A FREE INTERACTIVE PRIMER

World Models 101

How machines learn to predict what happens next, in nine chapters: what the phrase means, how the machinery works, and what is still broken.

BY NILUSHANAN KULASINGHAM

WORLDMODELS101.COM · 2026 · CC BY-SA 4.0

About the author

Nilushanan Kulasingham is a software engineer and founder who has spent a decade building interactive entertainment products. In 2014 he founded Paravine, an esports news and statistics platform that grew to 800,000 monthly readers and was acquired by CBS Interactive as the foundation of its esports vertical, onGamers. He later founded and led Nucanon, an AI gaming company whose tools generated canon-consistent story, lore and worlds for game studios, backed by Antler, Skalata, Futureverse and Jason Calacanis and acquired by the Indian gaming platform Zupee.

In between he launched SBS News's first live blog platform; helped rebuild the Qantas Rewards Store; led the frontend at Gamurs, the publisher of Dot Esports; and rebuilt Futureverse's core blockchain in Rust.

World Models 101 is his attempt to lay the field out plainly, in the order he wishes he had met it. The book is free and stays free. The interactive version lives at worldmodels101.com, where every figure in these pages can be pressed and dragged. The source, including every figure, is at github.com/NiluK/worldmodels101, under CC BY-SA 4.0: reuse it, with credit, and share what you make under the same terms. Corrections are welcome and credited; the About page on the site says how to send one.

```
https://www.linkedin.com/in/nilukulasingham/
https://github.com/NiluK/worldmodels101
```

01 What Is a World Model?

BY NILUSHANAN KULASINGHAM

28 MIN READ · INTERACTIVE

> The phrase has covered at least five different machines since 2018, and the people using it rarely say which. A field guide to the five, where each came from, and the one test that separates them.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/what-people-mean.

If you've spent any time on your timeline lately, you'll probably have seen a clip like the one below. Somebody is walking through a landscape with the arrow keys, and the caption says it was generated rather than built. It looks like a video game that nobody made. The replies call it a world model, so you go and look the phrase up.

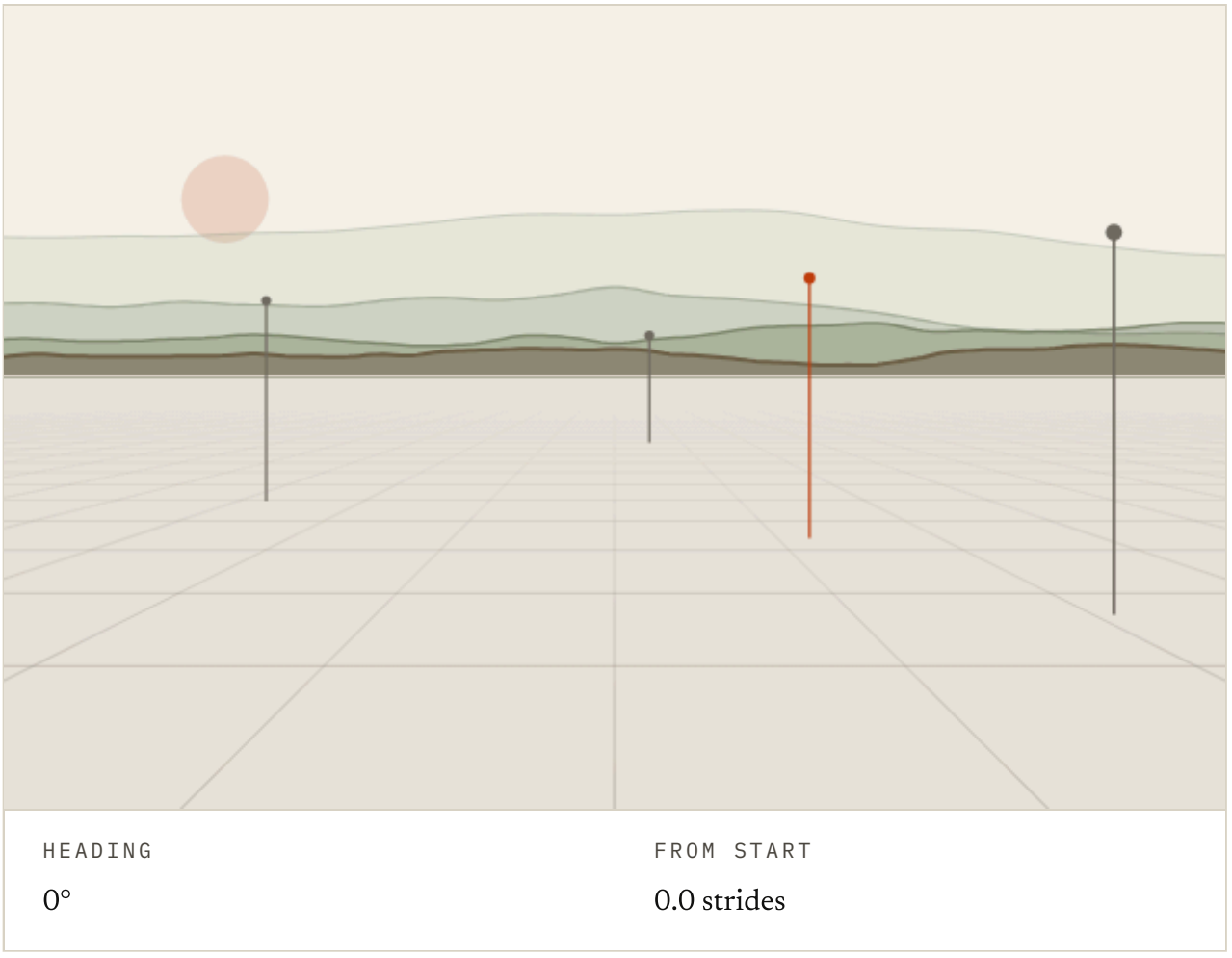


FIG. 1.1 Nothing in this picture is a recording. Every frame is drawn from where you stand and which way you face, and nothing behind it stores a landscape. Drag to look around, or click it and walk with the arrow keys.

Ten minutes later you have five tabs open, and they describe five different machines. One generates video that you steer with the arrow keys. One exports geometry, meaning 3D shapes and where they sit, into Blender, the 3D modelling program. One is a paper about predicting embeddings, lists of numbers that stand in for pictures, and it never shows you a video at all.

I should say the phrase confused me for a long time too. The explanations online, while helpful, tended to pick one of the five machines and explain it as if it were the whole story. What I wanted was one page that laid all five out side by side and said how they came to share a name. This chapter is that page, and if you are confused like I was, hopefully it helps.

The mess has a history, and the history is a lot tidier than the tabs. Between about 2018 and 2024, four traditions arrived at the same phrase from four directions. They were reinforcement learning and control, self-supervised representation learning (teaching networks to summarise images and video), generative video and spatial models, and interpretability, the study of what goes on inside a trained model. Each meant something real, and none of them had to check what the others meant.

“world model” 5 INCOMPATIBLE ANSWERS

A DeepMind demo where somebody walks through a generated landscape with the arrow keys.

Genie 3 · Google DeepMind ↗

TURNS OUT TO BE

The Renderer
https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/

A Meta paper about predicting video embeddings that never shows you a video.

V-JEPA 2 · Meta AI ↗

TURNS OUT TO BE

The Representation
https://ai.meta.com/blog/v-jepa-2-world-model-benchmarks/

A 3D environment you can load straight into Blender.

Marble · World Labs ↗

TURNS OUT TO BE

The Simulator
https://www.worldlabs.ai/blog/marble-world-model

A reinforcement learning result from 2018 about a car in a racing game.

World Models · David Ha & Jürgen Schmidhuber ↗

TURNS OUT TO BE

The Dynamics Model
https://worldmodels.github.io/

An argument, conducted mostly at volume, about whether a language model that has never seen a chessboard has one inside it anyway.

Emergent World Representations · Kenneth Li et al. ↗

TURNS OUT TO BE

```
The Implicit Model
https://arxiv.org/abs/2210.13382
```

FIG. 1.2 Have a look at these five results, from five different classes of system. Every one of them was called a world model by the people who built it.

By 2026 the labs had started publishing dictionaries. Five definitions are in current use, each out of its own tradition and its own year. There is a four-second test that ought to tell them apart, and as we'll see, it does not.

The five definitions

Let's go through the five first. Treat these categories as contracts rather than species. Each one describes what a system promises to hand you. Modern work crosses between them all the time, as NVIDIA's Cosmos and Meta's V-JEPA 2 are about to show.

FIG. 1.3 Four of these are systems you can run. They are ordered by how abstract the predicted object is. The fifth is a claim about another system rather than a system itself, which is why it sits off the axis.

The Renderer predicts observations, usually pixels. GameNGen, from Dani Valevski and colleagues at Google (2024), draws frames of the video game DOOM. Each frame comes from the frames before it and the player's inputs. It runs at about twenty frames a second on a single TPU chip, and that is fast enough to play.

Genie 3, from DeepMind, is reported to generate interactive video at 720p and 24 frames per second. It is said to stay coherent for several minutes, with the scene hanging together as you move. It also takes plain-language instructions mid-session: change the weather, add a flock of birds.

Sora belongs in this category too, but without the action input: it predicts observations you cannot steer. Keep that difference in mind, because whether actions change the prediction is one of the three questions at the end of this chapter.

So persistence exists, meaning things stay where you left them. It is learned through generation, though, rather than guaranteed by stored geometry.

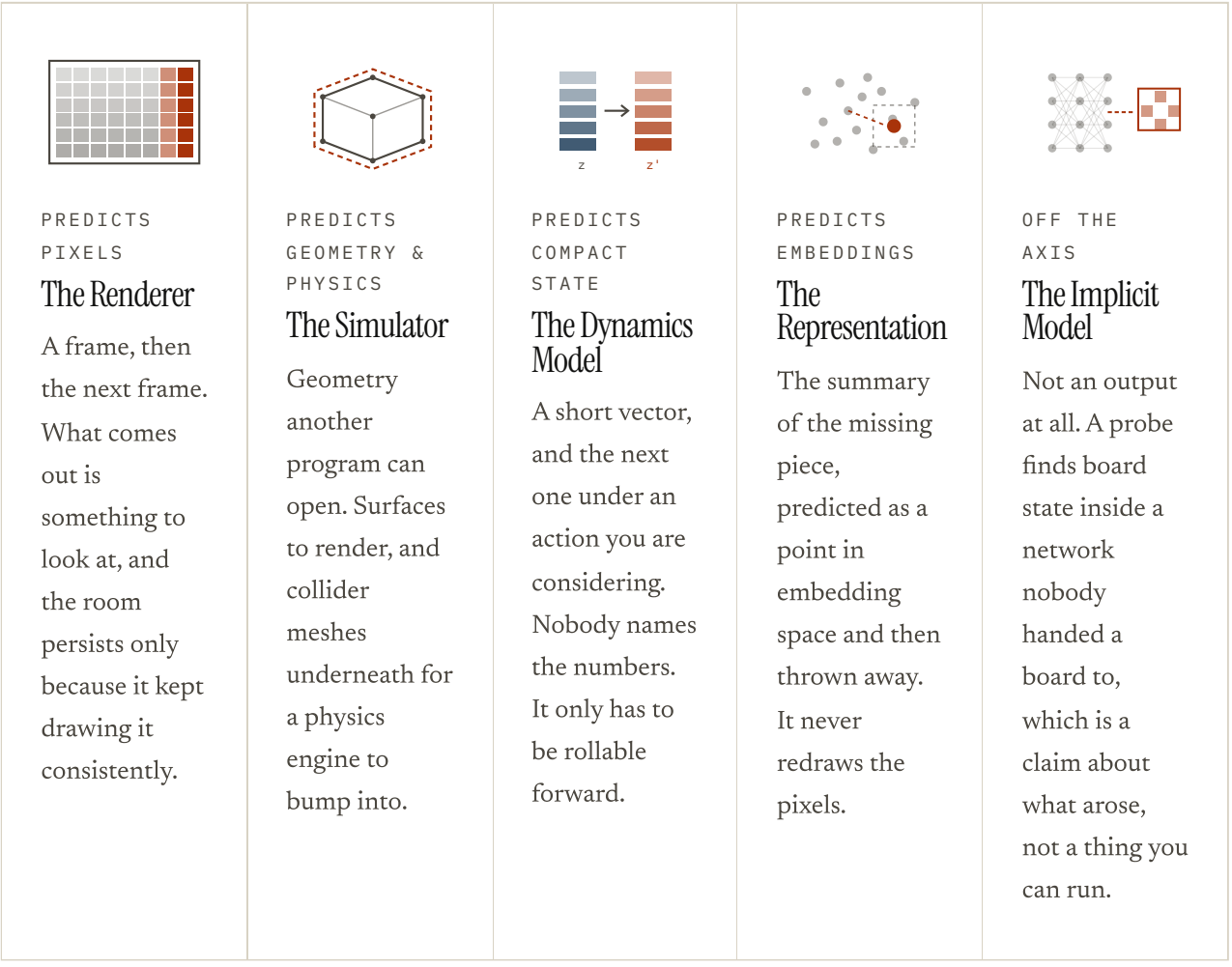


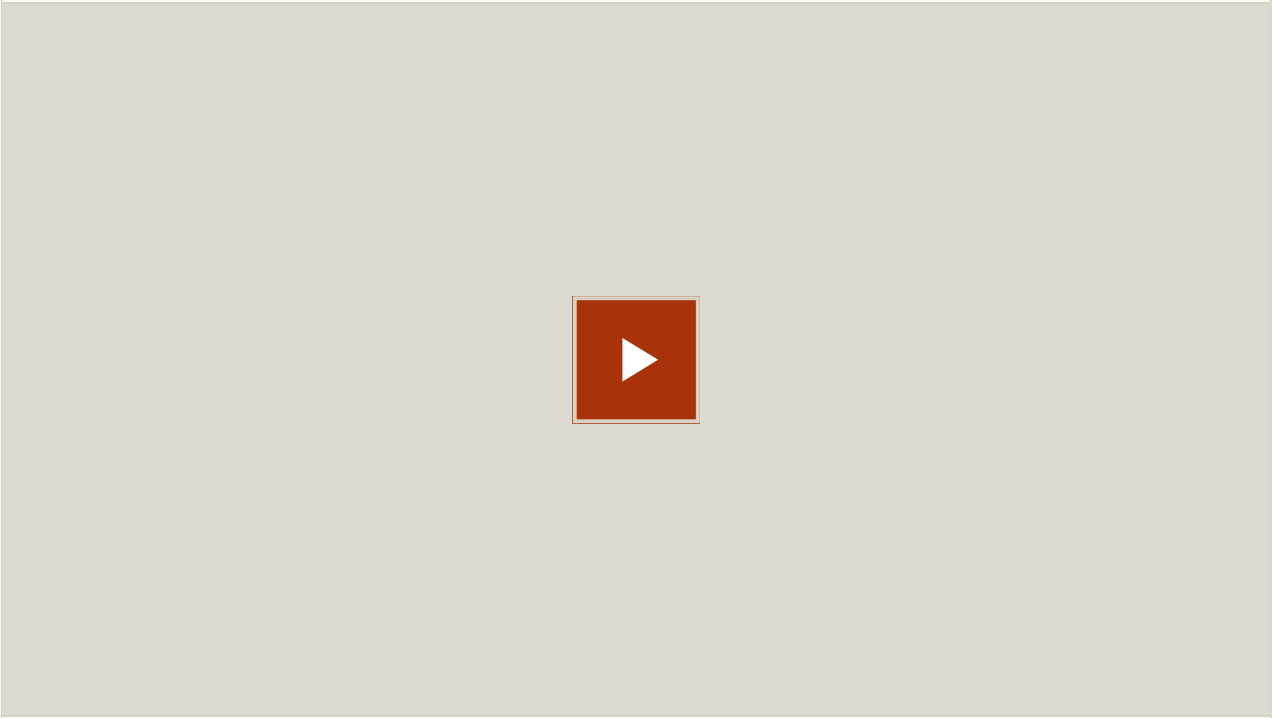
FIG. 1.4 The same five, this time drawn by what each one hands you. Vermilion, the orange-red, marks the predicted object in every panel. Left to right is the same axis as the map above: a picture, then structure, then a compact state, then a summary, and finally something that is not an output at all.

