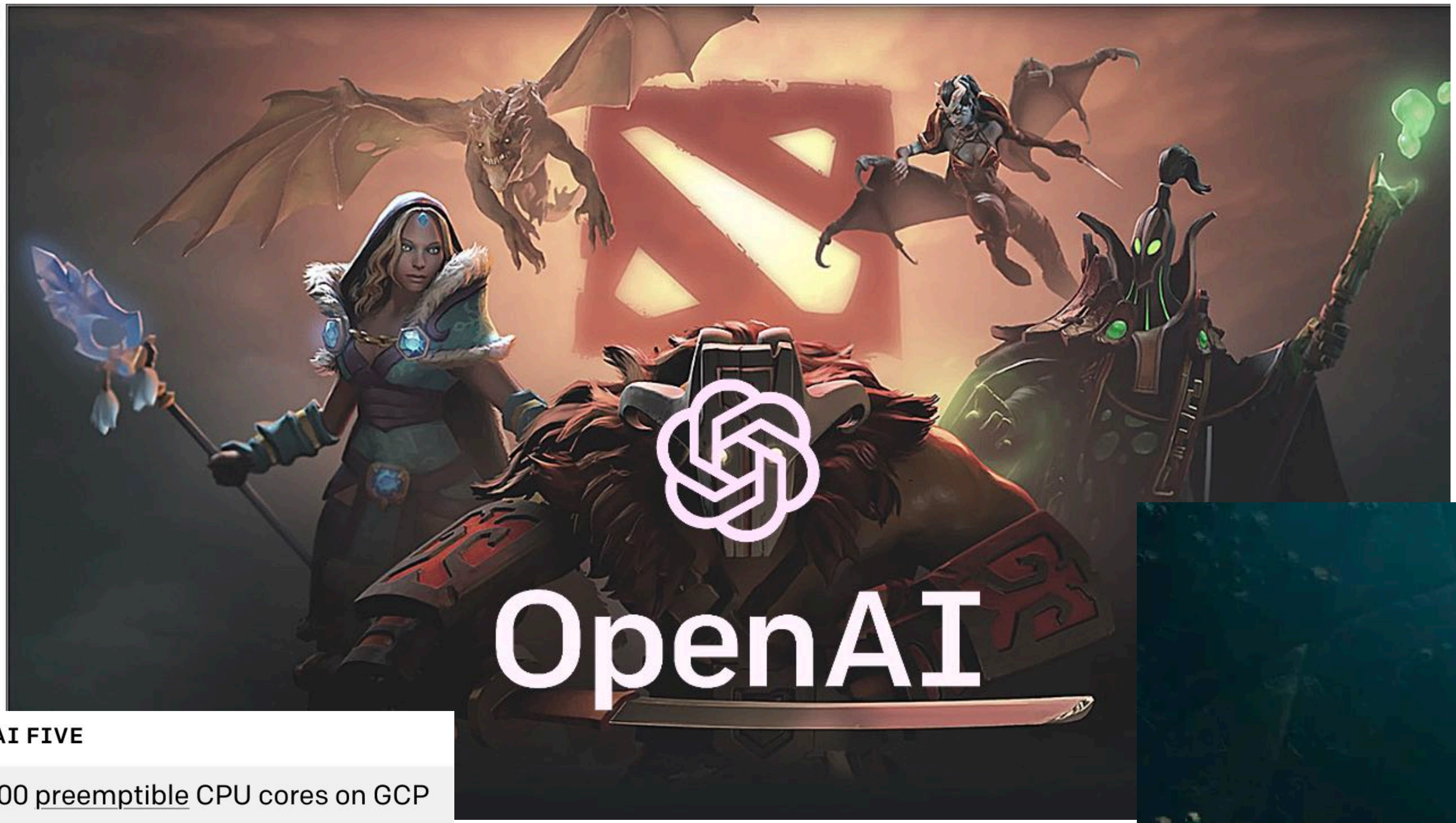
Common simulation approach: treat simulator as a black box, gain high throughput via scale-out parallelization



Treat existing simulation engines as a black box.

Run many copies of the black box in parallel.

