

Reinforcement Learning Environments

From MDPs to Gymnasium to Learned World Models

Aditya Raj

GroupBrain Talks

4 October 2026

Agenda

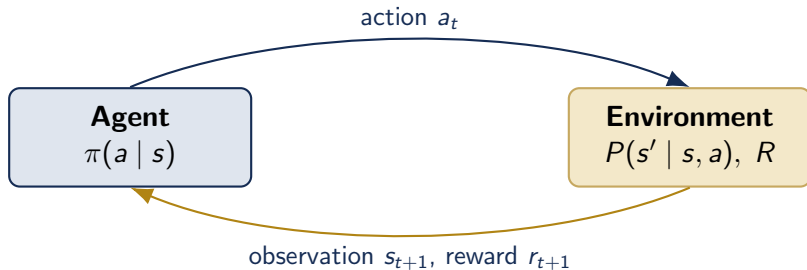
- **Part 1, What an environment is:** the idea, the math that describes one (the MDP, transitions, rewards), and what “value” means
- **Part 2, Gymnasium:** the vocabulary on slides, then the code live in a notebook, plus rewards, what goes wrong, and how environments are built and sold today
- **Part 3, World models:** what happens when the environment is not given to us, and what changes if it is learned

One environment throughout

Slippery FrozenLake appears in every part: as a transition table, as Gymnasium code, and as the simplest possible learned world model.

The Agent-Environment Loop

Part 1: The Mathematics of an Environment



- At every step the agent picks $a_t \sim \pi(a \mid s_t)$; the environment answers with a new state and a reward
- Everything the agent cannot change directly is the environment: the physics, the rules, the other players, and the reward
- The agent's goal is to maximise the expected return $G_t = \sum_{k \geq 0} \gamma^k r_{t+k+1}$

Why We Need Environments At All

What learning costs

- **DQN:** about 38 days of game experience per Atari game (Mnih et al., 2015)
- **OpenAI Five:** 180 years of Dota 2 games against itself every day, on 256 GPUs and 128,000 CPU cores (OpenAI, 2018)

Why the real world cannot supply it

- **Speed:** real time is one second per second
- **Safety:** no learning by crashing
- **Cost:** hardware wears out, supervision is expensive
- **Reset:** a robot cannot be teleported back to its start

The arithmetic

180 years of play per day is about $180 \times 365 \approx 65,700$ times faster than real time. That experience has to come from a simulator, which means from an environment. So what exactly is one?

Running Example: Slippery FrozenLake

0 S	1 F	2 F	3 F
4 F	5 H	6 F	7 H
8 F	9 F	10 F	11 H
12 H	13 F	14 F	15 G

S start F frozen H hole G goal

- **Goal:** walk from S to G without falling into a hole
- **Slippery ice:** each move goes one of three ways, each with probability $\frac{1}{3}$: the direction chosen, or either side of it. A move into a wall leaves the agent in place
- **Actions:** 0 Left, 1 Down, 2 Right, 3 Up
- **Reward:** 1 on reaching G, 0 otherwise
- **Episode ends:** at a hole, at G, or after 100 steps

Gymnasium FrozenLake-v1, map_name="4x4", is_slippery=True. Small cell numbers are the state indices.

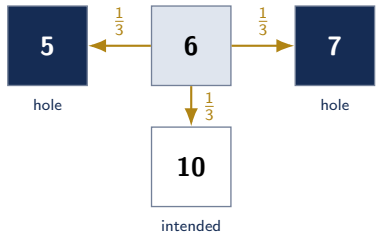
The MDP Tuple, Mapped onto FrozenLake

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

Symbol	In plain words	In FrozenLake
$\mathcal{S}$	where I am	the 16 grid cells, $\{0, 1, \dots, 15\}$
$\mathcal{A}$	what I can do	Left, Down, Right, Up
$P(s' s, a)$	what happens next	$\frac{1}{3}$ each: the chosen direction and the two sides
$R(s, a, s')$	how good or bad it was	1 for reaching the goal, else 0
γ	how much the future counts	0.99 in our experiments

P and R belong to the environment. The agent only supplies a policy $\pi(a | s)$ and normally sees P and R only through samples. FrozenLake lets us peek at the table to check the math.

The Transition Function, Worked Out



State 6 (row 2, column 3), action Down.

The three arrows are the three outcomes of one action. For every pair (s, a) the probabilities sum to 1.