The Simulator predicts structure you can query, meaning 3D shapes you can ask questions of. Marble comes from World Labs, a start-up co-founded by the Stanford researcher behind ImageNet. That is the picture dataset that set off the deep learning boom. Marble takes text, an image or a rough 3D layout, and hands back two things at once.

Gaussian splats (millions of coloured blobs, cheap to render, nothing to bump into) carry the look. Triangle meshes carry the structure. A mesh is just a surface built from many small triangles. That is ordinary 3D geometry, collider meshes included (the invisible shapes a physics engine bumps into).

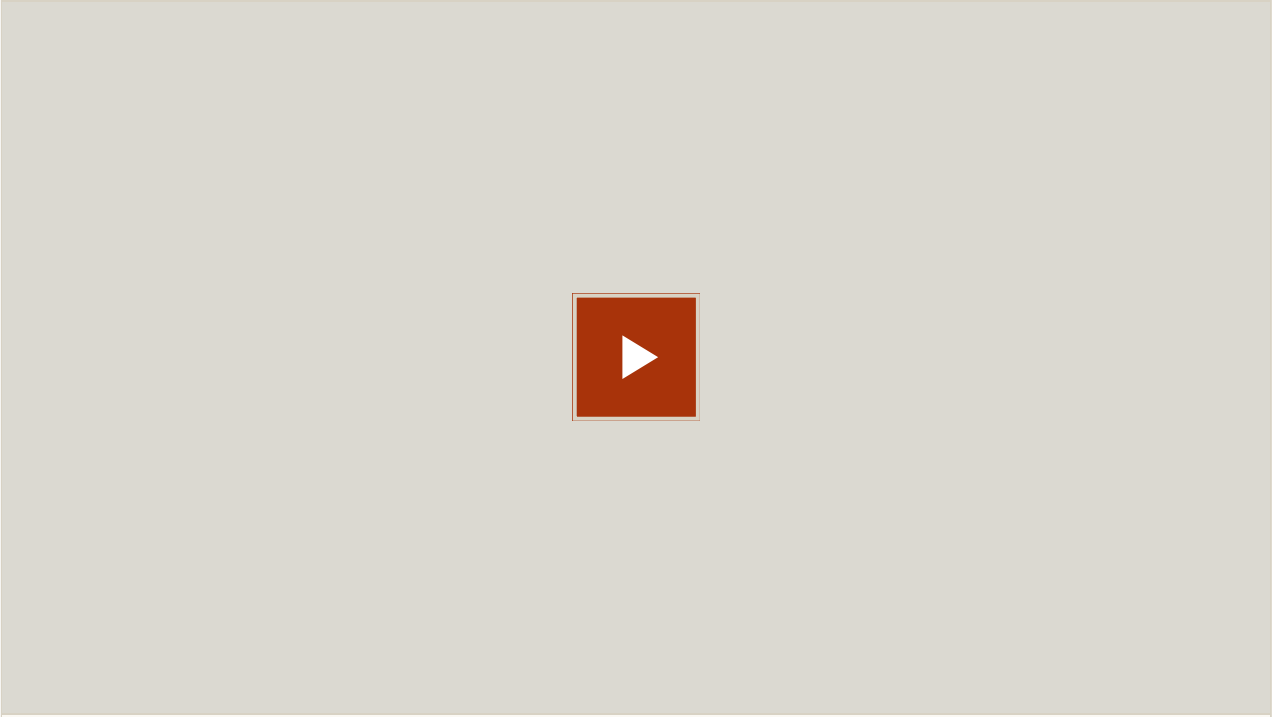
World Labs shipped Marble to a limited beta in November 2025 and opened it to everyone in February 2026. NVIDIA open-sourced Cosmos in January 2025. It generates physically-aware video, built as training data for robots and self-driving cars.

RENDERER / PREDICTS OBSERVATIONS



GOOGLE DEEPMIND Genie 3: Creating dynamic worlds that you can navigate in real-time

SIMULATOR / PREDICTS STRUCTURE



WORLD LABS Introducing Marble by World Labs

FIG. 1.5 I'd read this as a comparison of promises rather than a pass/fail test. Marble hands you geometry that another program can open, so you can check it yourself. Genie 3's coherence is reported by the lab that built it, and you cannot check that.

NVIDIA itself calls Cosmos Predict, the part that makes the video, generative video. The real 3D simulator is Omniverse, and that is a different NVIDIA product. If you put the whole thing under "simulator" you have hidden that split from yourself.

A platform is not a category. One product can expose a pixel predictor, a 3D simulator and an action-conditioned model all at once. Action-conditioned just means it takes your actions as input. Ask which one you are being sold, and which interface you are about to build against.

The Dynamics Model predicts compact state under actions, often with the reward. Compact means a short list of numbers rather than a picture. It is the oldest of the four runnable categories. It is the one the line from Schmidhuber to Dreamer, which we will walk through in the history below, was building toward.

MuZero, from Julian Schrittwieser and colleagues at DeepMind, is the proof. Its model predicted only reward, value and likely moves, never the board. With that it matched AlphaZero, its famous predecessor, at Go, chess and shogi. It was never told the rules.

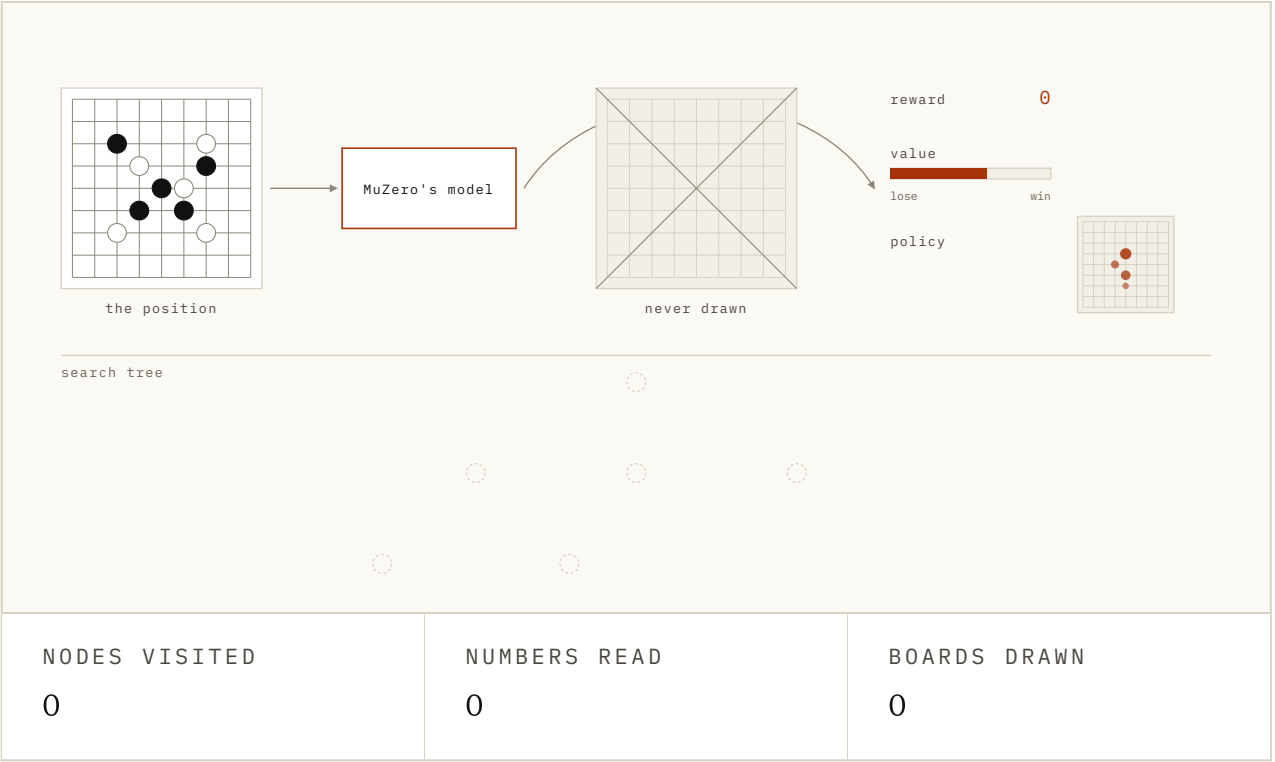


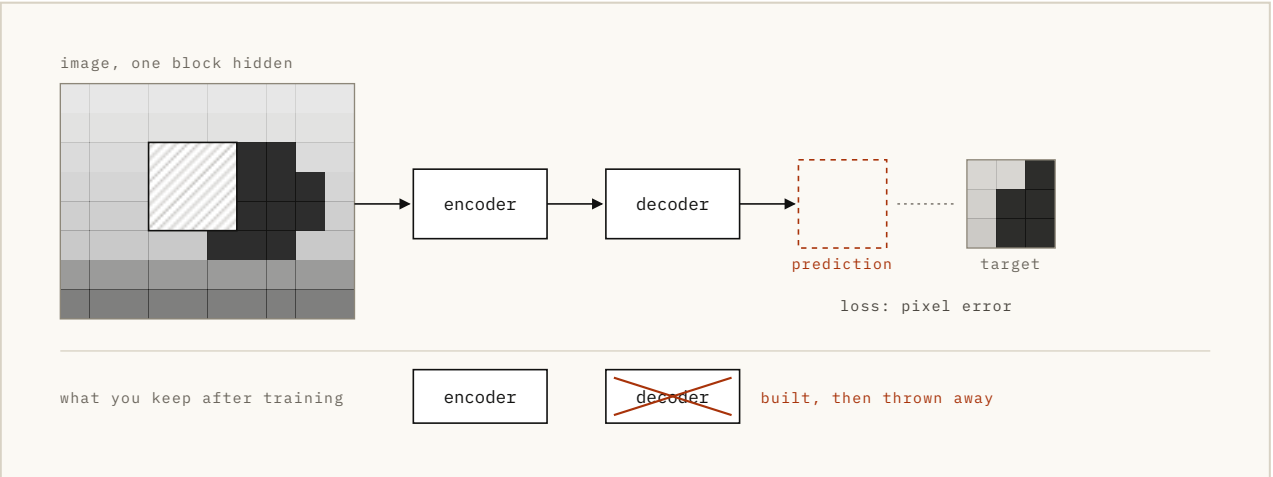
FIG. 1.6 MuZero's model sits in the middle. Leave the switch on three numbers and press Search to step a small tree: every node asks for reward, value and policy, and the board on the right is never drawn. Now flip the switch to the next board and watch each node pay for a picture that the search then reads nothing from. The values are illustrative.

Two systems you will meet properly in the history, PlaNet and Dreamer, use it in different ways. PlaNet plans inside a model like this, searching afresh at every step. Dreamer learns behaviour from futures imagined through one. The category is defined by whether you can search inside the model, not by how real its predictions look.

The Representation predicts embeddings (a list of numbers standing in for an image or a clip, with similar things close together). Then it throws the prediction away. I-JEPA (2023), the image version from Mahmoud Assran and colleagues, hides part of an image and predicts the embedding of the missing piece. It never redraws the pixels.

V-JEPA 2 (2025), the video version, pre-trains on more than a million hours of video with no actions involved. A second stage then trains an action-conditioned variant on top. That one is built for model-predictive control (plan a few steps, take one, plan

again). Meta reports it driving a Franka robot arm in a lab it never saw during training.



TARGET	LOSS	DECODER
the missing pixels	pixel error	built, then thrown away

FIG. 1.7 Choose what the network is asked to predict for the hidden block, then press Predict. With pixels as the target, a decoder has to turn the prediction back into a picture, and the best it can do for a block it cannot see is a blur. With a summary as the target, nothing is turned back into pixels. Look at the bottom row, which is what survives training: the encoder in both cases, and only one of them ever built a decoder.

Meta also reports it planning around 30× faster than Cosmos. Be careful with that number: it compares two different contracts rather than ranking them.

Notice that the second stage takes actions, rolls forward and gets searched over. That is the Dynamics Model's contract, reached from the other side. I call it migration: a system starts in the category it was trained for and grows the machinery of another on top. The categories describe a training target rather than a fixed identity.

The Implicit Model is a different kind of claim altogether. It is the least demoed of the five and the most argued about, because there is nothing to run: the claim is about the inside of a network.

Othello-GPT is a small language model trained only to predict legal moves in Othello, a board game played with two-sided discs. It was never shown a board. Kenneth Li and colleagues (2022), who trained it, reported in *Emergent World Representations* that the board was there inside it anyway.

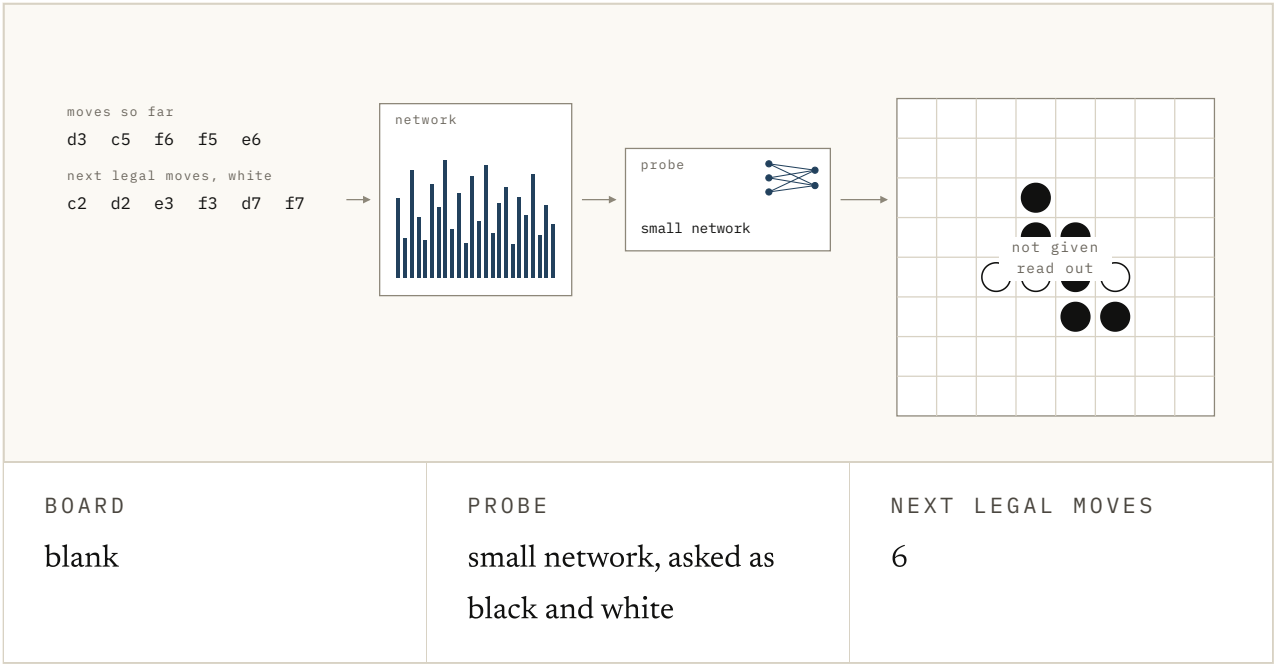


FIG. 1.8 Othello-GPT only ever saw move tokens like the strip on the left. Press Read the board and a probe draws an 8 by 8 board out of the network's activations, and nobody put one in. Flip a square in that inner board and watch the list of next legal moves change, which is how Li and colleagues showed the model was playing from it. Then switch to mine and theirs, which is Nanda's reframing, and the probe can be a straight line.