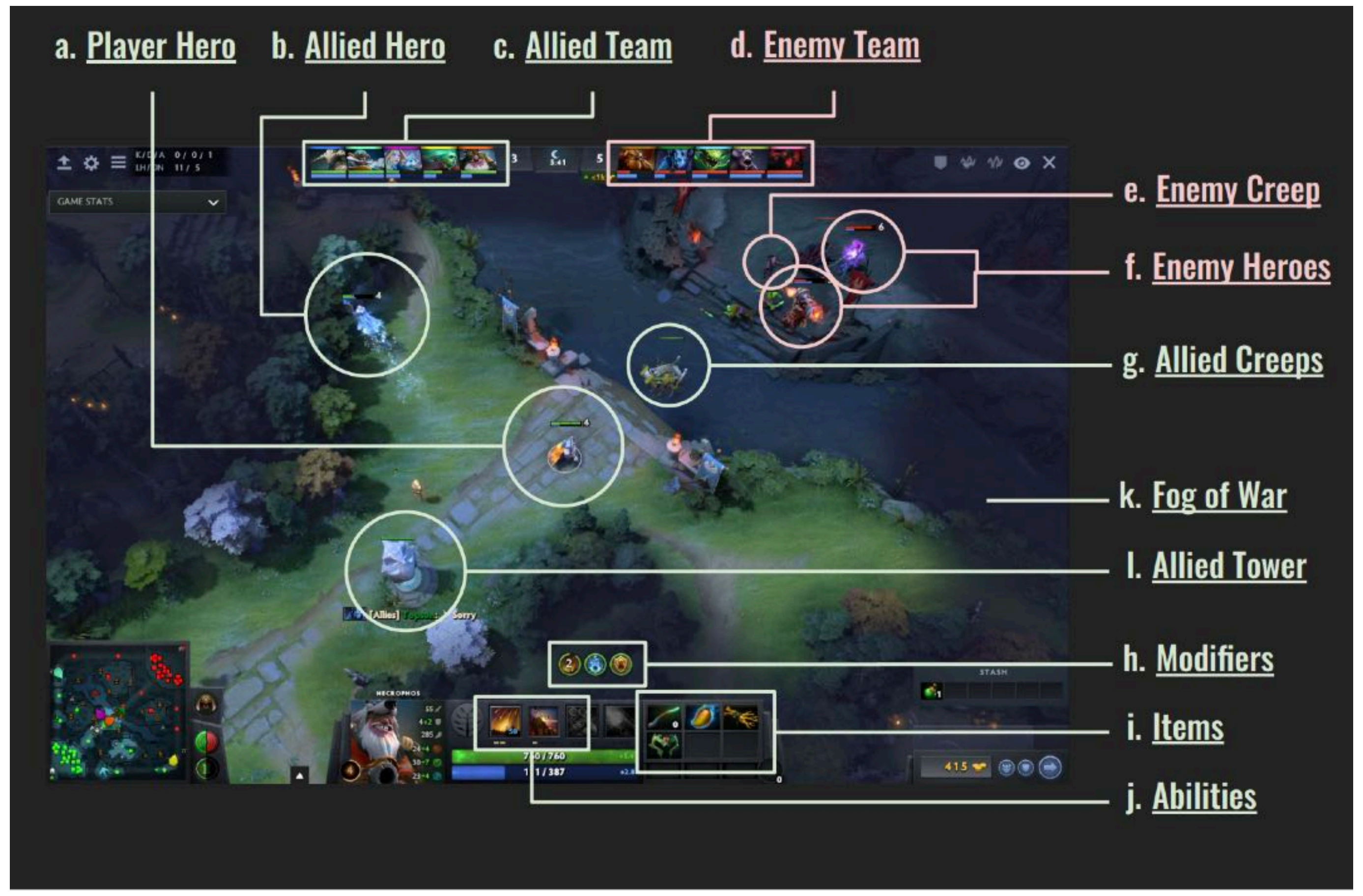
OpenAI's "OpenAI 5" Dota 2 bot



	OPENAI FIVE
CPUs	128,000 preemptible CPU cores on GCP
GPUs	256 P100 GPUs on GCP
Experience collected	~180 years per day (~900 years per day counting each hero separately)
Size of observation	~36.8 kB
Observations per second of gameplay	7.5
Batch size	1,048,576 observations
Batches per minute	~60