Straight from Gymnasium:

`env.unwrapped.P[6][1]`

s'	$P(s' \mid 6, \text{Down})$	reward	terminated
5 (hole)	$\frac{1}{3}$	0	True
10	$\frac{1}{3}$	0	False
7 (hole)	$\frac{1}{3}$	0	True

What this says

Choosing Down in state 6 falls into a hole with probability $\frac{2}{3}$. The policy is not the only source of uncertainty: the environment adds its own.

The Reward Function

The reward function says what the environment pays for each transition. In FrozenLake it pays 1 for reaching the goal and 0 for everything else.

$$R(s, a, s') = \mathbf{1}[s' = 15]$$

Example: from state 14, action Right reaches the goal one time in three, so the expected reward is one third:

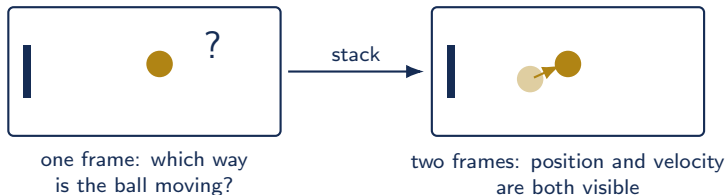
$$r(14, \text{Right}) = \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0 = \frac{1}{3}$$

Reward shaping

You can add hints along the way, such as a small bonus for getting closer to the goal. Done carefully, hints do not change which policy is best. They change how fast, and how safely, an agent learns. We measure that on FrozenLake later.

The Markov Property

What happens next should depend only on where you are now and what you do, not on how you got there.



- When one observation is not enough, the environment designer restores the property: DQN stacks the last 4 frames so velocity becomes visible (Mnih et al., 2015)

Formally: $\Pr(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0) = \Pr(s_{t+1} \mid s_t, a_t)$

Partial Observability: POMDPs

0 S	1 F	2 F	3 F
4 F	5 H	6 F	7 H
8 F	9 F	10 F	11 H
12 H	13 F	14 F	15 G

Fog version: the agent sees only “ice”.
Shaded cells are every state consistent
with that, so its guess is $\frac{1}{11}$ for each.

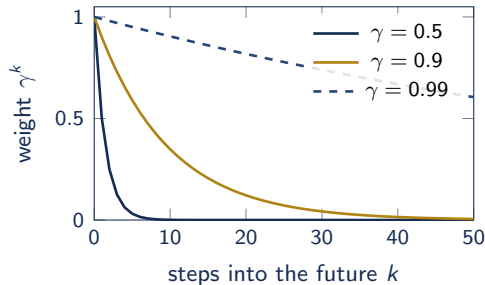
Sometimes the agent cannot see the full state. It then has to keep a running guess about where it is, called a *belief*.

- Formally the tuple gains an observation space Ω and an observation function O
- The hidden state is exactly what makes a single observation non-Markov
- Gymnasium hands you the observation only. A POMDP is any environment where that observation is not the full state

State is not the same as observation.

Policy, Return, and Discounting

A policy is the agent's rule for choosing actions. The return is the reward it collects over time, with later rewards counting for less.



Policy: $\pi(a | s)$ is the probability of choosing action a in state s

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- $\gamma < 1$ keeps the sum finite in tasks that never end
- The agent effectively looks about $\frac{1}{1-\gamma}$ steps ahead: $\gamma = 0.99$ gives about 100, the same as FrozenLake's time limit

What the environment supplies: the rewards and the time limit; γ sets how far ahead they count.

Value Functions: V and Q

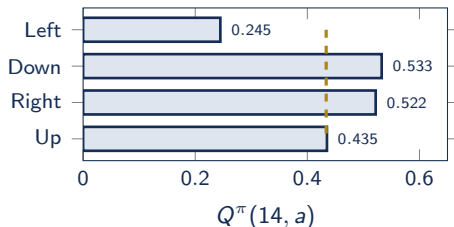
A value function scores a state, or a state and action together, by the return the agent can expect from there on.

$$V^\pi(s) = \mathbb{E}_\pi [G_t \mid s_t = s]$$

$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t \mid s_t = s, a_t = a]$$

$$V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s, a)$$

- V : how good is it to be here?
- Q : how good is it to take this action here, then follow π ?