A probe (a small classifier trained to read one specific fact out of a network's activations) could read it out, and when they changed that inner board by hand, the model's next moves changed too. Neel Nanda and colleagues (2023) followed up in *Othello-GPT has a linear emergent world representation*: the board was there plainly enough for the simplest probe to read, once you asked for "mine" and "theirs" instead of black and white.

Wes Gurnee and Max Tegmark widened the claim in 2023 with *Language Models Represent Space and Time*. Probes on a large language model recovered something close to latitude and longitude for places, and dates for events. Nobody handed

either model a function called `world_model`. This is where the argument over the term gets loudest, because a claim like that is a finding about what grew inside another system rather than an interface you can run.

> Jim Fan of NVIDIA pushed back on the space and time paper at the time: a probe that recovers coordinates is doing classification, and you cannot make an intervention against it or close a decision loop around it. The Othello work is the stronger case, because flipping a square in the recovered board changed what the model played next. The question to ask of any of these results is which of the two you are being shown.

One equation, sixty years

Now for where the term came from, because it started somewhere specific. Everything after it either grew from that start or pushed against it. The start was control theory, the engineering maths of keeping a machine on course: an autopilot, a thermostat, a rocket.

Later it moved into reinforcement learning (training an agent by letting it act, watching what happens, and rewarding what worked), where a world model is a learned transition function, a rule for what comes next that the agent picks up from experience. A state and an action go in, and the next state comes out. When I say state, I mean a description of how things stand at one moment. Written down, it looks like this.

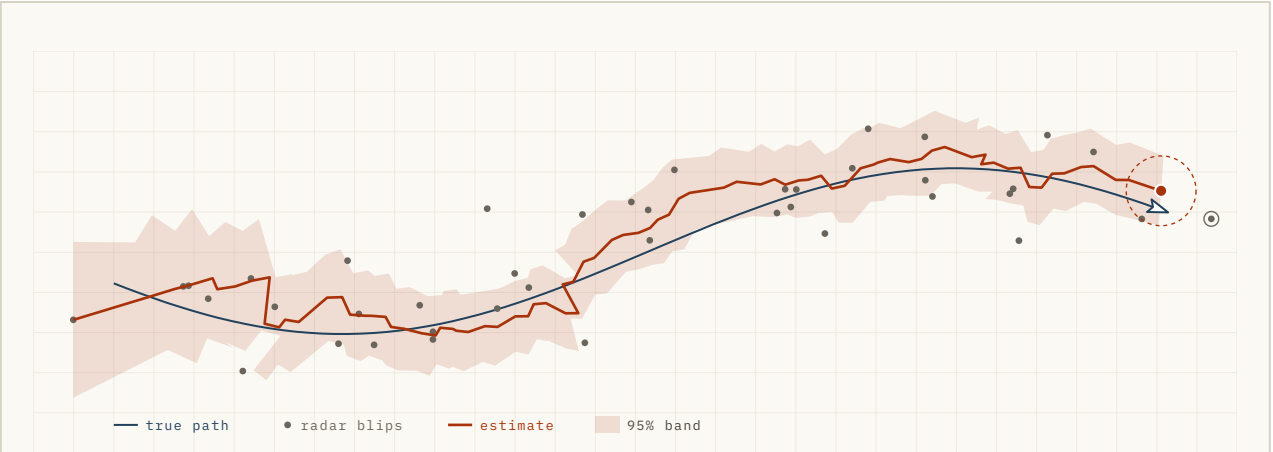
$$p(s_{t+1} \mid s_t, a_t)$$

the chance of what happens next, given where you are now
and what you do

FIG. 1.9 Every system below makes a different choice about what belongs in place of *s*. It could be pixels, 3D geometry, a short list of numbers, or an embedding. That choice is the main axis the five definitions separate along, so keep it in mind.

The oldest ancestor is the Kalman filter, from 1960. Rudolf Kalman, a Hungarian-born engineer, set it out in *A New Approach to Linear Filtering and Prediction Problems*. A filter here is a piece of maths that cleans up noisy readings. If you feed it a radar's blips on a plane, it keeps a running best guess of where the plane is and how fast it is going.

Within a decade it was flying in Apollo's guidance computer. It is still the maths inside a GPS receiver. I'd say that is one half of what a world model does. There is a hidden state you never see directly, and the job is to estimate it from what you can see. Under the hood it keeps two stories about where the plane is, one from physics and one from the radar, and blends them into a guess sharper than either. I take that blend apart in chapter 5; here the shape is what matters.



The band is tighter than the radar's scatter: the filter trusts its own dynamics as much as the radar, and it carries a speed it never measured.

LATEST BLIP OFF BY	ESTIMATE OFF BY	BAND, 95%
420 m	140 m	±260 m
SPEED, ESTIMATE / TRUE	GAIN, PHYSICS 0 TO RADAR 1	
110 / 105 m/s	0.27	

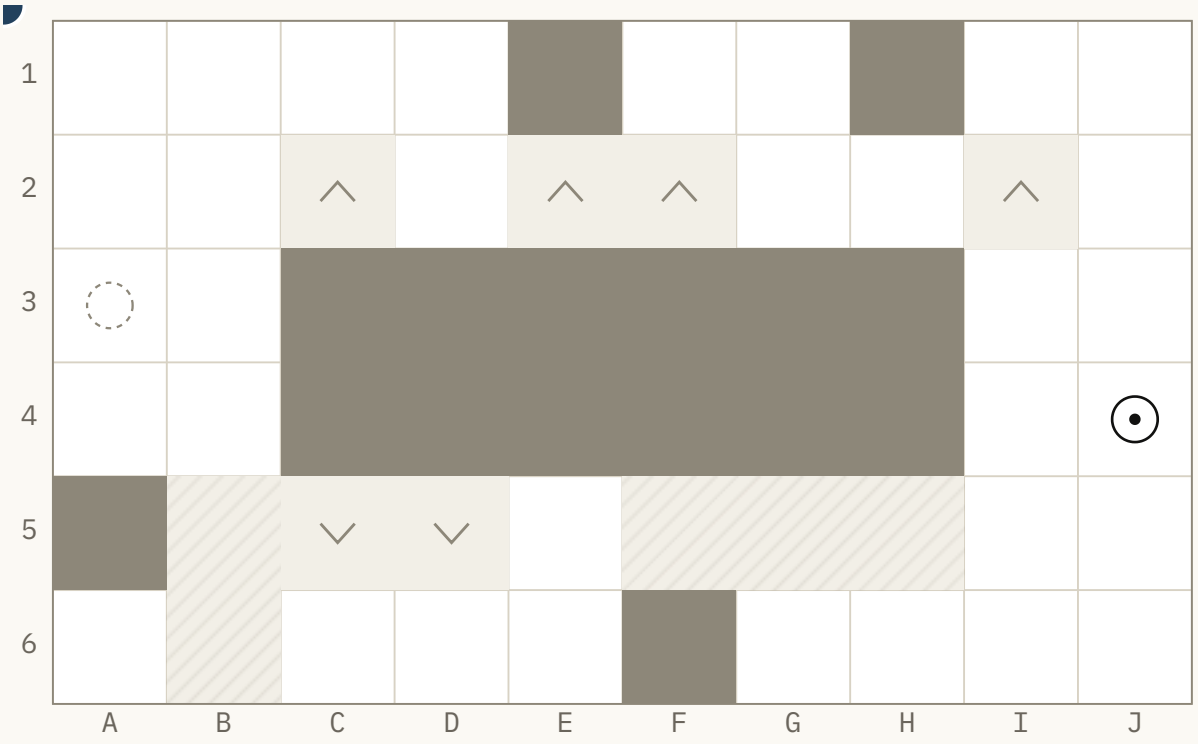
FIG. 1.10 A plane flies along a smooth path, and a radar reports its position every couple of seconds with noise on top. A Kalman filter keeps a running guess of where the plane is and how fast it is going. Scrub the time or press play, and watch the band around the estimate tighten as the blips arrive. Turn the radar noise up, and the estimate still sits closer to the plane than the blips do. Drop the blips for a stretch and it coasts on its own dynamics while the band opens. The gain readout is how far it leaned toward the radar at each step, a number chapter 5 takes apart. The greyed-out control is the point I want you to notice: the filter can propagate a turn you hand it, but nothing here learns the dynamics or weighs one turn against another. Distances are illustrative.

What it does not do is the other half. You can hand a Kalman filter a known control input, a turn you have decided on, and it will propagate it, so it can tell you where that turn would take the plane. But the rules for how the state moves are written in by the engineer rather than learned, and nothing in it compares two turns to pick one.

The other half took thirty years and a different field. By 1990 neural networks, programs that learn from examples rather than from rules someone typed in, were back in fashion. Three groups reached for the same design within a year of each

other, so let's take them one at a time.

Jürgen Schmidhuber, then a young researcher in Munich, built a system in two parts. His 1990 technical report had a title only he could love: *Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments*. One network learned to predict what the world would do next. A second one chose actions, and the first told it what each choice would lead to.



WORLD MODEL

s, a ⟶ s′

One square per move. Knows the walls, not the patches.

CHOOSER

Plans the route inside the model. Takes its first step.

No laps finished yet. Reaching the goal ends a lap and the dot goes back to the start.

LAP	STEPS THIS LAP	SURPRISES THIS LAP	CORRECTIONS HELD
1	0	0	0

FIG. 1.11 This is the 1990 design in miniature. Point at an arrow and the world model draws where it thinks you would land. Press it, and the world answers. The model knows about the walls but not the patches: ice carries you on, and drift and lift push you off your row. Now switch on "Learn from mistakes" and press "Run a lap" three times. You should see the surprises fall lap by lap,

because the chooser now plans inside a model that knows the route. Then press "Forget", switch learning off, and run three more: every lap is as surprised as the first.

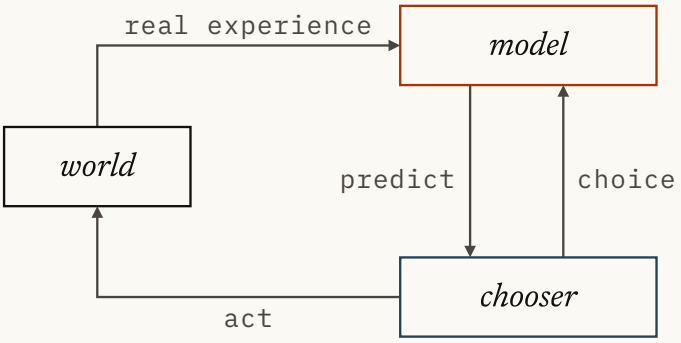
He called the first network the world model, and that is where the phrase comes from. Schmidhuber, who later co-invented the LSTM memory cell, has spent a career insisting the field forgets who had its ideas first. On this one, the record is plainly his.

In the same period Richard Sutton built Dyna. It was a learning agent that practised partly on real experience and partly on experience its own model made up. His paper was *Dyna: an Integrated Architecture for Learning, Planning and Reacting*.

Sutton and Andrew Barto went on to write the textbook every reinforcement learning student still reads. The two shared the 2024 Turing Award, announced the following spring.

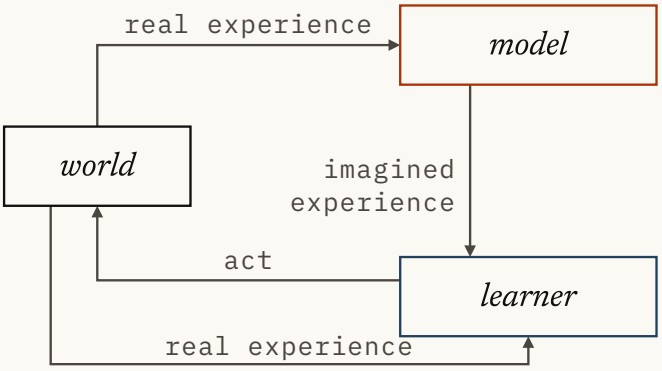
JÜRGEN SCHMIDHUBER 1990

Making the World Differentiable (technical report FKI-126-90)



RICHARD SUTTON 1991

Dyna, an Integrated Architecture for Learning, Planning and Reacting



SEBASTIAN THRUN, KNUT MÖLLER AND ALEXANDER LINDEN 1990

Planning with an Adaptive World Model

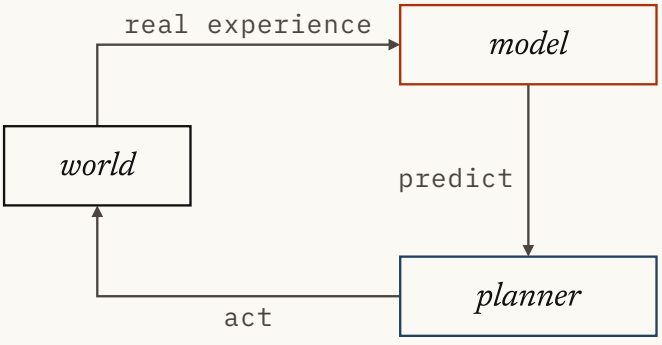
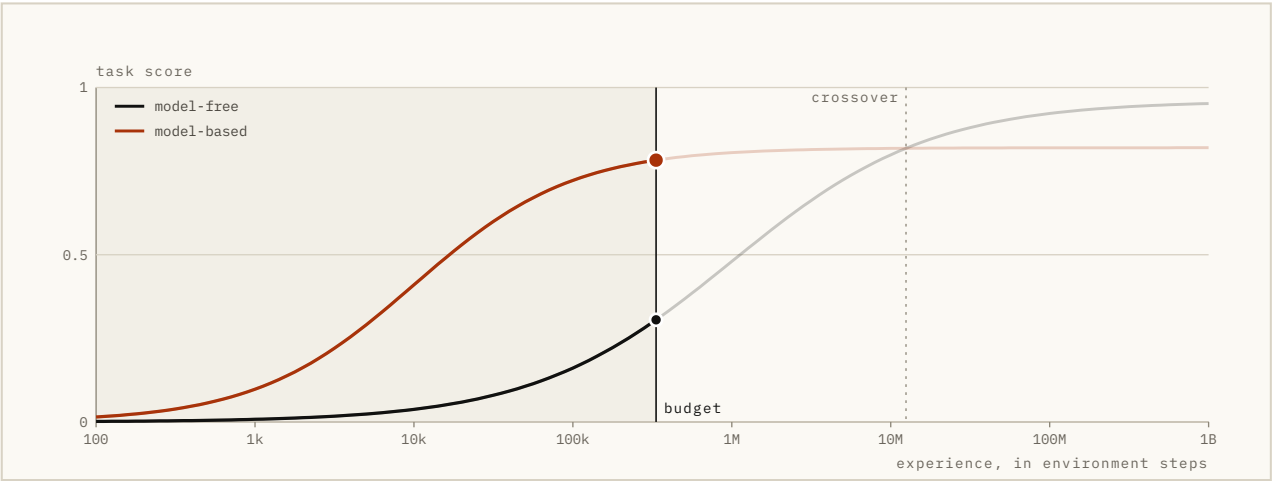


FIG. 1.12 Here are the three systems from 1990 and 1991, each drawn as a loop. Click a card to enlarge it and read what its model does and what uses it. Then turn on the shared shape and watch: the world and the arrows fade, and the same two boxes light up on every card.

Sebastian Thrun, Knut Möller and Alexander Linden made the third, in *Planning with an Adaptive World Model*. As you can see from the figure, all three had the shape of the modern idea. A learned model says what happens next, and something else uses it to decide.