Generating simulated experience is computationally demanding

OpenAI Five

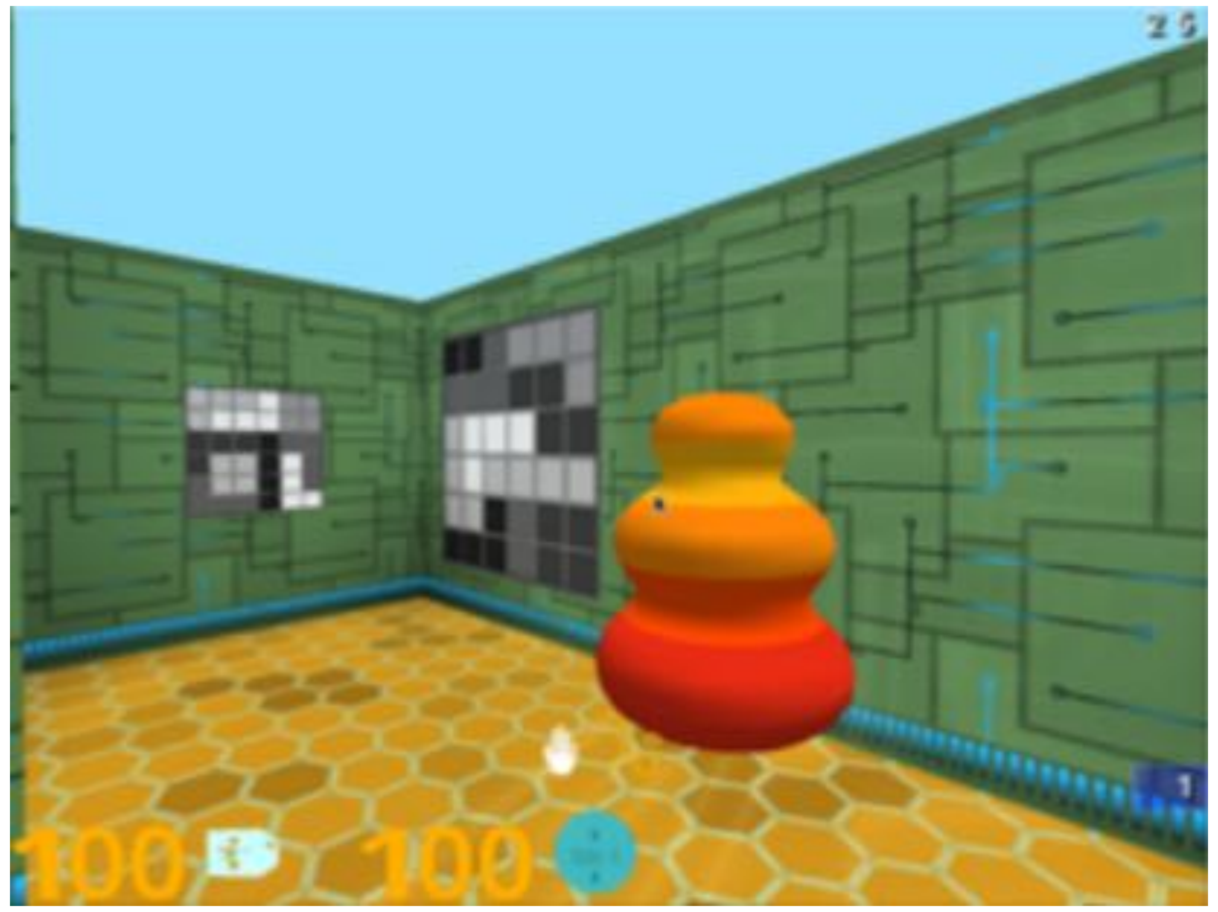


Rapid: 128,000 CPUs,
days of training

Navigation in 3D scanned environments

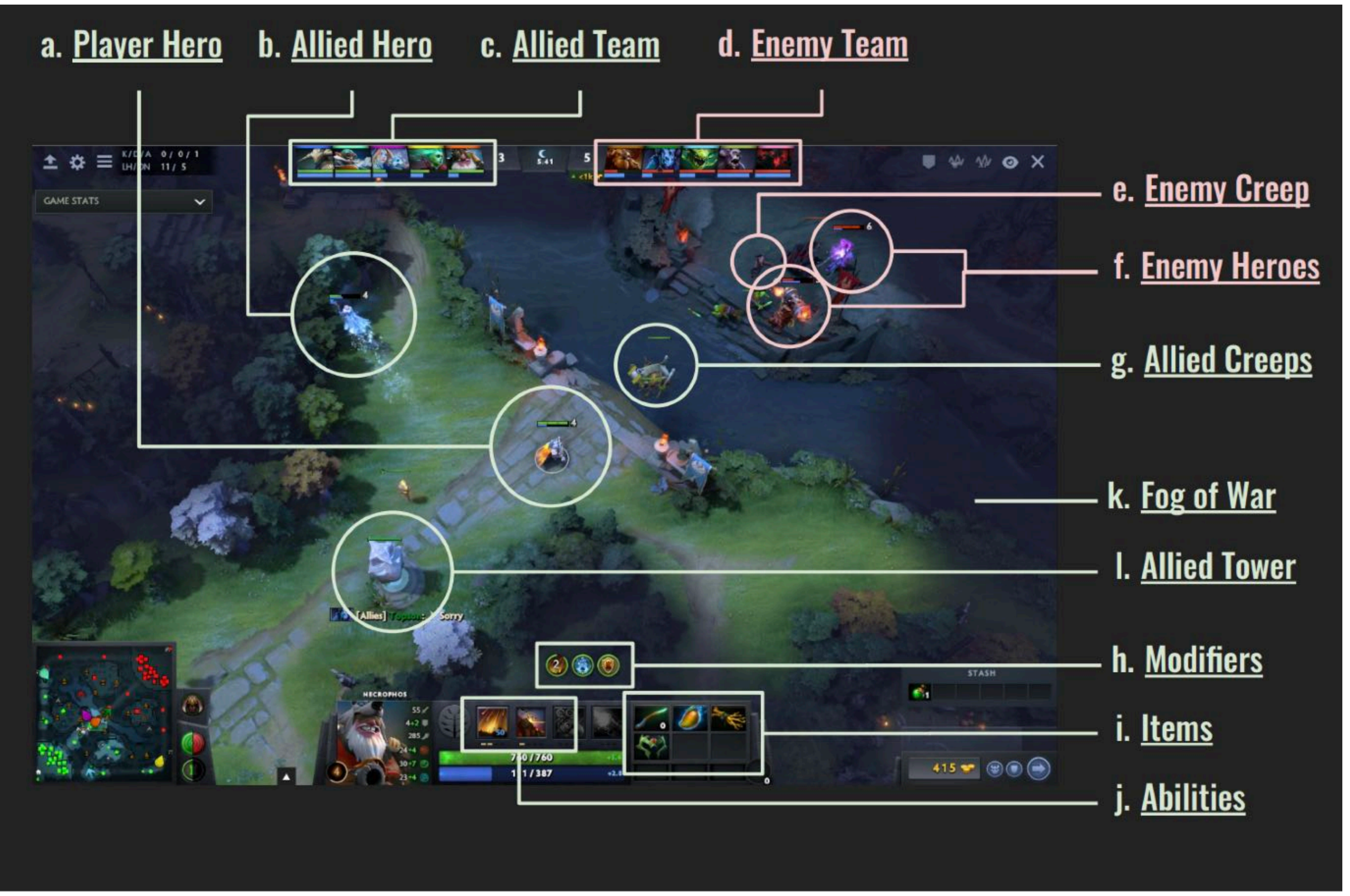
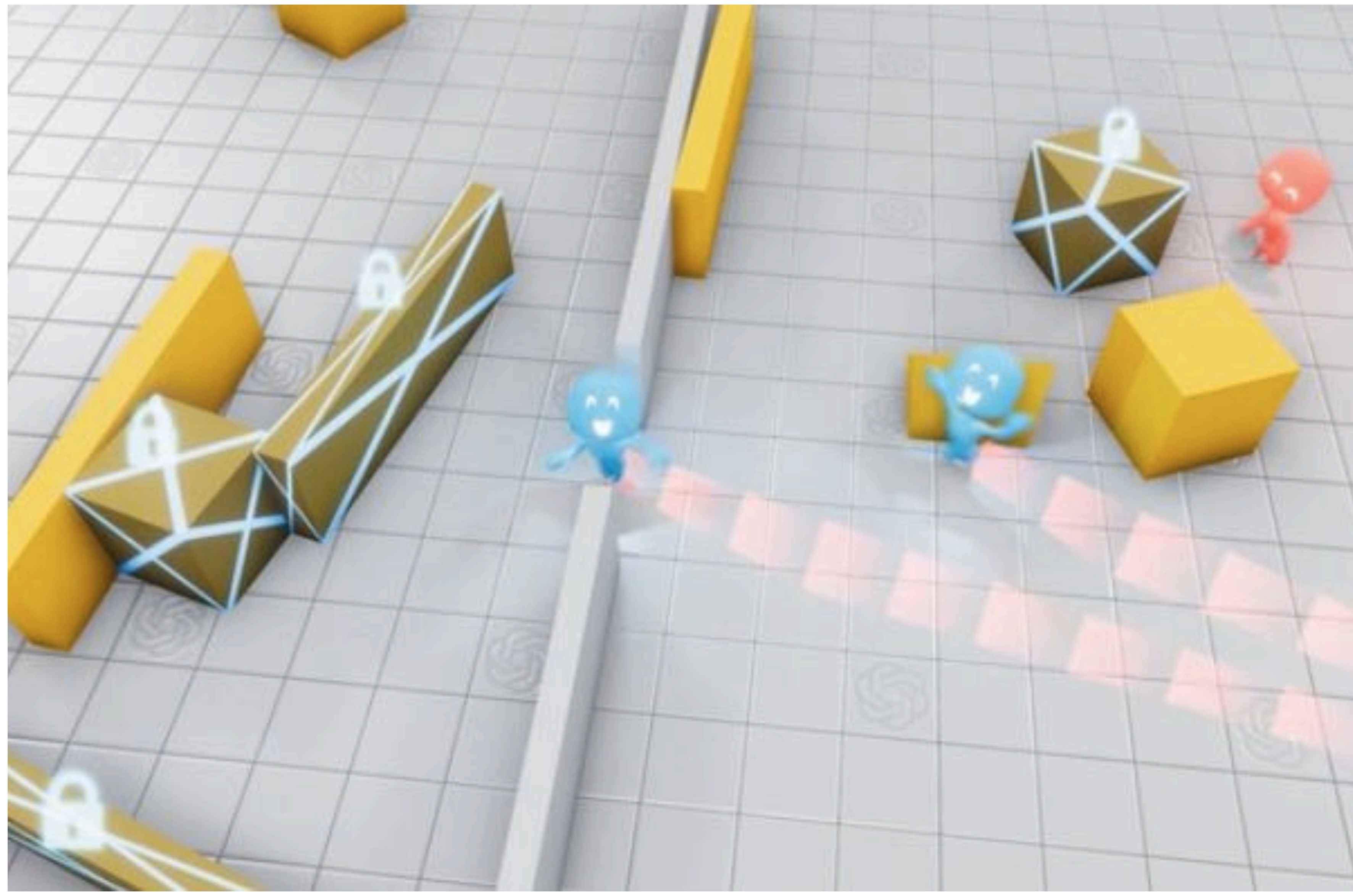
64 GPUs over 2.5 days
(2B experience samples)