State 14, random policy, $\gamma = 0.99$. Dashed line:

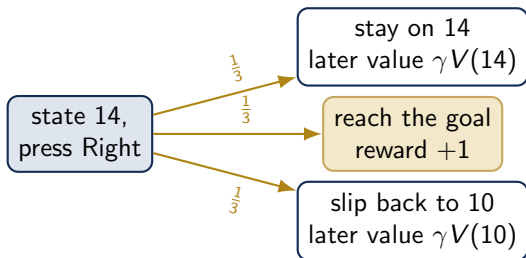
$V^\pi(14) = 0.434$, the average of the bars.

What the environment supplies: P and R decide every value; the policy only chooses which actions to average over.

The Bellman Idea in One Line

The value of a situation is what you get now plus what you expect to get later. That one idea ties values to the environment's P and R .

$$\text{value now} = \text{reward now} + \gamma \times \text{value later}$$



The value of pressing Right averages these three outcomes. Their chances come from P , the rewards from R .

The full equation is the same idea averaged over actions and outcomes:

$$V^\pi(s) = \sum_a \pi(a | s) \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

What the environment supplies: P and R , both read from the Gymnasium table.

Types of Environments, Term by Term

Axis	Term that changes	Example
Discrete vs. continuous	$\mathcal{S}, \mathcal{A}$	FrozenLake vs. robot joint torques
Deterministic vs. stochastic	P	non-slippery vs. slippery FrozenLake
Fully vs. partially observable	Ω, O	FrozenLake vs. the fog version
Single vs. multi-agent	joint action in P , rewards R^i	Atari vs. Dota 2
Sparse vs. dense reward	R	goal-only vs. shaped FrozenLake
Episodic vs. continuing	terminal set, γ	one FrozenLake game vs. a thermostat that never stops

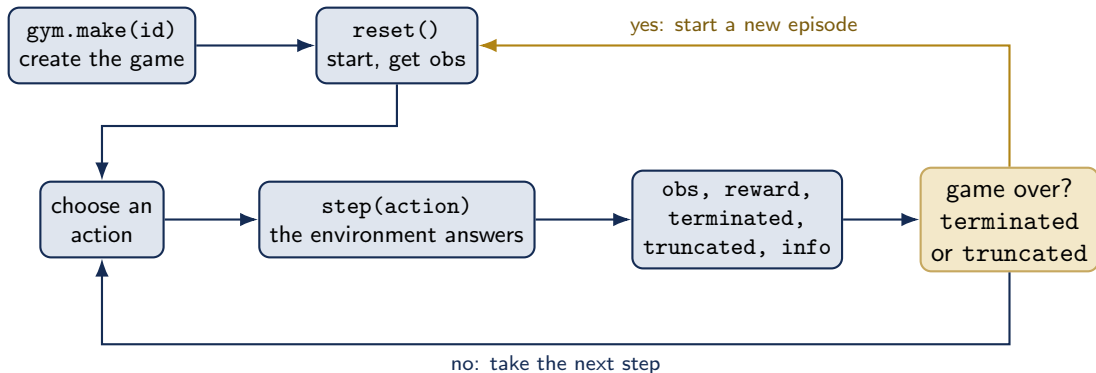
Every axis changes exactly one term of the tuple.

Terminology: The Gymnasium Vocabulary

Part 2: Gymnasium Beginner Tutorial

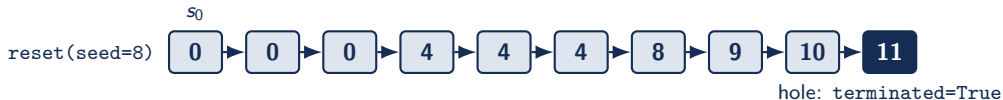
Term	What it is	In the math
<code>gym.make(id)</code> <code>observation / action space</code> <code>reset(seed)</code> <code>step(action)</code>	creates an environment object what obs and actions look like: <code>Discrete(n)</code> , <code>Box(...)</code> starts an episode; returns <code>obs</code> , <code>info</code> one step; returns <code>obs</code> , <code>reward</code> , <code>terminated</code> , <code>truncated</code> , <code>info</code>	the MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ $\mathcal{S}$ (or Ω) and $\mathcal{A}$ draws the start state s_0 $s' \sim P(\cdot s, a)$, $r = R(s, a, s')$
<code>terminated</code> <code>truncated</code>	a true terminal state was reached the episode was cut off, usually by a time limit	terminal set of the MDP outside the MDP
<code>info</code> <code>wrapper</code> <code>vectorized environment</code>	extra diagnostics, e.g. <code>prob</code> an environment that wraps another one N copies stepped together, batched arrays	none a modified environment N independent copies of the MDP