The label did not stick, though. The networks of 1990 were small, and their worlds were mazes and a few dozen numbers. Nobody could learn a useful model of anything with a camera in it.

For most of the next twenty-five years the model-based line, meaning agents that carry a model and use it, stayed a minority interest. When deep learning reached reinforcement learning, the headline result was DeepMind's DQN in 2013. It learned Atari games from pixels with no model of the game at all. Model-free had the momentum, and the two-part design of 1990 looked like a relic.



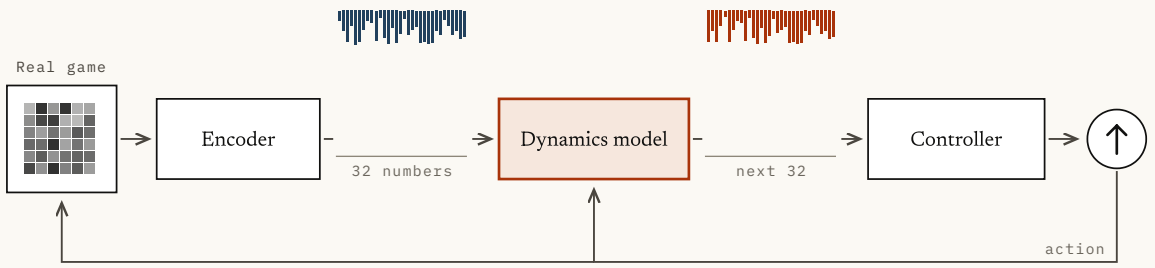
AFFORDABLE STEPS	MODEL-FREE SCORE AT BUDGET	MODEL-BASED SCORE AT BUDGET
about 330 thousand	0.3	0.8

FIG. 1.13 The curves here are shapes rather than measurements. Pick what one step costs, then drag the budget and watch where the shaded region stops. In a simulator, a program that imitates the world, the budget reaches past the crossover and the slow climber wins. On a robot it stops long before that, and the only agent that has learned anything is the one carrying a model. Then switch on the bad model to see what happens when the model is wrong in the places you go.

David Ha and Jürgen Schmidhuber's *World Models* (2018) is the paper that made the label popular. I read it as a reply to that momentum. Ha was then at Google Brain. The paper showed the 1990 design working on a car-racing game and a level of the video game DOOM, in three pieces.

The first is an encoder (a network that squeezes a video frame down to a short list of numbers), thirty-two numbers here. The second is a dynamics model, which predicts where those numbers go next. The third is a small controller, the piece that picks actions.

The controller was trained almost entirely inside the model's own imagined rollouts, futures the model played out for itself. Then it was moved back into the real game, and it still worked.



PIECE | ENCODER | DYNAMICS MODEL | CONTROLLER

IN — The thirty-two numbers, plus the action taken.

OUT — The next thirty-two numbers.

FOR — Predicts where those numbers go next. It is the one piece that can run on its own output.

REAL STEPS	IMAGINED STEPS	LATENT WIDTH
0	0	32
		Ha and Schmidhuber's choice

FIG. 1.14 Ha and Schmidhuber's three pieces, drawn as a loop. Select a box to see what goes in and what comes out, then press Step and watch the thirty-two numbers change. Now flip the switch to run inside the dream: the dynamics model feeds its own prediction back, and the game is no longer consulted. That is how the controller was trained before it was moved back into the real game. Then have a go at the question underneath.

One thing that can mislead you in the papers is the naming. Ha and Schmidhuber call their third module the controller, meaning the small policy that picks actions. The world model is the module beside it, the one the controller plans inside. If you called the whole thing "the controller" you would name it after the one piece that is plainly not a world model.

PlaNet came the same year, from Danijar Hafner and colleagues, with Ha among the authors. The paper was *Learning Latent Dynamics for Planning from Pixels*. It learned those dynamics straight from pixels. Then it planned in latent space (latent just means the model's own compressed description: a few hundred numbers instead of a million pixels).

At every step it tried many action sequences inside the model and took the best first move. Dreamer (2019) came from the same group, in a paper called *Dream to Control*. Instead of searching for a plan at every step, it learned its behaviour from futures imagined inside the model.

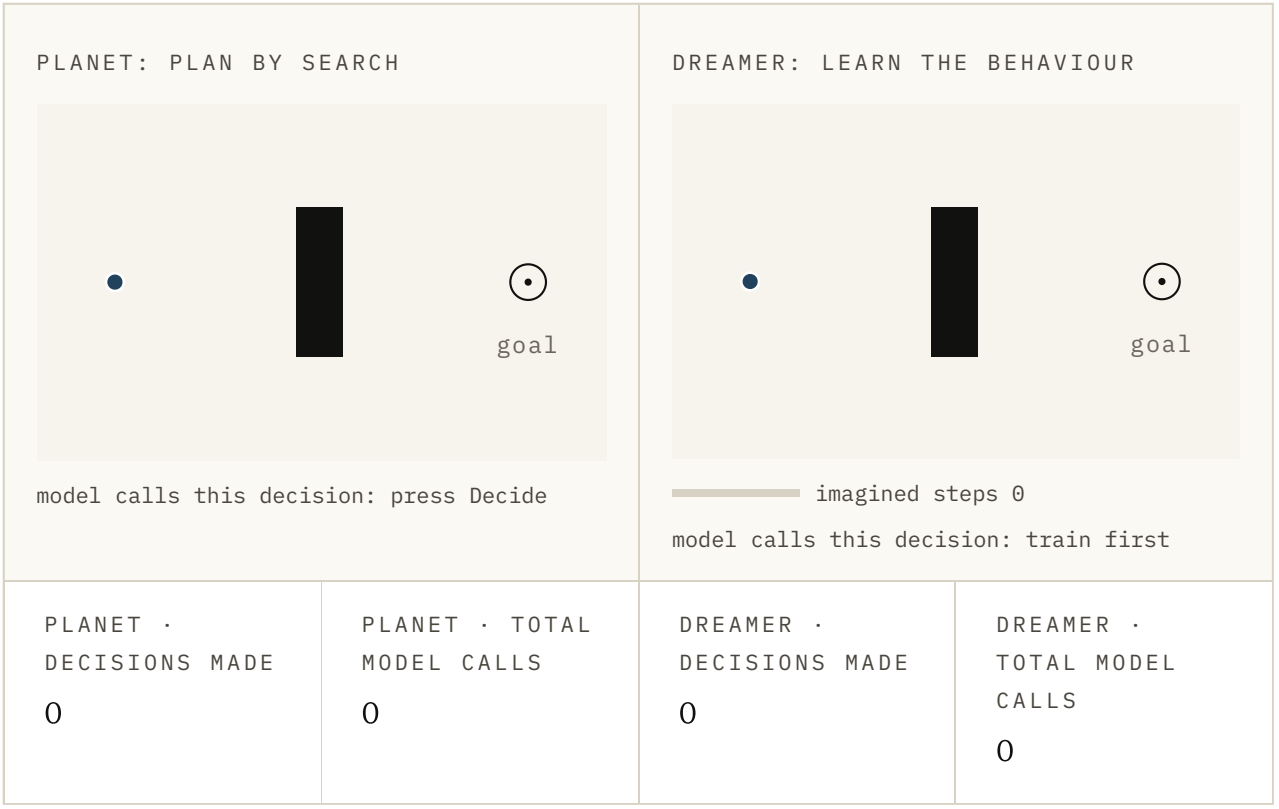
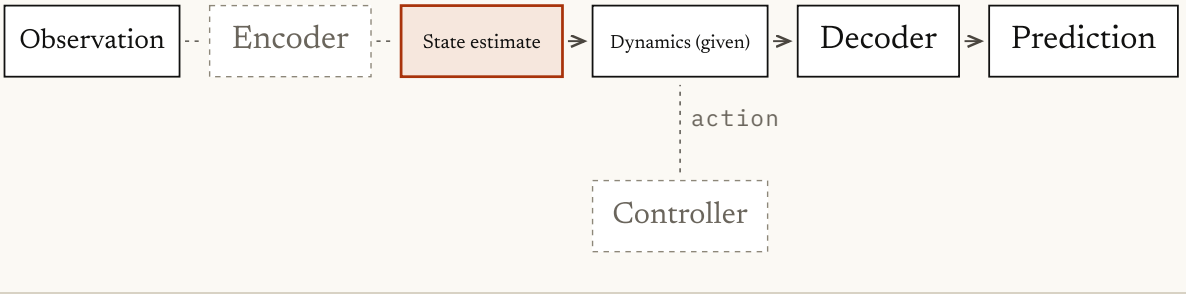


FIG. 1.15 Two copies of the same small world. On the left, press Decide: the model imagines twelve futures, keeps the best, and the dot takes one step, so every decision costs seventy-two model calls. On the right, press Train in imagination once and then Decide: the behaviour was learned in advance, so each step costs the model nothing. Watch the two total-calls counters rather than the dots.

Its third version reached *Nature* in 2025 with one set of settings across more than 150 tasks. By the end of 2019, though, the reinforcement learning reading of the term was settled.

Then the term travelled, in two directions at once. In 2022 Yann LeCun published *A Path Towards Autonomous Machine Intelligence*. He was Meta's chief AI scientist at the time, and he is a Turing Award winner. He argued that predicting every pixel of the next frame is the wrong job, because most pixels are detail nobody can predict.

JEPA, the family of models that followed at Meta, predicts a summary of the next frame instead. That lets it drop the decoder, which is the part that would have turned the prediction back into pixels.



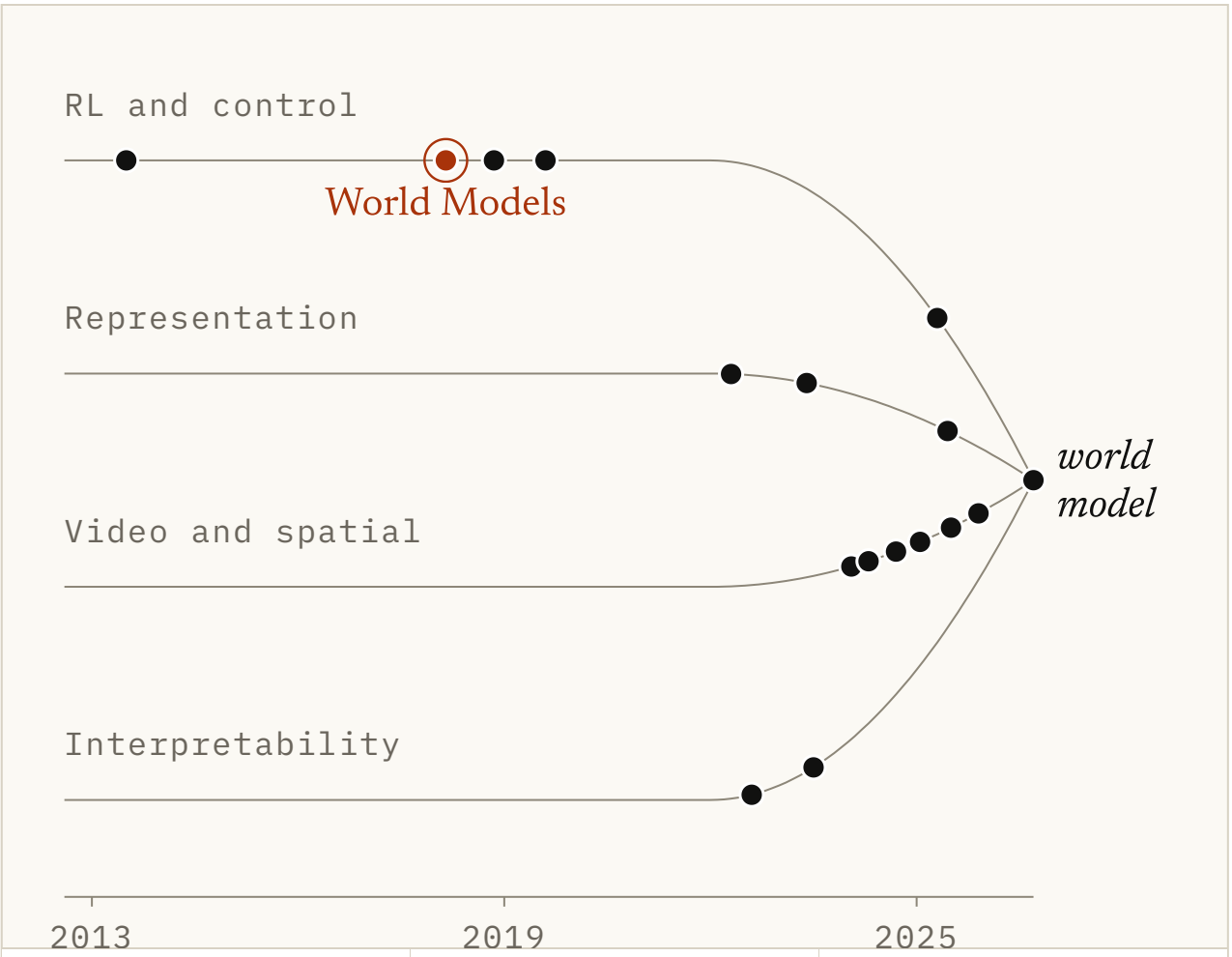
Recover a hidden state from noisy measurements by running a forward model and correcting it against what you observe. The dynamics are supplied, not learned, and nothing here learns them or chooses between actions.

FIG. 1.16 Scrub through the years and watch the parts. Each era switches a part on, relabels it or points it at a new target, and now and then one switches a part off. Nothing ever gets replaced, and that is how systems built for different reasons ended up sharing a name.

The other direction went toward worlds you can see and walk through. In February 2024 OpenAI published Sora with a report titled *Video generation models as world simulators*. The same month Genie arrived from Jake Bruce and colleagues at DeepMind. It learned its own set of actions from unlabelled internet video of games, then let you play what it drew.

Cosmos from NVIDIA and Marble from World Labs followed in 2025. Meanwhile the interpretability people had borrowed the term for something with no demo at all. Their claim was that a model trained for one job had grown a model of the world inside itself.

By June 2026 World Labs had published *A Functional Taxonomy of World Models*, a sorting system for the mess, and the dictionaries had begun. Four traditions each held a different piece of one problem, and from the outside the term looks confused.



2018 · RL AND CONTROL

World Models

David Ha and Jürgen Schmidhuber, 2018. An encoder, a dynamics model and a small controller trained almost entirely inside the model's own imagined rollouts.

WHAT THEY MEANT BY THE PHRASE

A learned model of what happens next, good enough to train inside.