Game playing



Deepmind Lab Training with
4000 CPUs and 64 TPUs

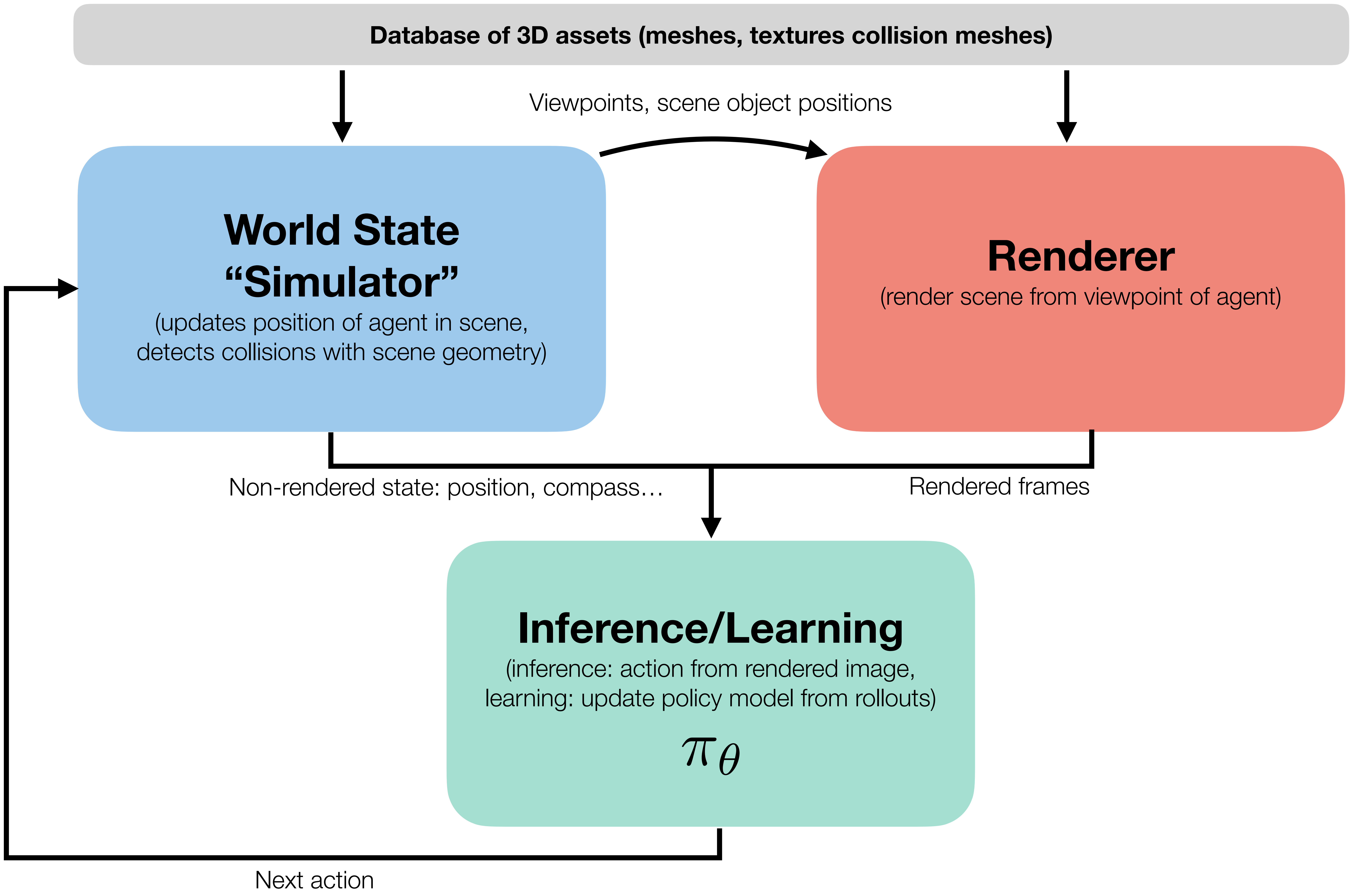
Large-scale agent training is expensive!

OpenAI Five	Robotics in Virtual World	OpenAI Hide and Seek
		

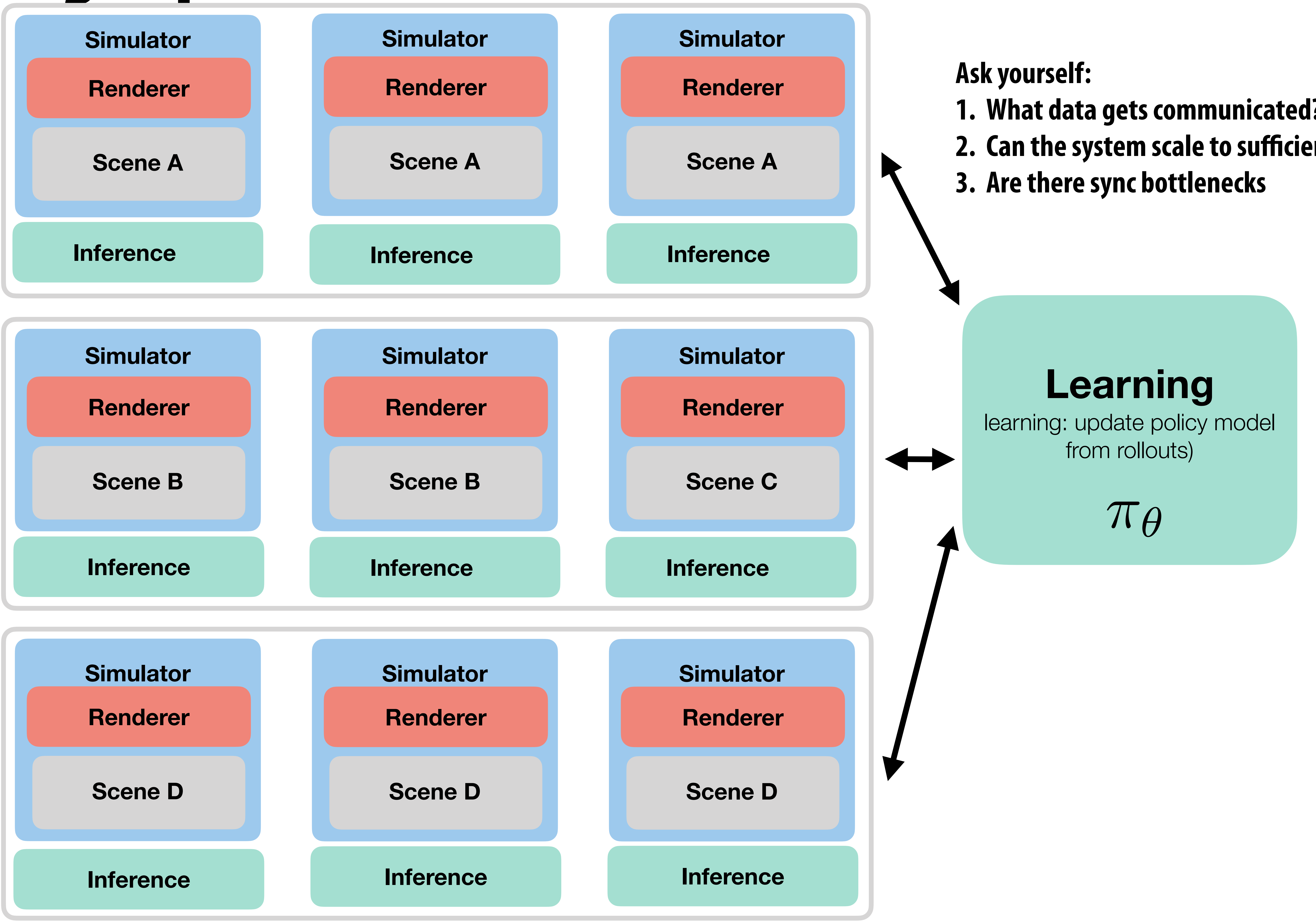
Learning Dota 2: Months of training	64 GPUs over 2.5 days (2B experience samples)	High-level strategies emerge after billions of world time steps