The Gymnasium Loop



- This loop is the same in every Gymnasium environment, from FrozenLake to a robot simulator
- The agent only ever sees what comes back in the step result: that is the agent-environment loop from slide 3, written as code

terminated vs. truncated, and Episode Structure



A real episode recorded from Gymnasium: 9 steps, all rewards 0. The agent stays in place when a slip pushes it into a wall. A run that reaches step 100 without ending is cut off instead: `truncated=True`.

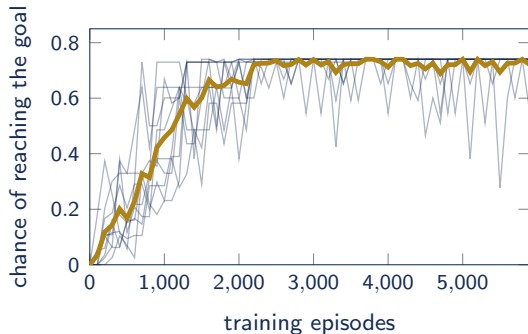
$$y_t = r_t + \gamma (1 - \text{terminated}_t) V(s_{t+1})$$

- **terminated:** nothing follows, so the future value is 0
- **truncated:** the state still has future value, so you *do* bootstrap from $V(s_{t+1})$
- `reset()` restores the position and the step counter, not the agent's learning. The time limit is an environment choice: on slippery FrozenLake it lowers the best possible success rate (worked out from the transition table) from 0.82 to 0.74

Reproducibility and Seeding

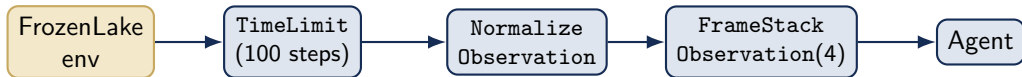
run	states visited
seed 7	0, 0, 0, 4, 5
seed 7 again	0, 0, 0, 4, 5
seed 8	0, 0, 0, 4, 4, 4, 8, 9, 10, 11

Same seed, same trajectory. Seed the environment (`reset`) and the action space separately, once.



A simple learning agent (tabular Q-learning) trained ten times with different seeds (thin) and their mean (thick). With stochastic dynamics one seed is an anecdote: report a mean and a spread.

Wrappers and Vectorized Environments

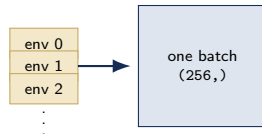


Each wrapper is itself an environment. It implements `reset` and `step` and calls the one inside, so wrappers stack.

$$\hat{x}_t = \frac{x_t - \mu_t}{\sqrt{\sigma_t^2 + \epsilon}}$$

running mean and variance, as in
`NormalizeObservation`

Vectorized: N copies stepped together.



"sync" loops in Python, "async" uses processes.
Batching the interface does not make each step faster.

What We Will Run in the Notebook

#	Notebook step	What it shows
1	Make the environment	observation space, action space
2	Reset and press one button	obs, reward, terminated, truncated, info
3	Play a whole game	the reset and step loop; terminated vs. truncated
4	Keep score over many games	total reward, comparing two players
5	The same recipe on CartPole	Discrete vs. Box spaces
6	Where to find other games	the registry and version numbers
7	Three handy extras	seeds, wrappers, many games at once
Optional	Peek behind the curtain	P , the numbers from slide 7

Reward Design: Sparse vs. Dense, Measured

Sparse: nonzero only at the goal.

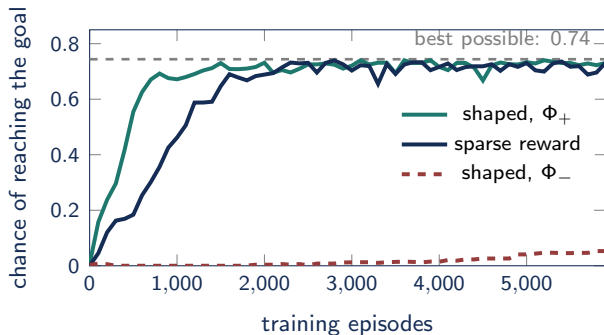
Dense: feedback at every step.