WHAT IT HANDS YOU

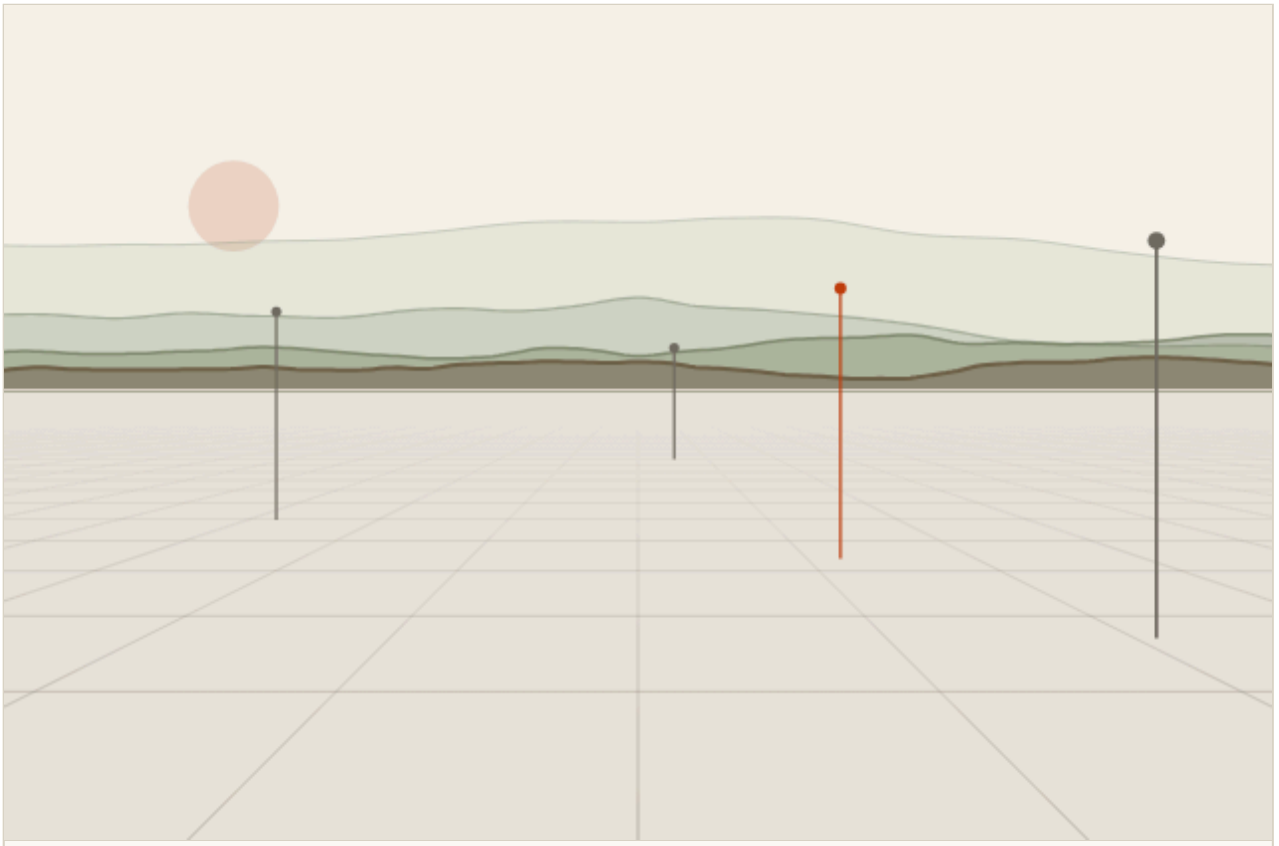
Compact state: thirty-two numbers a frame.

FIG. 1.17 Drag the year back to 2013 and come forward. For most of the decade only one lane has anything on it, and the guides only begin to bend after 2022. Click a dot to see who used the phrase, what they meant by it, and what their system hands you. As you can see, the lines meet at one phrase rather than at one thing.

The four only noticed they were neighbours when their outputs started to look alike. By then each of them had been using the term for years.

The test that was supposed to settle it

One test ought to sort all five out, and it takes about four seconds. Turn around and walk away, then turn back: is it the same room? Something that holds a world inside it keeps the furniture where you left it, and something that is only drawing pictures cannot. I call it the turn-around test.



WHAT IS STORED

nothing. each frame is drawn from the heading

HEADING	FROM START	LANDMARK
0°	0.0 strides	13° right, 5.5 strides

FIG. 1.18 The turn-around test, made playable. With the switch off, press Turn away and back and look at where the vermilion (orange-red) post comes back. Then switch the world on and run it again. The panel on the right shows you what is stored in each case.

It does not work, and the figure above shows why. Leave the switch off and press "Turn away and back". You should see that the post has drifted, because nothing behind the picture is holding it there.

Now switch "Hold the world" on and do it again. The post stays put, because a stored coordinate is being read back. Excellent.

So those are the two ways to pass the test. You can store the room and look it up when the viewer turns back. Or you can get very good at drawing a room that matches the one you drew a moment ago. From the outside they look the same, and only the switch tells them apart.

A large model trained on a lot of video learns the second way. Nobody gave it a room to keep. It got good at continuing what it had already started, and continuing consistently is part of that. That is why Genie 3's persistence is real, and why the turn-around test was never going to catch it.

> DeepMind reports this of Genie 3: scenes holding together over several minutes, and details coming back when you return to them. I'd take it as their report rather than something anyone outside the lab can check.

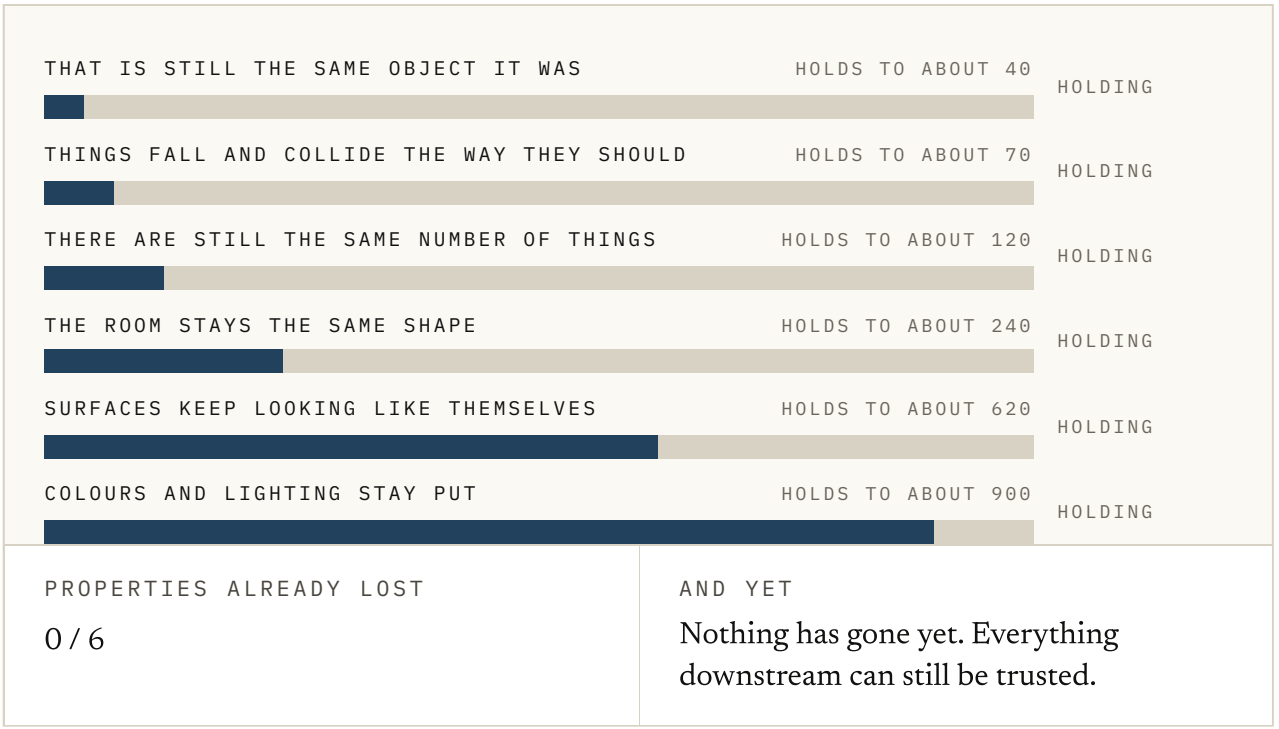


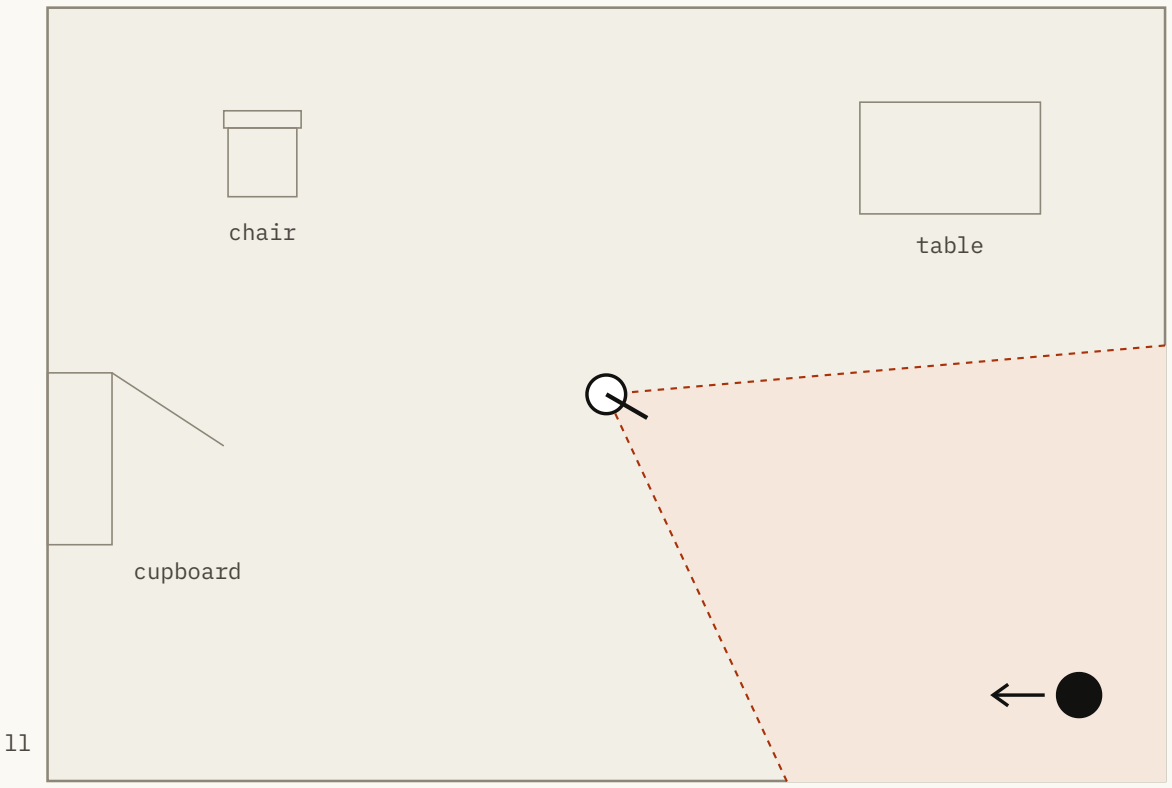
FIG. 1.19 This is persistence learned through generation, watched over a long run. Drag out to a thousand steps and read off which properties are still holding: texture and count go last, and identity and physics go first. Nothing here was stored; all of it was continued.

So the turn-around test cannot sort them, but it was reaching for the right distinction. When you turn away from the chair, it stops being *visible*, but it does not stop existing. Those are two different things.

What is there, whether or not you can see it, is called the **state**. The part of it that you can see right now is called the **observation**.

> State is the underlying reality of the world; complete in principle, but never directly visible to any agent inside it. Observations are an agent's partial view of that reality.
>
> World Labs on the distinction underneath the taxonomy (A Functional Taxonomy of World Models, June 2026)
> https://www.worldlabs.ai/blog/taxonomy-of-world-models

You never get the state, only observations, and you work backwards from them. Think of the chair behind you, a ball still rolling out of shot, or whether the cupboard is open. All of it is real and none of it is visible.



	STATE what is there	OBSERVATION what the camera sees now
CHAIR	here	?
BALL	rolling left, at 7.4 m	rolling left, at 7.4 m
CUPBOARD	open	?
TABLE	here	?

HEADING	IN VIEW	TIME STEP
30°	1 of 4	0

FIG. 1.20 Turn the camera with the slider and watch the two columns. The state column never changes what it lists. The observation column only shows what falls inside the wedge, and marks the rest with a question mark. Now turn away from the ball, press Step time once or twice, then turn back: the ball has moved, and your last observation of it was wrong the whole time.

Working out the rest from the part you can see is called **partial observability**.

None of this is new. A Cambridge psychologist, Kenneth Craik, described the fix in 1943, in a book called *The Nature of Explanation*.

He was twenty-nine when it came out. Two years later he was dead, knocked off his bicycle on a Cambridge street. That was seventeen years before Kalman, and long before there was anything to build it out of.

> If the organism carries a 'small-scale model' of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise... and in every way to react in a much fuller, safer, and more competent manner to the emergencies which face it.
>
> `Kenneth Craik on internal models, nearly fifty years before anyone trained one (The Nature of Explanation, 1943)`

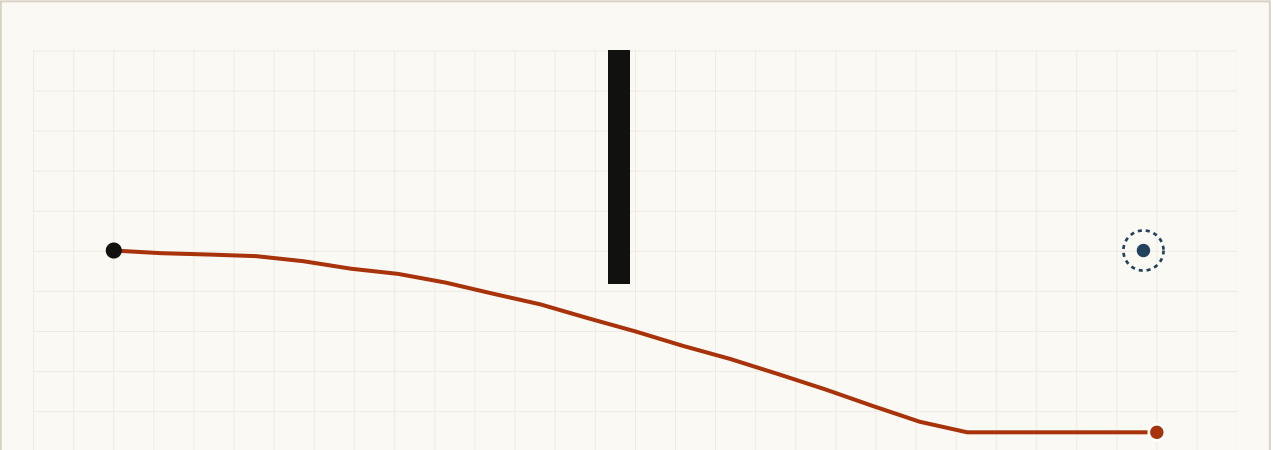


FIG. 1.21 Craik's small-scale model, run by a machine. The planner writes down runs of actions, plays each one through the model, and keeps the best. Raise the budget and you should see the rejects fade and the plan sharpen.

If you carry a small model of the world in your head, you can test an action before you take it. Every definition on the map is a different answer to what Craik's small-scale model should hold.

What they were all fighting over

One object sits underneath all five definitions. World Labs derives their own taxonomy from the same object, and Sutton's Dyna was running round it back in 1990.

An environment has a state. An agent receives observations that reveal part of it, and from observations and memory it builds an internal state. A model predicts how things could change. The agent acts, the world changes, and a new observation arrives.

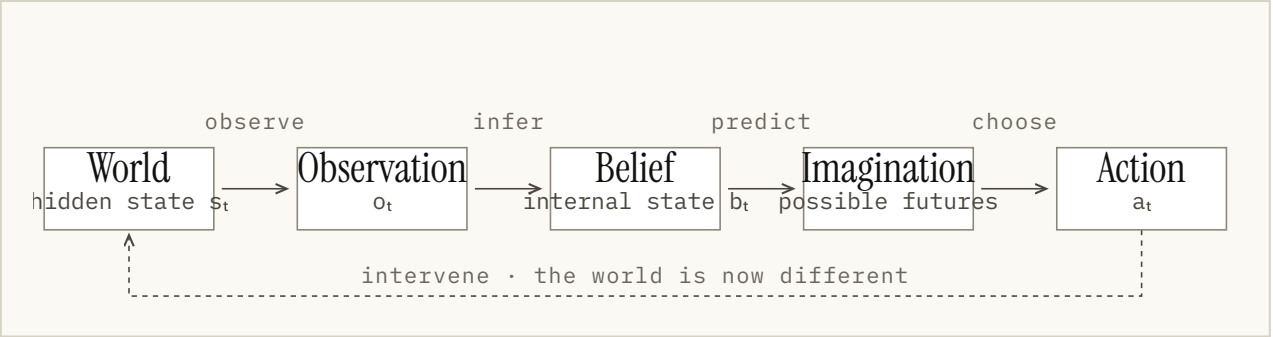


FIG. 1.22 The agent-environment loop, with each definition shown as a specialist on one arc. Systems that look incompatible share a name because each one owns a piece of the same loop.