CPUs	128,000 preemptible CPU cores on GCP
GPUs	256 P100 GPUs on GCP
Experience collected	~180 years per day (~900 years per day counting each hero separately)

Example: PointGoal navigation task system components



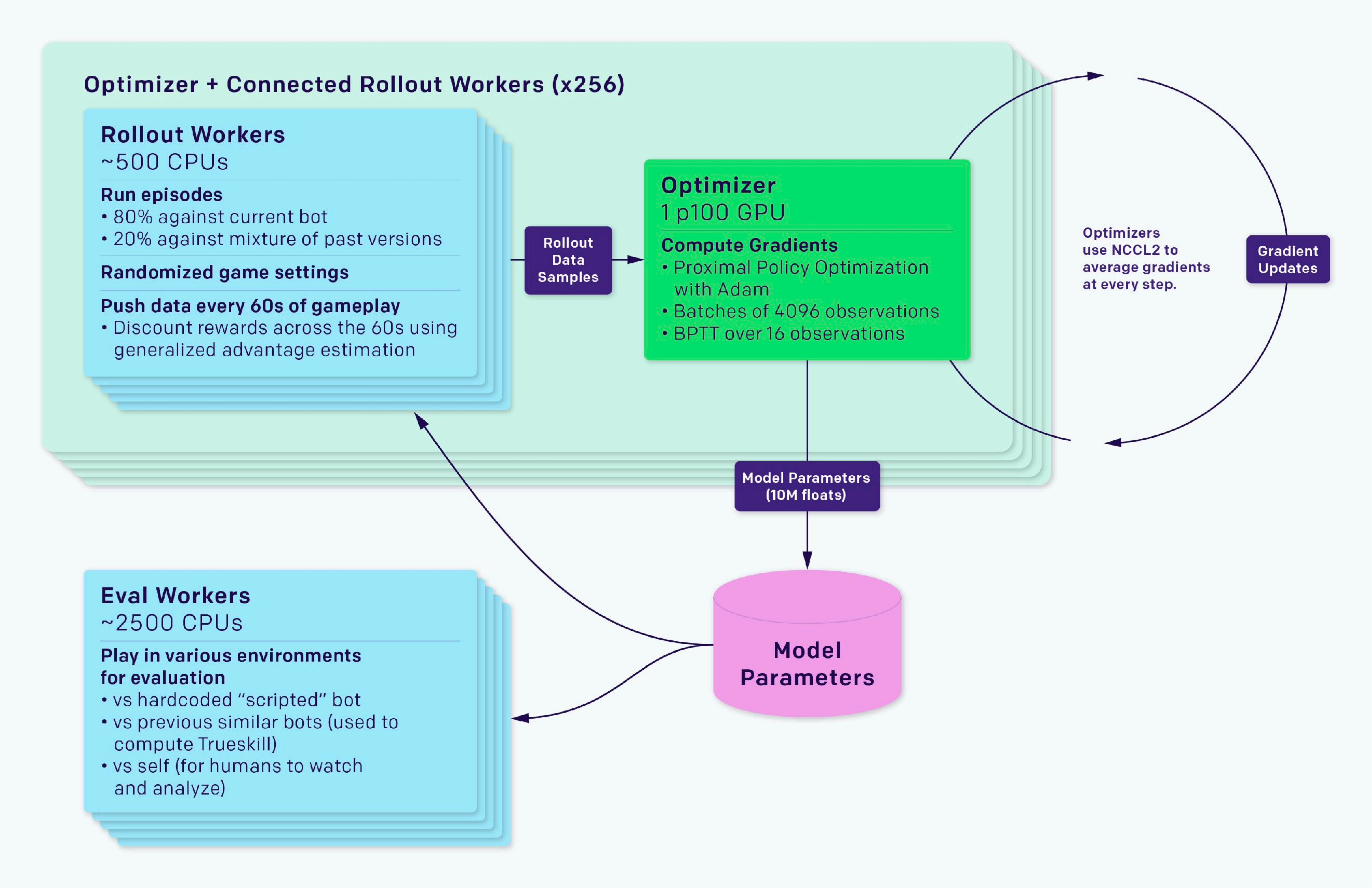
Basic design: parallelize over workers



Ask yourself:

1. What data gets communicated?
2. Can the system scale to sufficient parallelism?
3. Are there sync bottlenecks

Example: Rapid (OpenAI)