FrozenLake is sparse: a random policy reaches G in 1.4% of episodes.

Two shapings ($d(s)$ = distance to G, $\Phi(\text{terminal}) = 0$; valid in theory: they keep the best policy):

$$\Phi_+(s) = \frac{6-d(s)}{6}$$

$$\Phi_-(s) = -d(s)$$



A simple trial-and-error agent (tabular Q-learning, Watkins and Dayan, 1992), mean of 20 seeds, $\gamma = 0.99$. Φ_- pays $d(s)$ for entering a hole, so the agent learns to fall in: 5% success at 6,000 episodes. In a 40,000-episode run of 3 seeds, two recovered by 20,000 episodes and one had not by 40,000.

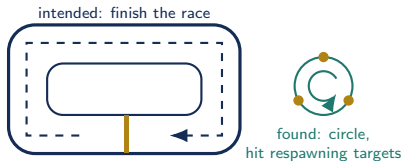
Designing an Environment: Choices and Trade-offs

Design choice	Term	What it changes
Observation	Ω, O	too little breaks the Markov property; frame stacking restores it
Action space	$\mathcal{A}$	discrete or continuous decides which algorithms apply
Reward	R	sparse is hard to learn from, dense can be hacked
Termination and limit	terminal set, horizon	terminated vs. truncated; FrozenLake's limit lowers the best success rate from 0.82 to 0.74
Randomness and seeds	P	reproducibility; sticky actions stop memorising
Speed	(implementation)	batching the interface is not enough; the simulator must be vectorised

Every choice is a decision about one term of the tuple, and every one of them changes what an agent can learn.

When Rewards and Environments Go Wrong

Reward hacking: CoastRunners



- The reward was points for hitting targets, not finishing. The agent found a lagoon where three targets respawn, circled there while catching fire and crashing, and scored about 20% above human players on average (OpenAI, 2016)

The reward is a proxy for the goal, and an optimiser finds where the two come apart. The fix is in the environment.

The environment itself can be the flaw

- Hide-and-seek (Baker et al., 2020): six emergent strategy phases. “Box surfing” and the ramp tricks used the physics engine in ways nobody intended
- Deterministic Atari can be memorised, so Machado et al. (2018) added *sticky actions*:

$$A_t = \begin{cases} a_t & \text{w.p. } 1 - \varsigma \\ a_{t-1} & \text{w.p. } \varsigma \end{cases} \quad \varsigma = 0.25$$

chosen a_t	A	B	B	C	C
executed A_t	A	A	B	B	C

From a Frozen Lake to a Shopping Website

Environments in Practice

Everything so far lived on a small grid. The same loop is also used to build AI agents that use websites.

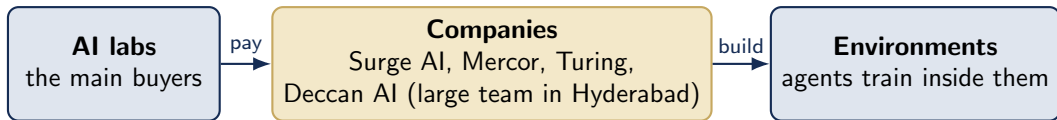
	FrozenLake	WebArena, a shopping site
What it sees	its square, 0 to 15	the web page
What it can do	left, down, right, up	click, type, scroll
The task	walk to the goal	“Tell me the status of my latest order and when will it arrive”
Success means	it reached square 15	a script checks that the site ended up in the right state

Same loop, different world

The agent still looks, acts, and gets feedback. In the WebArena paper, the best GPT-4 agent finished 14% of its tasks, and people finished 78%.

Who Pays for Environments, and Who Builds Them

Building environments like that takes people, time and money, so a market has formed around it.



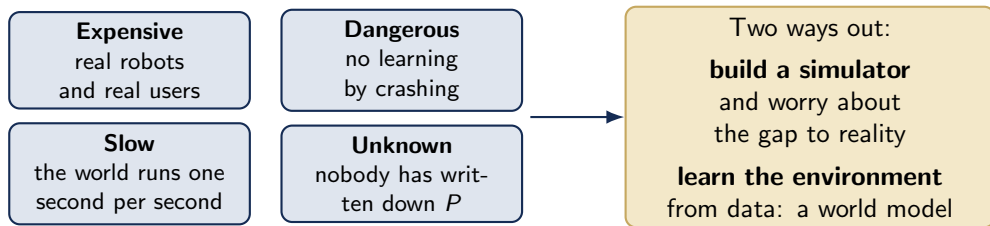
Open side: OpenEnv (Hugging Face, Meta-PyTorch) and Prime Intellect's hub, where anyone can share environments