If you draw the loop that way, three problems fall out of it. They are the same three problems the field has been arguing about since Kalman wrote down a filter in 1960. Each gets its own chapter later.

How sure should it be? Throw a ball behind a wall and ask where it is now. A single exact position is the wrong shape of answer, because you cannot know. It might have bounced off something, or stopped against a kerb, or still be rolling.

A model that hands back one confident coordinate has thrown away the fact that it was guessing. PlaNet carries two things at once instead. One part is deterministic (computed the same way every time, so it always follows from the last state). It holds whatever reliably follows from what came before.

The other part is stochastic (sampled rather than computed, so the same input can produce several different futures). It holds whatever could still go either way. You can think of the split as the model admitting what it does not know.

TIME HIDDEN	WIDTH OF THE SPREAD	THE SINGLE DOT
0 steps	collapsed to a point	on the ball

FIG. 1.23 Drag time forward and watch the ball go behind the wall. The faint dot is the answer a deterministic model gives: one coordinate, moved on at the known speed. Switch on the stochastic part and the model also carries a spread, which widens the longer the ball is hidden. Keep going past the point where the dot comes out, and ask yourself which answer held up.

Do actions change the prediction? *What happens next* and *what happens if I push this* are two different questions. Only the second one lets you compare options before choosing.

A model that answers it is called action-conditioned (it is told which action was taken, so it can answer what-if instead of only what-next). Kalman's filter could propagate a control input you gave it, but it never learned the dynamics or weighed one action against another. The learning and the choosing came later.

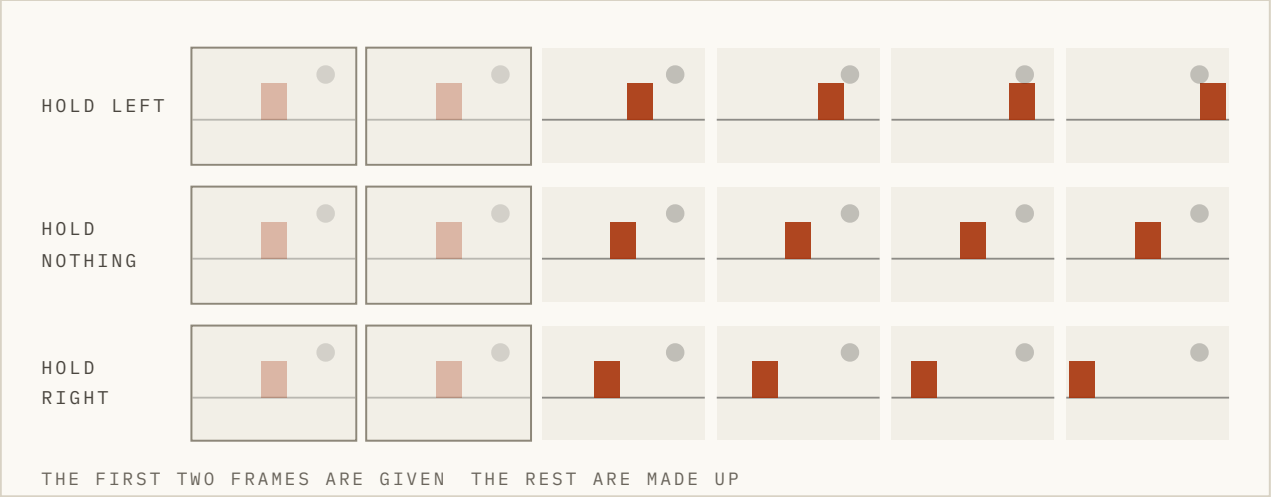


FIG. 1.24 One start and three futures, chosen by which key you hold. Turn the action input off and watch the three rows fold into one: without an action there is only what probably happens next, and no longer what happens if you do this.

How far can it roll forward? One prediction is rarely enough. To plan, a model has to feed its own answer back in and predict again, and then again, on top of what it just made up. The number of steps you ask for is called the **horizon**.

Errors pile up along that stretch. Nothing dramatic happens at any single step, which is what makes the problem hard to notice. Get step one slightly wrong, though, and step twenty can be somewhere else entirely.

$$p(s_{t+1} \mid s_t, a_t) \implies p(s_{t+1:t+H} \mid s_t, a_{t:t+H-1})$$

Ask for the next state instead of every state from here to H
, and you still only get the same single state you are in and a whole plan.

FIG. 1.25 The same model, this time asked for a stretch of future instead of a single step. The future you ask for and the plan you hand in both grow, and the state you are given does not.

Dreamer exists to learn behaviour across many imagined steps. DeepMind names long-horizon coherence as the big open problem for its generated worlds. That means a scene that stays true to itself over many steps.

Dreamer is a latent dynamics model from 2019, and Genie 3 is a pixel renderer from 2025. They sit at the two ends of the map, fighting the same problem from opposite sides.

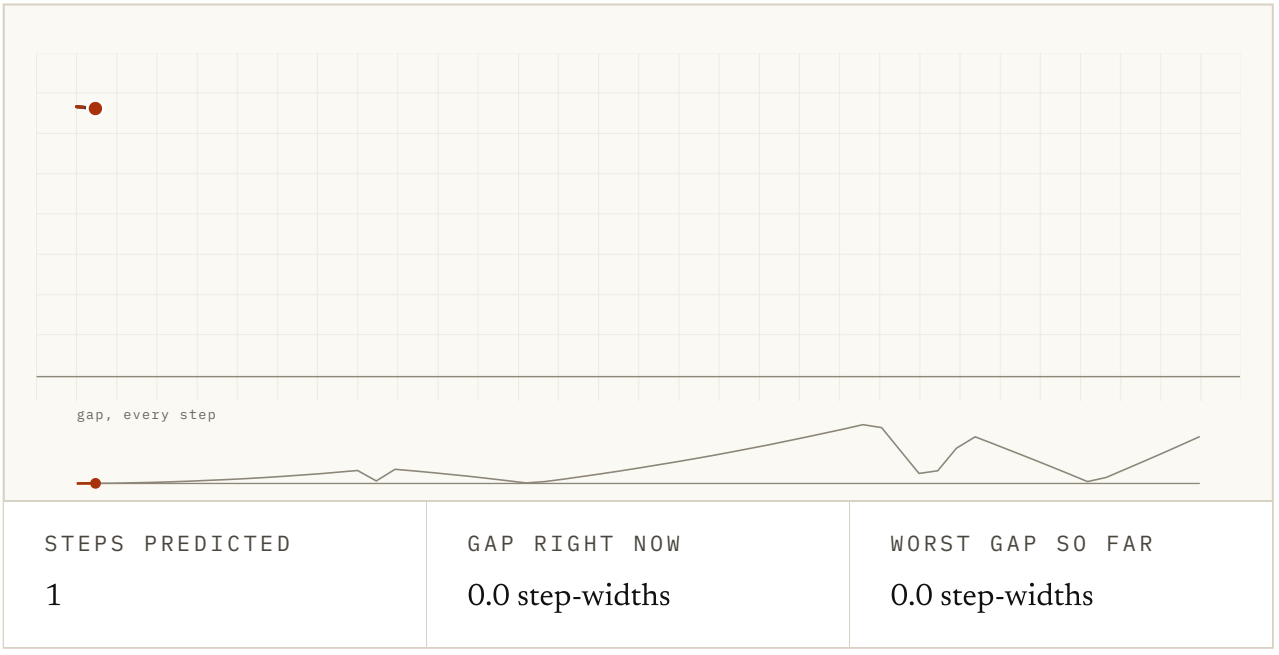


FIG. 1.26 Both dots start in the same place under the same rules, except that one is running dynamics that are a few per cent off. Drag to about 8 and you'll see they are still together. Keep going, and notice that nothing breaks at any single step, but the gap keeps piling up.

So sixty years on, the open problem is drift, the very thing Kalman's filter was built to correct. The difference is that an imagined rollout has no fresh readings to correct against.

Now let's go back to the clip. It is a Renderer, and you can say why: it predicts observations. It holds the room because it learned to, and nothing in it promises geometry underneath.

The next one that goes past your timeline will be called a world model too. The question to ask is which of the five it is, and whether the person posting it could tell you.

Hopefully this chapter helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section. I would be glad to hear from you.

Try it

1 It takes one photo of a kitchen and returns a mesh you can import into a game engine, with collision volumes on the worktops.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Simulator. Something else can open it and compute against it. Collision volumes are the giveaway: they exist for a physics engine to bump into, not for you to look at.

2 You are watching a camera feed of a room. There is a chair behind the camera. Where does that chair sit?

a. Outside the state, because nothing can see it
b. In the state but not the observation
c. In the observation but not the state
d. In neither, until the camera turns

> ANSWER b. In the state but not the observation. Turning away does not delete furniture. The chair is part of what is there, just not part of what you can currently see. The gap between those two is where most of the hard problems in this course come from.

3 You hold a key and it streams video of a city that has never existed, a frame at a time, reacting to which way you steer.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Renderer. The output is the picture. It may well stay consistent as you drive, but nothing underneath is obliged to, and there is no city to hand anyone.

4 A system keeps the room consistent when you turn away and turn back. What has that proved?

a. It is storing the room
b. It is not storing the room
c. Nothing on its own
d. It must be a Simulator

> ANSWER c. Nothing on its own. There are two ways to pass that test, and from the outside they are identical. You can keep the room, or you can be very good at redrawing it. The result is the same, so the result cannot tell you which.

5 It is trained only to predict the next move in chess games. Researchers later probe it and find it tracks where the pieces are.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Implicit Model. Nobody built a chess model here and nobody can run one. The claim is about structure found inside a network trained for something else, which is a claim of a different kind from all the others.

6 A model is slightly wrong at every step. You feed its own output back in twenty times. What happens to the error?

a. It stays about the same
b. It roughly doubles
c. It grows, unevenly, and can end up somewhere else entirely
d. It cancels out over enough steps

> ANSWER c. It grows, unevenly, and can end up somewhere else entirely. Each imagined state becomes the input to the next prediction, so mistakes are built on. Nothing dramatic happens at any single step, which is what makes it hard to catch.

7 It hides part of a video and learns to predict a summary of the hidden part. Once trained, the predictions are thrown away and the rest is bolted onto a robot.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Representation. The forecast was scaffolding. What survives training is the way it learned to describe things, which is the product.

8 Going from predicting one step to predicting H steps, what does NOT get bigger?

a. The stretch of future you are asking for
b. The number of actions you have to supply
c. What you are given to start from
d. The number of ways it can go wrong

> ANSWER c. What you are given to start from. However far ahead you ask, you are still standing in exactly one place with one observation of it. The question grows; the evidence does not.

9 Given a compact state, a few numbers describing the scene, and a motor command you are considering, it returns the compact state you would be in next. A search loop calls it a few hundred times per decision.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Dynamics Model. It hands you state rather than a picture, and you can roll it forward under actions nobody has taken yet. That is the Dynamics Model's contract, and the search loop is what it is for. Had it returned the next sensor reading instead, you would be looking at a Renderer.

10 Why does the Kalman filter count as ancestry rather than as one of the five?

a. It is too old to count
b. Its dynamics are supplied rather than learned, and it never compares actions to choose one
c. It cannot take a control input at all
d. It only works on linear systems

> ANSWER b. Its dynamics are supplied rather than learned, and it never compares actions to choose one. It does half the job well: work out a hidden state from noisy measurements, and it will even propagate a control input you hand it. What it never does is learn the dynamics or weigh one action against another, and those are what make the rest of these useful for choosing. Linearity is a detail of the original, not the reason.

11 One era in the history removed a part instead of adding one. Which, and why?

a. The encoder, because pixels stopped mattering
b. The decoder, because the prediction target moved off the pixels
c. The controller, because planning was abandoned
d. The dynamics, because they became implicit

> ANSWER b. The decoder, because the prediction target moved off the pixels. JEPA predicts a summary of the next frame rather than the frame, so nothing needs to turn the prediction back into pixels. That is the same reason the forecast can be discarded and the features kept.

12 A lab generates photorealistic video of motorway driving to train a self-driving stack. It is marketed for robotics.

`The Renderer · The Simulator · The Dynamics Model · The Representation · The Implicit Model`

> ANSWER The Renderer. This is the trap. Being aimed at robots suggests a Simulator, but the output is still video, with no geometry anyone can collide against. What a system is for is a weaker clue than what it hands you. Even that is not a complete answer, because a large platform can ship several interfaces, so the question is which one you are about to build against.

13 A lab reports its system stays coherent for several minutes. You cannot run it yourself. How should that sit in your notes?

a. As a fact, since they built it
b. As reported, not checked
c. As false until proven
d. As irrelevant to the category

> ANSWER b. As reported, not checked. This is not scepticism for its own sake. Some claims you can open and verify, like a mesh you can load; others you can only receive. Knowing which is which is part of reading this field.

FIG. 1.27 Thirteen questions across the whole chapter. They ask you to place systems I never named, using the ideas the figures were built to teach. One of them is a trap, and it is the kind of mistake I'd rather you made here than in a meeting.

Sources

A Functional Taxonomy of World Models WORLD LABS, 2026 ↗

First-party renderer/simulator/planner split, derived from the agent loop.

https://www.worldlabs.ai/blog/taxonomy-of-world-models

A New Approach to Linear Filtering and Prediction Problems KALMAN, 1960 ↗

The hidden-state ancestor. Paywalled.

https://doi.org/10.1115/1.3662552

Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments (FKI-126-90)
SCHMIDHUBER, 1990 ↗

A recurrent model predicting the consequences of a controller's actions.

https://people.idsia.ch/~juergen/world-models-planning-curiosity-fki-1990.html

World Models HA & SCHMIDHUBER, 2018 ↗

The paper that popularised the modern label.

https://arxiv.org/abs/1803.10122

Learning Latent Dynamics for Planning from Pixels (PlaNet) HAFNER ET AL., 2018 ↗

Raw pixels to stochastic latent state to online planning.

https://arxiv.org/abs/1811.04551

Dream to Control (Dreamer) HAFNER ET AL., 2019 ↗

Behaviour learned from multi-step latent imagination.

https://arxiv.org/abs/1912.01603

I-JEPA ASSRAN ET AL., 2023 ↗

Predicting representations of masked regions, not pixels.

https://arxiv.org/abs/2301.08243

V-JEPA 2 META AI, 2025 ↗

Action-free pre-training, then action-conditioned control.

https://ai.meta.com/blog/v-jepa-2-world-model-benchmarks/

Genie: Generative Interactive Environments BRUCE ET AL., 2024 ↗

Action-controllable generated environments.

https://arxiv.org/abs/2402.15391

Genie 3 GOOGLE DEEPMIND, 2025 ↗

Reports recalling previously seen detail over multi-minute interaction.

https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/

Marble WORLD LABS, 2025 ↗

Gaussian splats plus collider meshes: an explicit structural export.

https://www.worldlabs.ai/blog/marble-world-model

Cosmos NVIDIA ↗

A boundary case: predictive video worlds beside explicit simulation.

https://www.nvidia.com/en-us/ai/cosmos/

Emergent World Representations LI ET AL., 2022 ↗

Board state found and causally manipulated inside Othello-GPT.

https://arxiv.org/abs/2210.13382

Othello-GPT has a linear emergent world representation NANDA ET AL., 2023 ↗

The follow-up that sharpened the finding.

https://arxiv.org/abs/2309.00941

Language Models Represent Space and Time GURNEE & TEGMARK, 2023 ↗

Probes recovering place and date from a language model, and the paper the argument over this sense of the term formed around.

https://arxiv.org/abs/2310.02207

FIG. 1.28 I've ordered these for someone building on this rather than for historical completeness, and preferred first-party material throughout.