What modern graphics engines are designed to render

4K image outputs

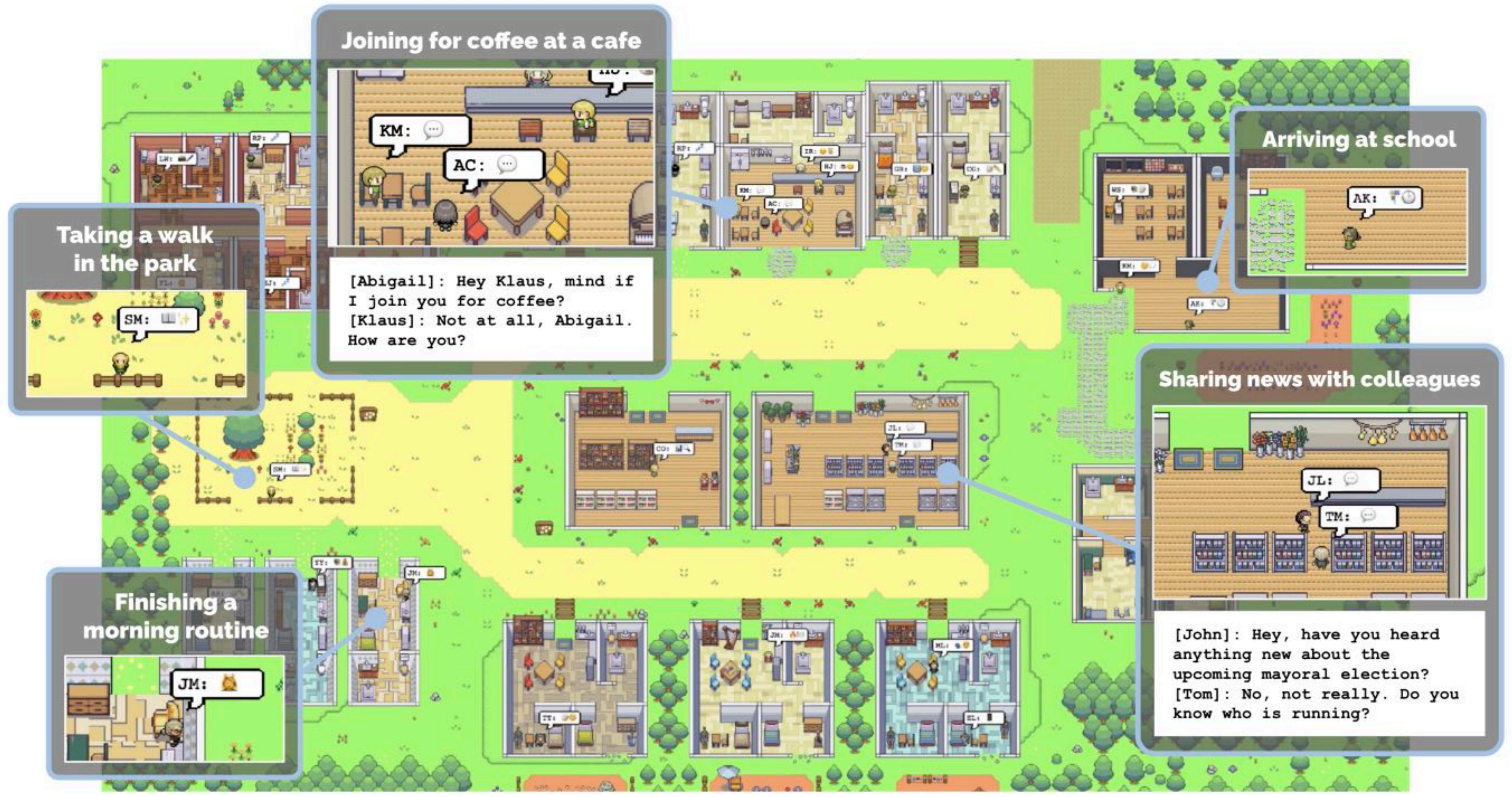
30-60 fps

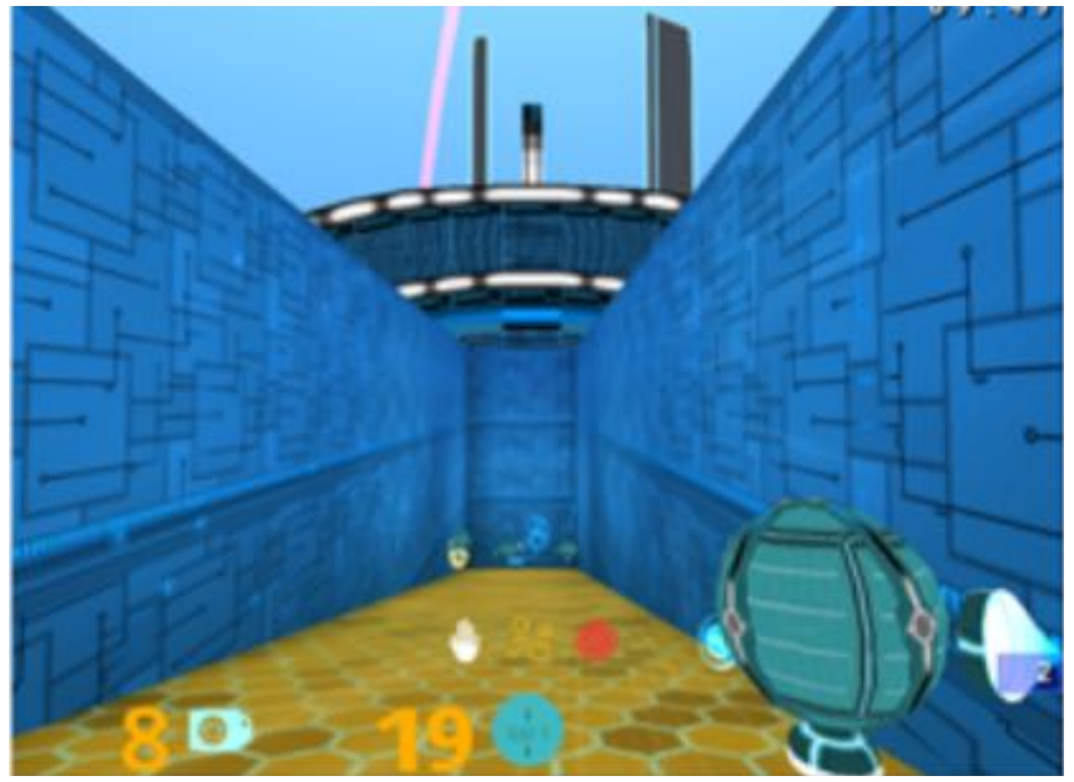
Advanced lighting and material simulation