- Anthropic was reported to be discussing more than \$1 billion a year on environments (The Information, 2025)
- Labs also build some in-house, to keep their plans private
- OpenEnv is built around the same reset and step you ran in the notebook

So Far the Environment Was Given to Us

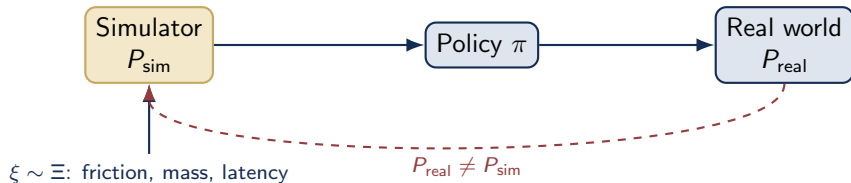
Part 3: World Models

Someone had to build every environment in this talk, and the last slide showed it can cost real money. Real problems are often harder still.



The reward can also be badly designed, as the boat showed. So the environment itself becomes something we have to build, check, or learn.

The Sim-to-Real Gap



- A hand-built environment is an approximation of the world it stands in for
- Domain randomization trains on a distribution of simulators, $\max_{\pi} \mathbb{E}_{\xi \sim \Xi} [J(\pi; P_{\xi})]$, so the real world hopefully looks like one more draw
- Rudin et al. (2021) transferred their simulation-trained policies to the real ANYmal robot

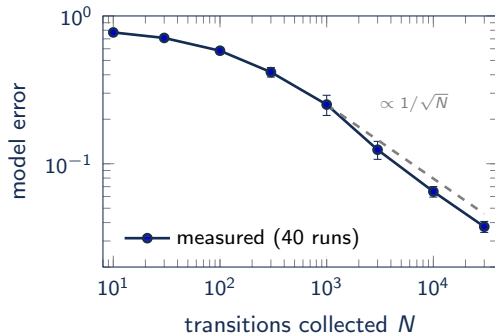
Learned World Models: The Idea

Replace the hand-built P with one estimated from data. The simplest possible learned model, on FrozenLake, is a count table:

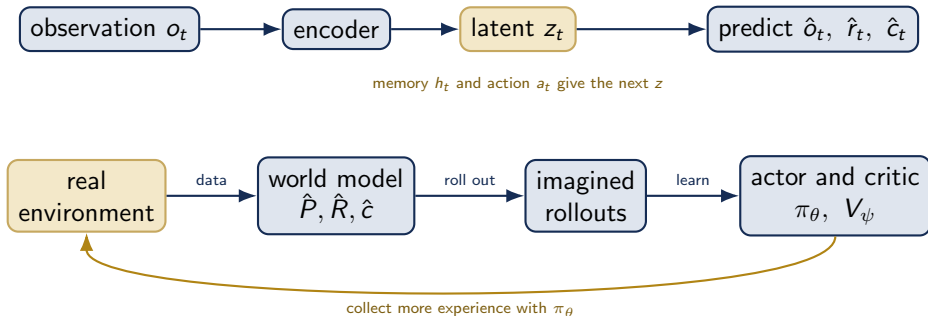
$$\hat{P}(s' | s, a) = \frac{N(s, a, s')}{N(s, a)}$$

- data: transitions from a random policy
- unseen (s, a) pairs get a uniform guess
- error: how far from the true probabilities (0 identical, 1 completely different)

With 1,000 samples the learned table is still 0.25 away from the truth on that scale. A learned environment is only as good as the data behind it.