02 How Do World Models Work?

BY NILUSHANAN KULASINGHAM

25 MIN READ · INTERACTIVE

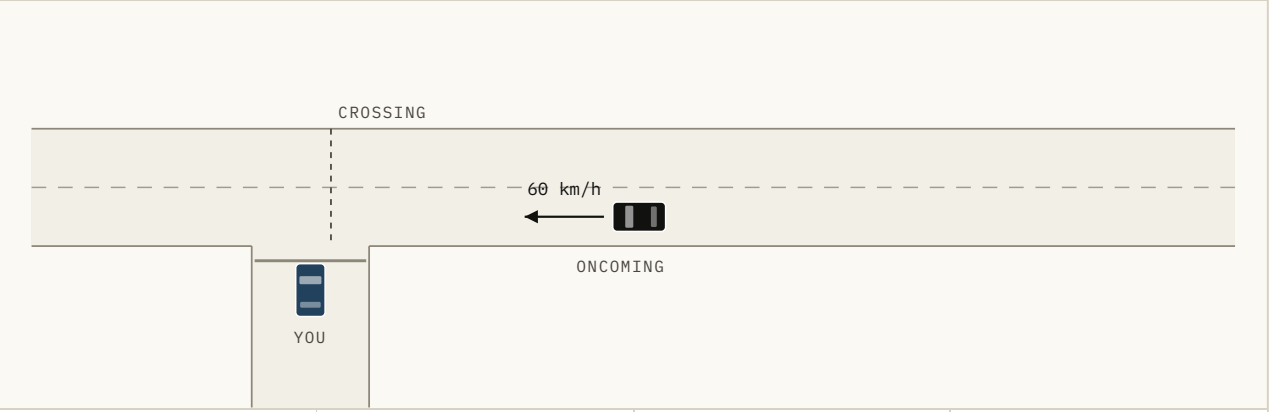
> A model you can run forward lets you try an action before paying for it. That is the oldest idea in the field and still the best one. The trouble is that a search good enough to find the best plan is also good enough to find the places where the model is wrong. Nobody has engineered that away.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/the-idea.

I want to start at a junction, because I think it is the most ordinary world model there is. You are waiting to turn right. You look right and there is a car coming. Somewhere in the next second you decide: go now, or wait for it to pass.

What you do in that second is run the scene forward. That car is there, going about that fast, so in two seconds it will be here. If I go now I am clear of it, and if I wait I am not in its way either. You commit before anything has happened, on your own forecast of it.

Most of the time the forecast is good enough, and the whole thing takes less time than it took to read about. The cases where it is not good enough are the interesting ones. Learner drivers misjudge the speed of oncoming cars, and a wrong forecast here ends in a collision you do not get to undo. Let's make the decision one you can run.



FORECAST ARRIVAL	REAL ARRIVAL	FORECAST WAS OFF BY	OUTCOME
3.6 s	after Go	after Go	not yet

FIG. 2.1 The junction, with the forecast made visible. Set how far away the oncoming car is and how fast it is going, and read what the forecast says. Leave it on the learner, press Go, and watch whether you make it. Then switch to the experienced driver and run the same gap. The numbers are illustrative.

As you can see, nothing about the oncoming car changed between your forecast and the answer. What changed was only how good your forecast of it was, and that decided whether you got across or got hit. Past a certain speed, acting well depends on what has not happened yet.

Take what you have, run it forward, and move for what comes next. That is a dynamics model. Of everything the phrase *world model* has since been stretched to cover, and chapter 1 counted five senses, it is the oldest and the one the term was coined for. The coining happened in 1990, and chapter 1 told that story.

The model itself only predicts what would happen. Something else reads those predictions and chooses, and that something else has its own name, the **policy** (the part that chooses the action, by hard thinking or by pure reflex). Jürgen Schmidhuber's 1990 version kept the two as separate objects, and the split holds all the way through this chapter.

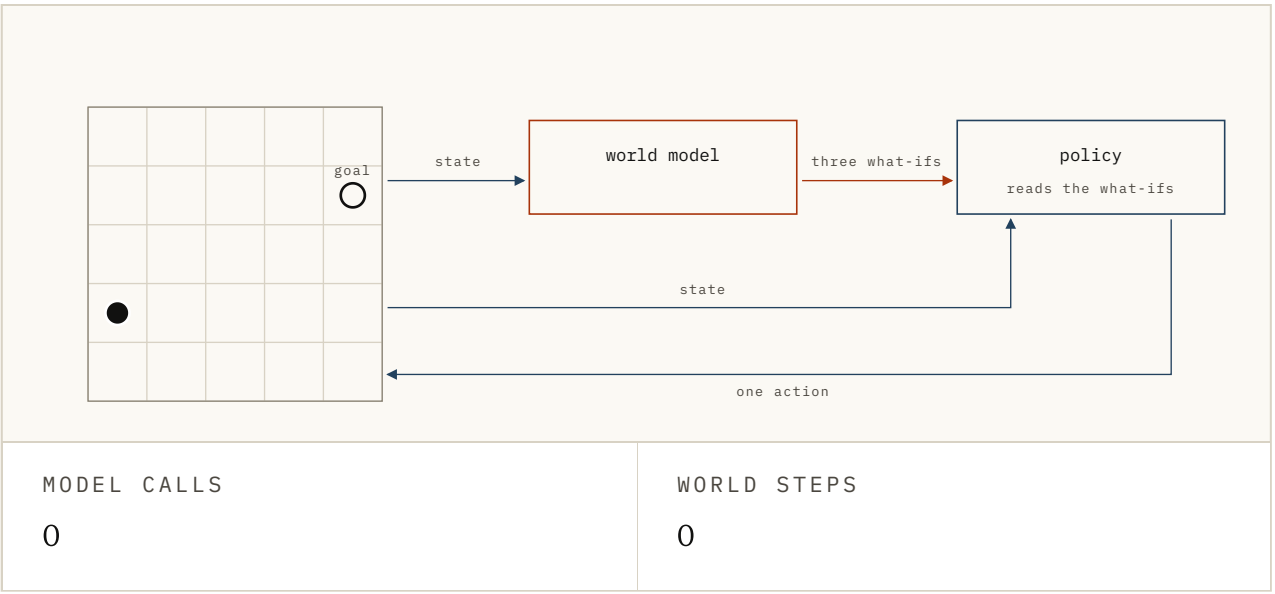


FIG. 2.2 The two halves, drawn apart. Press Ask the model and it answers three what-ifs, one per action, without touching the world. Press Act and the policy picks one of them and the world moves once. Now switch the model off and press Act again: the policy still acts, straight from what it sees, and nothing was imagined. Only the policy ever touches the world. Illustrative.

So when somebody tells you their system is a world model, ask which half they mean. The first half gets you a great deal, and it also fails in a way nobody has engineered away.

What the model holds, and what it can drop

The shape of it has not changed since 1990: where you are, what you do, what comes next. A dynamics model learns that mapping (dynamics means how things change over time, so this is a model of what happens next). It usually learns what the step was worth as well. Written down, with a hat on, it looks like this.

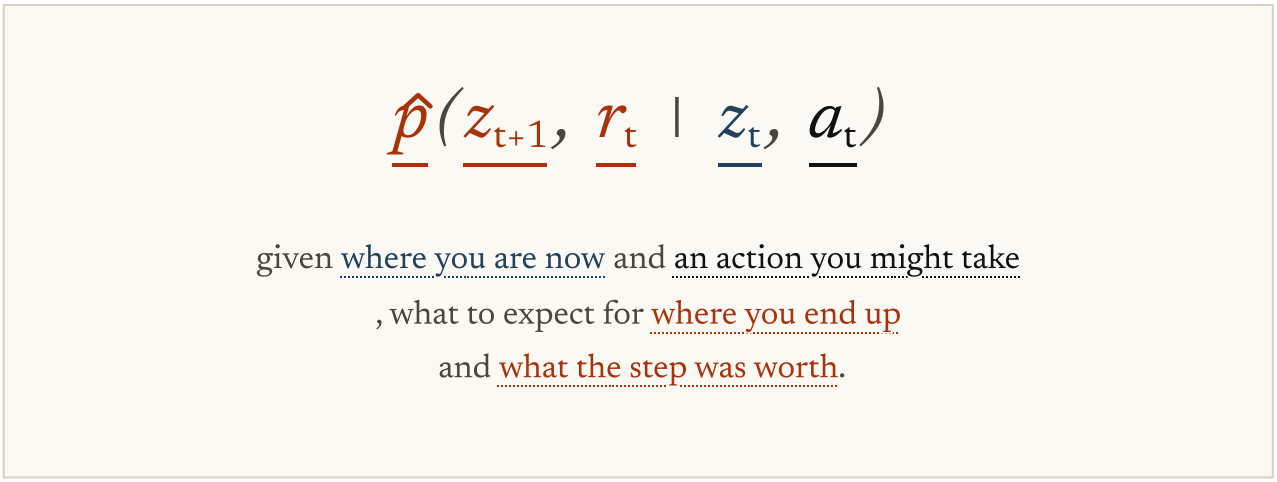


FIG. 2.3 The object everything else rests on. Hover any symbol, or any phrase under it, and both light up. The hat says *estimated*, and I'd keep an eye on it, because every failure later in the chapter is that hat coming due.

Where you are used to be written down by hand. In the control theory of the 1960s, the world of Rudolf Kalman's filter (the maths inside every GPS receiver, and the maths that steered Apollo), the state of a thing was its position and speed. The field has spent the years since 2018 letting go of that idea, in three steps, so let's take them in turn.

PlaNet was the first step. It is a 2018 system from Danijar Hafner and colleagues, described in *Learning Latent Dynamics for Planning from Pixels*. It learns its model from images and plans inside that model. It works out where it is from the pictures and never names any part of that answer.

MuZero was the second step, and it came from outside. Julian Schrittwieser and colleagues at DeepMind built it as the heir of AlphaGo, which beat the best human Go players, and AlphaZero, which learned chess, Go and shogi from the rules alone. MuZero was not even given the rules.

The paper, from 2020, is *Mastering Atari, Go, chess and shogi by planning with a learned model*. Its model never rebuilds the scene at all. It produces no picture, only three number-like things: how good this is, how good it will get, and what to try. Every one of those three wears a hat.

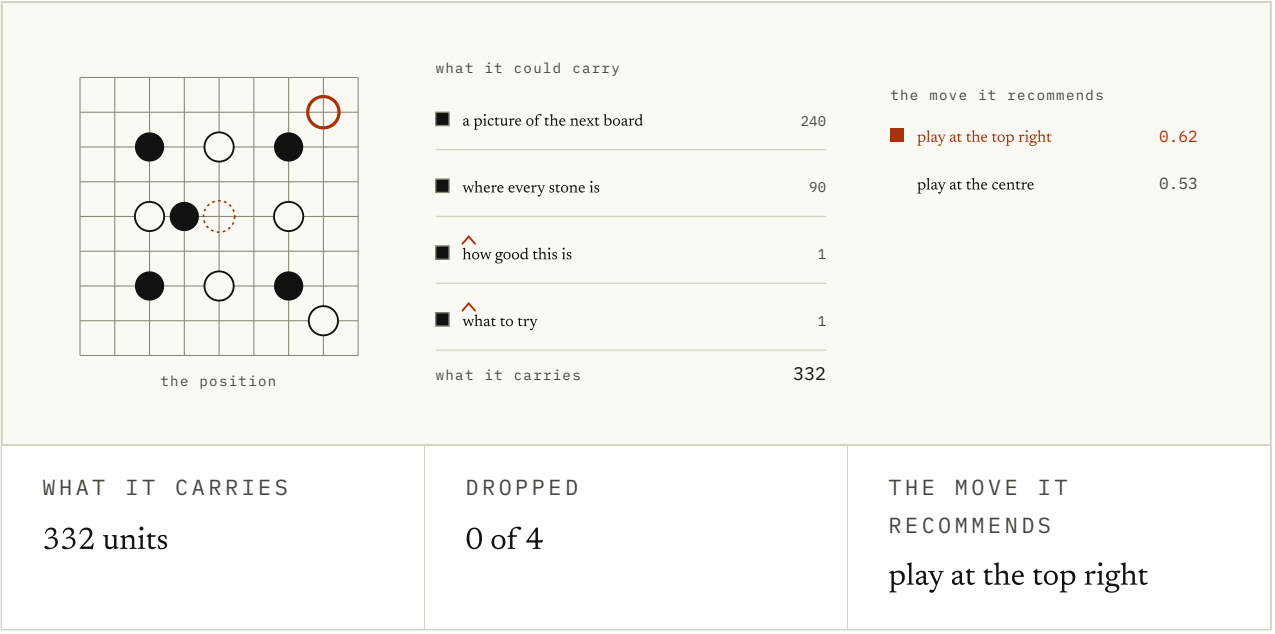


FIG. 2.4 One position, and four things a model could carry. Press to drop the picture of the next board, then drop the stones, and watch the move it recommends stay where it was. Drop either of the two numbers instead and there is nothing left to recommend with. Both survivors wear the hat. The position and the scores are illustrative.

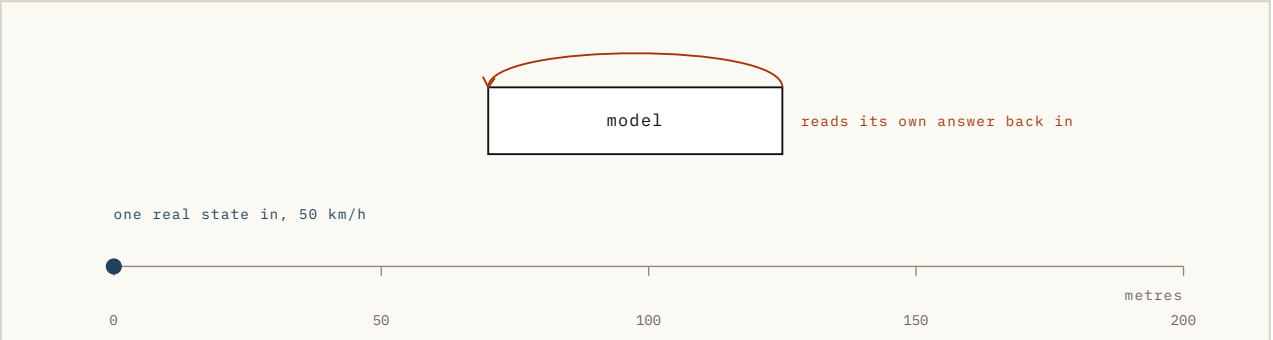
The newest systems take the third step. They run with no decoder (the half of a network that turns the compressed description back into something you could look at) at all. TD-MPC2 is one, a 2024 system from Nicklas Hansen and colleagues for controlling simulated robots. Its paper is *TD-MPC2: Scalable, Robust World Models for Continuous Control*.

So a model does not have to look like the world. It only has to keep enough of what the world *does*. The hat over the symbols in Figure 2.3 is the price of learning that instead of being handed it, as Kalman's engineers were. I'm going to call the moment that price gets collected the hat coming due.