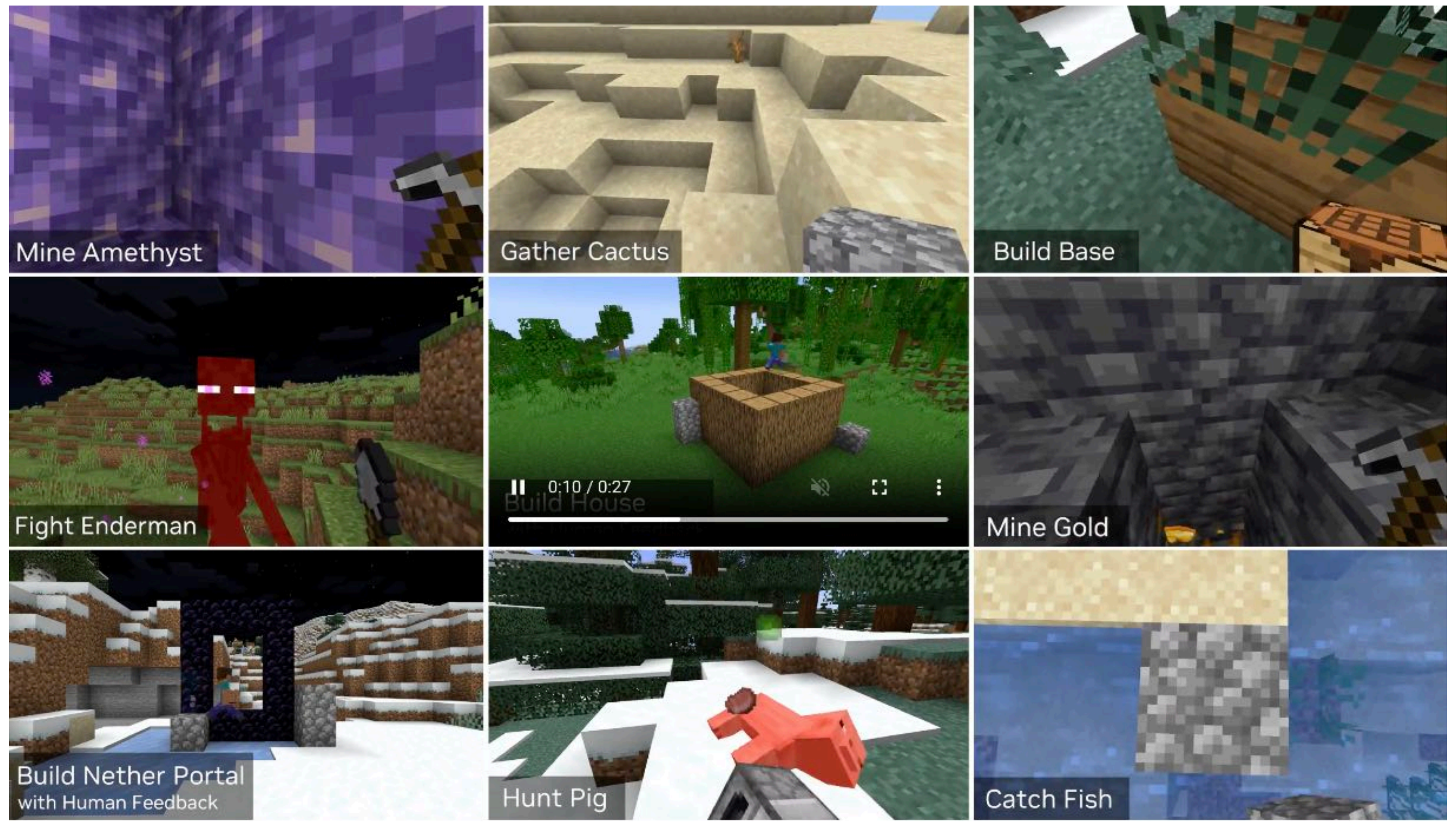
Forza-Horizon-5

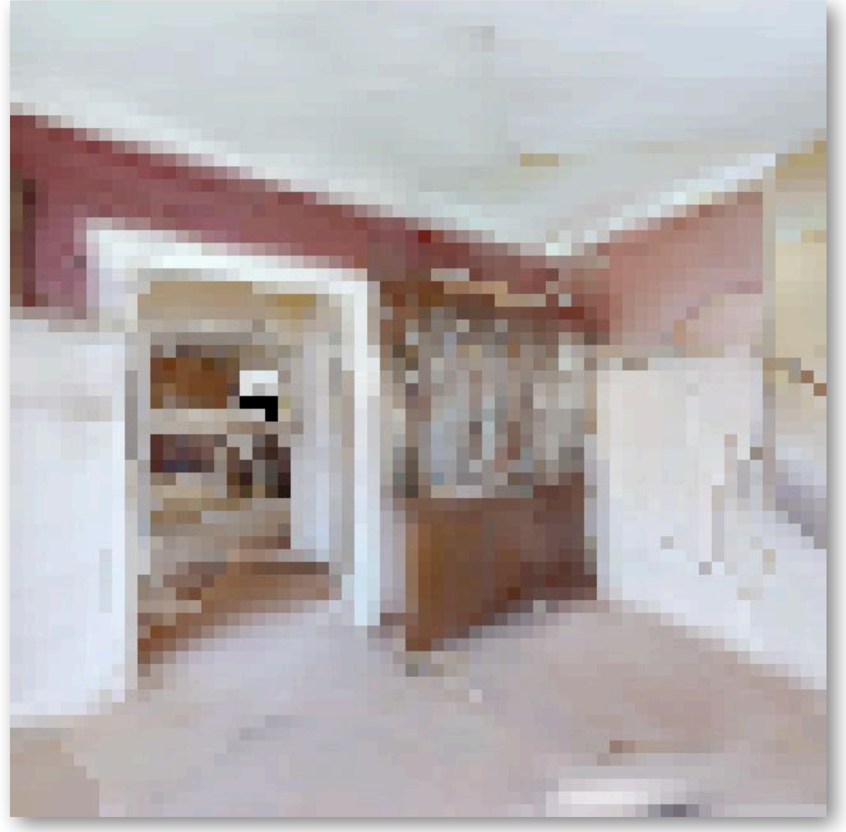
Unreal Nanite Demo

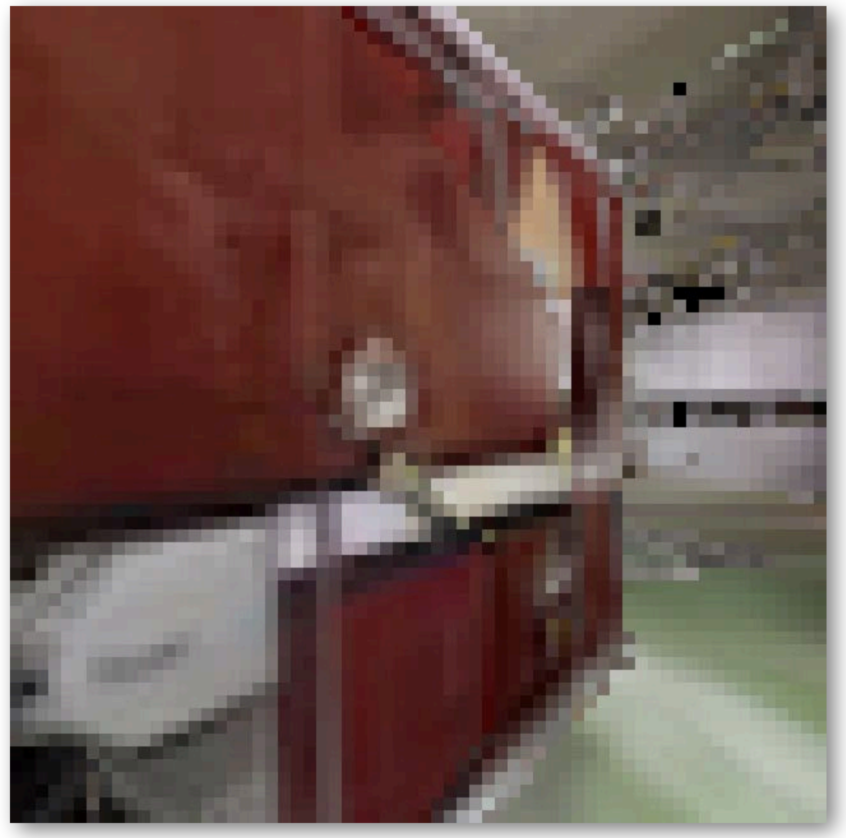
- **Example renderings from common RL learning environments**
  - Low resolution
  - Simple lighting/shading