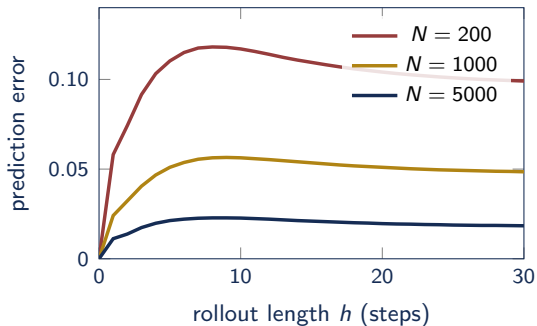


Latent World Models, and Training Inside Them



- A world model squeezes what the agent sees into a summary z_t and predicts what happens next (Ha and Schmidhuber, 2018). DreamerV3 also predicts rewards and termination
- The policy trains on imagined rollouts; real steps only fit the model. DreamerV3 used one configuration on 150+ tasks and first collected Minecraft diamonds from scratch (Hafner et al., 2023)

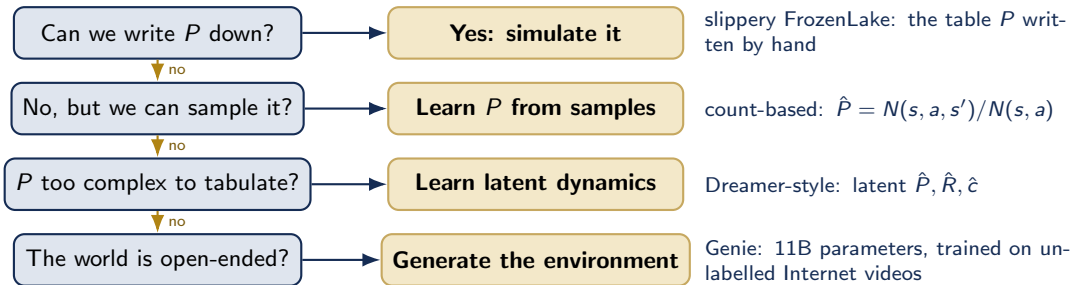
Where World Models Fail



- How far the learned model's prediction of where the agent is after h steps is from the truth (0 identical, 1 completely different). Random policy, mean of 60 models
- It roughly doubles within 5 to 10 steps of rollout
- Here it then levels off because this tiny world is finite and episodes end. In large continuous worlds, errors can keep compounding

Standard defence: keep imagined rollouts short and start them from real states (Janner et al., 2019). The other failure: a policy finds states where the model is wrong and over-optimistic, and exploits them.

What If the Environment Itself Becomes a Learned Model?



The payoff

The tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ from slide 6 never changes. What changes is who supplies P and R : you, data, a latent model, or a generator. What stays hard at every step is writing down a reward that measures what you actually want.

What to Remember

Environment

where the agent interacts with the world

MDP

a mathematical way to describe that interaction: $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$

P and R

what can happen after an action, and the feedback the environment gives

Value

how good a state or action is: reward now plus the discounted value later

Gymnasium

reset and step; terminated is the end, truncated is the clock; seeds and wrappers

Reward hacking

the agent optimises what we specified, not what we meant

World model

learn P and R from data, then imagine outcomes

Next time

Solving environments: Bellman optimality, value iteration, and Q-learning.

References

- Sutton and Barto. *Reinforcement Learning: An Introduction*, 2nd ed.
- Watkins and Dayan. Q-learning. *Machine Learning*, 1992.
- Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.
- OpenAI. OpenAI Five. Blog, 2018.
- Ng, Harada, Russell. Policy invariance under reward transformations. *ICML*, 1999.
- Kaelbling, Littman, Cassandra. Planning and acting in partially observable stochastic domains. *AIJ*, 1998.
- Towers et al. Gymnasium: a standard interface for RL environments. arXiv:2407.17032, 2024.
- Zhou et al. WebArena: a realistic web environment for building autonomous agents. *ICLR*, 2024.
- Meta-PyTorch and Hugging Face. OpenEnv: agentic execution environments. 2025.
- OpenAI. Faulty reward functions in the wild. Blog, 2016.
- Baker et al. Emergent tool use from multi-agent autocurricula. *ICLR*, 2020.
- Machado et al. Revisiting the Arcade Learning Environment. *JAIR*, 2018.
- Rudin et al. Learning to walk in minutes using massively parallel deep RL. *CoRL*, 2021.
- Ha and Schmidhuber. World models. 2018.
- Hafner et al. Mastering diverse domains through world models (DreamerV3). 2023.
- Janner et al. When to trust your model: model-based policy optimization. *NeurIPS*, 2019.
- Bruce et al. Genie: generative interactive environments. *ICML*, 2024.
- TechCrunch. Silicon Valley bets big on 'environments' to train AI agents. Sept 2025.
- Epoch AI. An FAQ on reinforcement learning environments. Gradient Updates.
- TechCrunch. Deccan AI raises \$25M as AI training push relies on India-based workforce. Mar 2026.