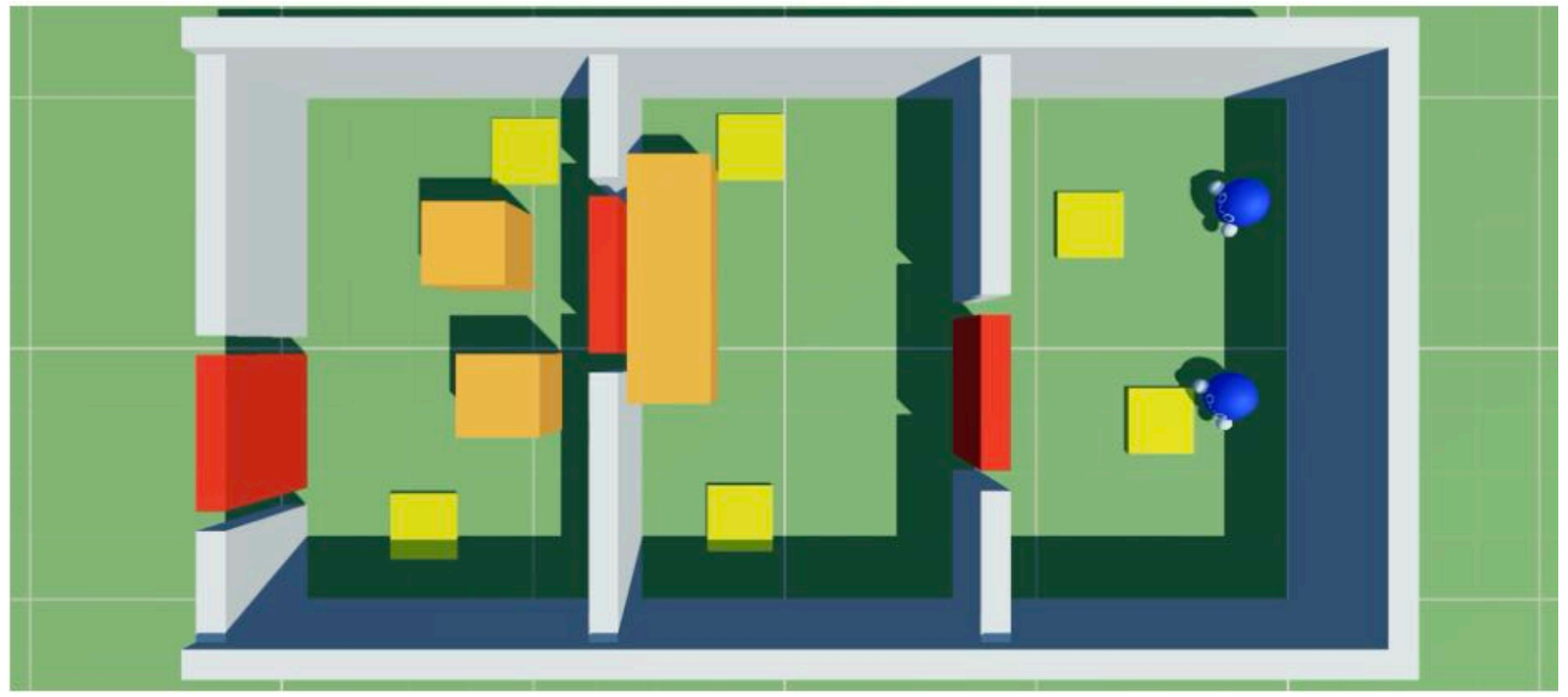


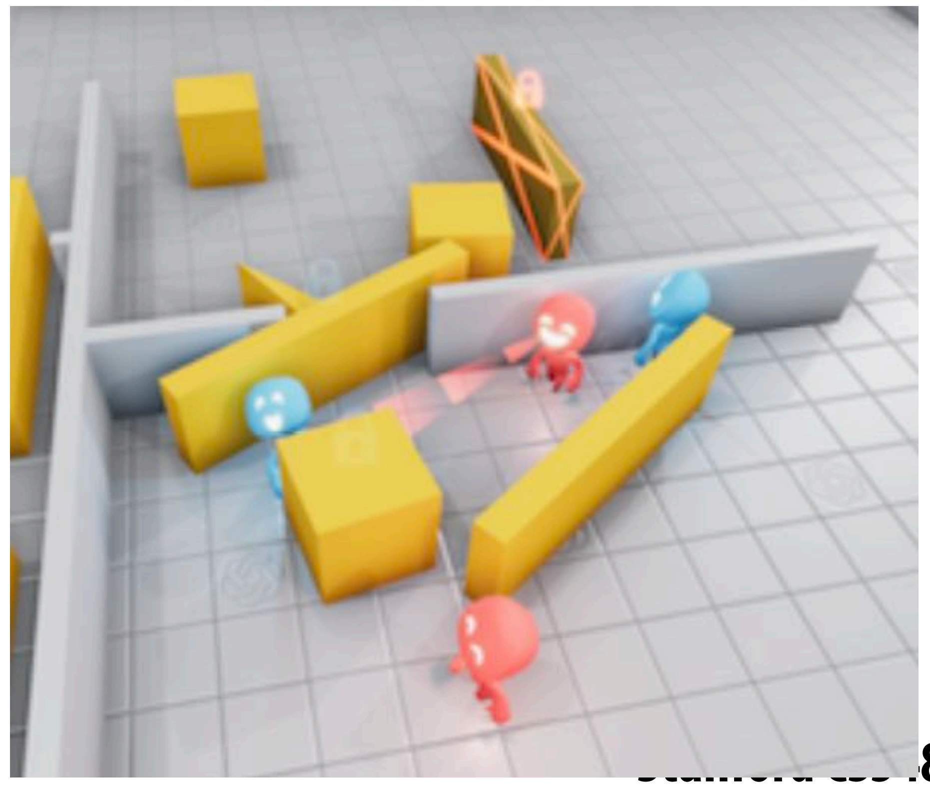












Design issues

- Inefficient simulation/rendering: rendering a small image does not make good use of a modern GPU (rendering throughput is low)

- Duplication of computation and memory footprint (for scene data) across renderer/ simulator instances

- Seems wasteful, right?

Tonight's reading

- **The design of a game engine for the specific case of running many independent world simulations at the same time on a GPU [Shacklett et al 2023]**