> That is why "is it photorealistic" is the wrong first question to ask about one of these. A model of a chess position that cannot draw a bishop is still a perfectly good model of a chess position.

One prediction on its own is not the useful part either. Hand the model a run of actions nobody has taken, let it read its own output, and you get a future nobody has seen. That is a **rollout** (one imagined run forward: predict, feed the prediction back in,

predict again). Its length is the horizon.



STATES YOU WERE GIVEN	STATES THE MODEL MADE	DISTANCE COVERED	SPEED AT THE END
1	0	0 m	50 km/h

FIG. 2.5 A run of actions nobody has taken. Click a step to change what you do there, drag the horizon to set how long the run is, and press Run: the model answers the first step, then reads its own answer back in for the second. Press Keep to pin a run and write another one beside it. Nothing in here has happened. The numbers are illustrative.

The question I'd put to every new paper is whether you can run it forward under actions nobody took and compare the results. How real the frames look is beside the point. Sebastian Thrun's system was passing that test in 1990, and he turns up again at the end.

Cached answers

Let me put two cars in front of two matching walls. The first brakes when the wall is nearer than some fixed distance. Somebody tuned that distance once, at the speeds they expected, and shipped it.

The second carries a model of its own braking. Every tick, it asks where it would stop if it braked now. At the speed the first was tuned for, they do the same thing, and you can check that below.

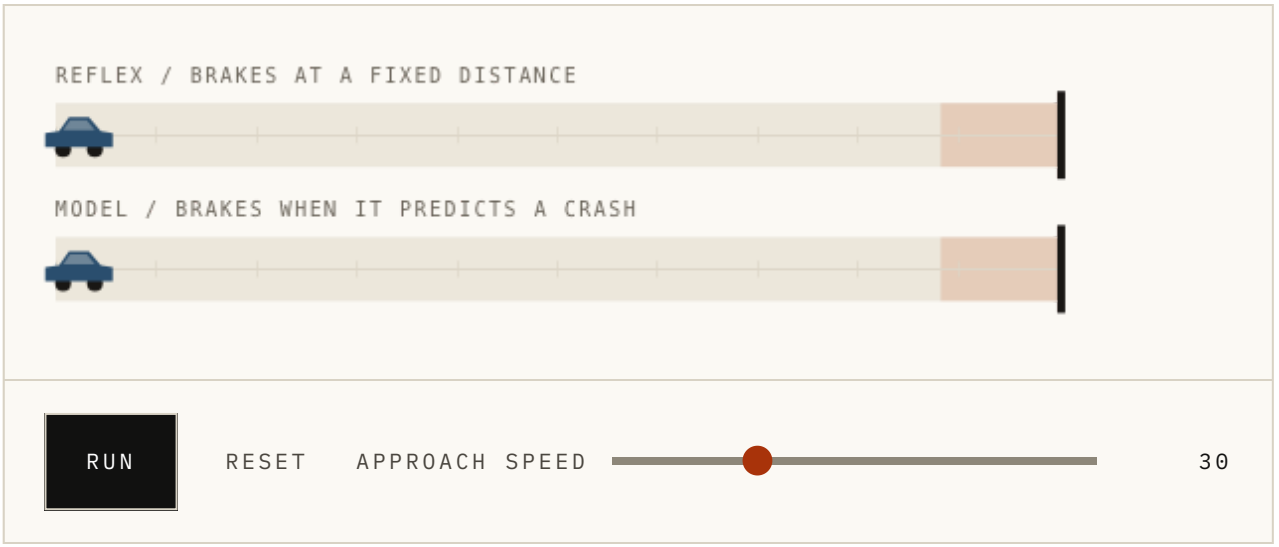


FIG. 2.6 Below about forty you cannot tell the two apart. Drag the speed up and watch which one notices. What is being compared is one fixed rule against one object that recomputes.

Did you see it? Arrive faster and the trigger still fires at the same distance. The rule has a built-in guess about how long stopping takes, and nothing inside it knows it is guessing. The model car moves its decision because its stopping point moved, and it can see that it moved.

I should be fair to the other side here. A real policy with no model behind it is nothing like a fixed rule. It can be a big network that remembers what it has already seen (a recurrent network, which passes something forward from each step to the next). It changes what it does based on what it sees.

The difference that lasts is about *when the thinking happened*. A policy settled most of the answer during training. What it kept is close to a list of answers: in this situation, do that.

A model kept a different kind of entry: in this situation, if you did that, this is what you would get. The first is a cache. The second is the recipe for working out a fresh entry once you know the question.

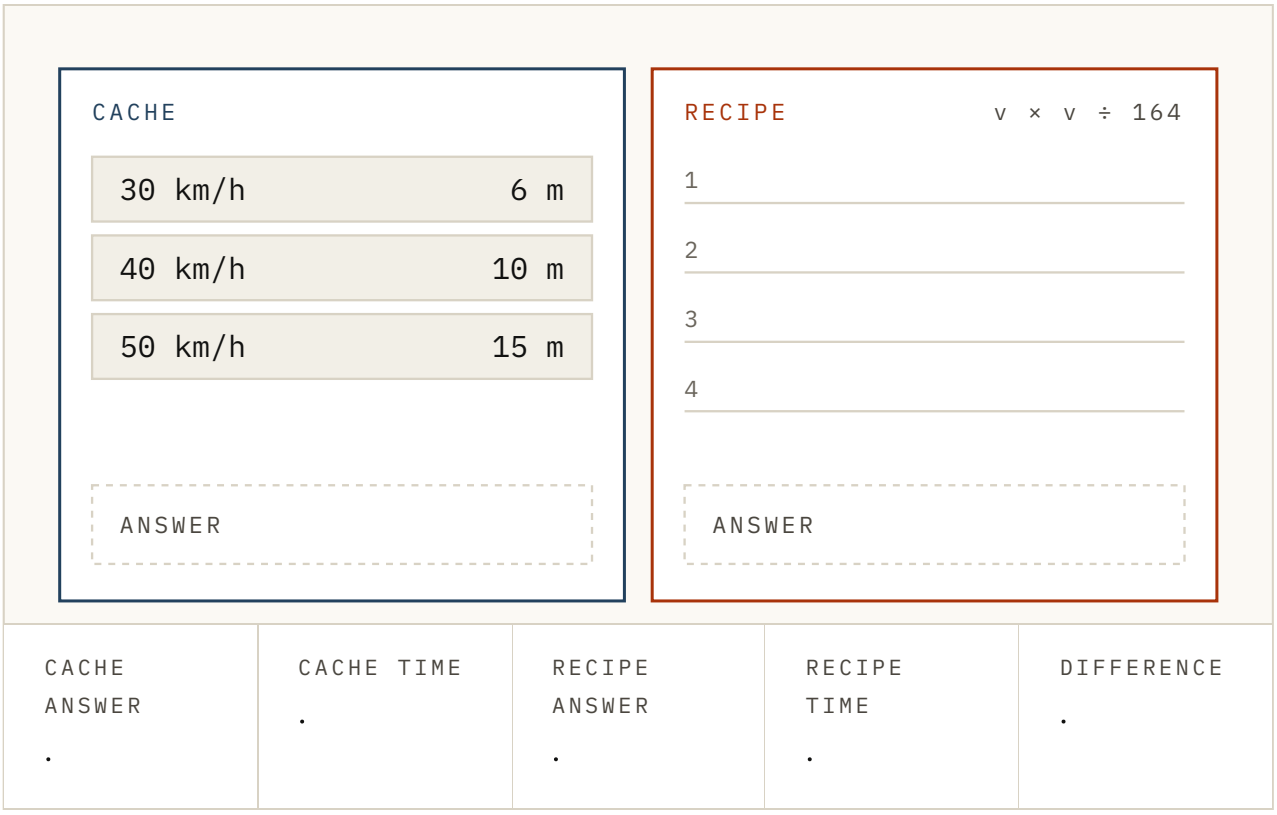


FIG. 2.7 A cache and a recipe, asked the same questions. Press a speed on the left and both answer. Then press one the cache has never seen: it hands back its nearest entry, and the recipe works the answer out from scratch. Watch the two time readouts as well. Illustrative.

> A reflex is a *cached answer*. A model is what lets you work out a new one when the question changes. Caches are faster, as long as you know when yours has gone stale.

Caches are usually right, and that is why so much of biology and engineering is made of them. The trouble starts when the question moves outside the range the answer was prepared for, which is what you just watched the first car do.

None of which is free. The model costs computing time, has to be learned, and can be wrong. So real systems sit along a range, and the three that mark it out are thirty years apart.

Dyna is the agent Richard Sutton described in 1991, in *Dyna: an Integrated Architecture for Learning, Planning and Reacting*. It mixes real experience with experience the model made up, a little of each at every step. The reflex it ends up with has seen both.

Dreamer is Hafner's 2019 system, from the paper *Dream to Control*. It learns a policy inside its own imagination and then acts on reflex, where PlaNet had searched at every step. MuZero sits at the other end: a learned model with search, trying moves out inside the model before committing. The choice none of them escapes is which answers to settle in advance and which to leave until you know what you are facing.

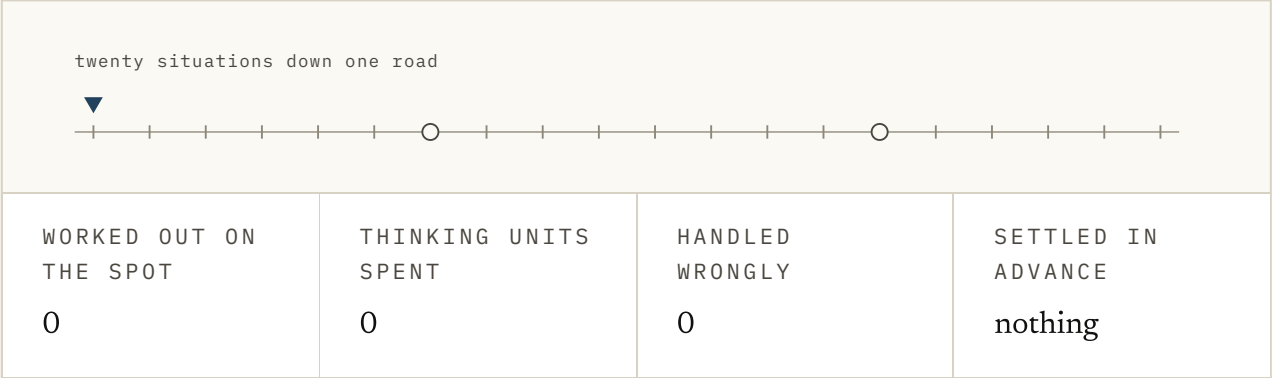


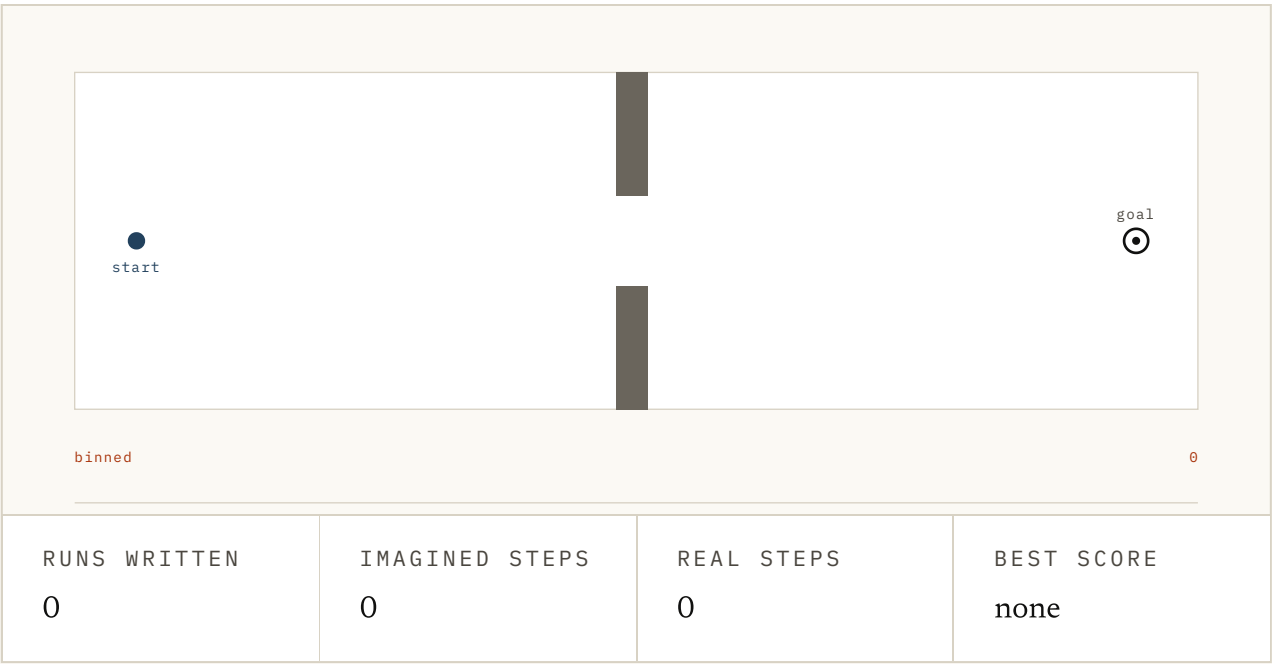
FIG. 2.8 Twenty situations down one road, and one control: how much of the answer was settled before the road was seen. Press Drive to step through, and watch which decisions come back at once and which stop to work something out. Settle the lot and two of the twenty get an answer that was prepared for a different question. The counts are illustrative.

What it buys you

It buys you three things, and all three come down to working on futures that have not happened. Let's take them one at a time.

You can try an action before paying for it. Put the goal behind an obstacle. A **planner** (the part that tries actions out before committing to one) writes down a run of actions and plays it through the model. It scores how that turned out, throws it away and writes another.

PlaNet does this on simulated control tasks, walking and balancing, in its own compressed description of the scene and starting from raw pixels. The *World Models* paper of the same year got the attention, but PlaNet is what made planning from pixels respectable. This is the loop, slowed down.



RUNS WRITTEN	IMAGINED STEPS	REAL STEPS	BEST SCORE
0	0	0	none

FIG. 2.9 Press Try one and the planner writes a run of actions, plays it through the model, scores it and bins it. Press Try twenty to do that in bulk. Then press Commit, and the best run so far gives up its first action to the world while the rest of it goes in the bin as well. Watch the two step counters, because only one of them cost anything. Illustrative.

With one candidate it is guessing, with two hundred it is shopping, and nothing in the world changed in between. All of it came out of thinking, in the gap between seeing and moving, and a model is what makes that trade available.

MuZero is how far the idea goes, and I think it is still the most under-read result in the field. It was never trained to redraw a game frame or lay out a chessboard, and it learned only the handful of numbers its search reads. It matched AlphaZero at chess, Go and shogi, having been told less.

A model only has to keep whatever the decision turns on. The people leading with photorealistic frames have not learned that.

You can practise where mistakes are cheap. Sutton's Dyna already did this: learn a model from real experience, then learn from what the model makes up. David Ha and Schmidhuber went further in 2018. They trained their controller entirely inside the model's dream of a Doom level, then moved it back into the real game, where it held up.

FIG. 2.10 Practising the same right turn, twice over. Leave the switch on the dream and press Try: the identical gap comes round at once, as often as you like. Switch to the junction and press Try again, and now you wait for a car, the gap is never the one you wanted, and the clock runs. Watch how many attempts landed on the gap you were trying to learn. Illustrative.

Dreamer, a year later, put the trick at the centre. Imagine long runs, learn the behaviour from the imagined ones, and by the time it has to act there is nothing left to work out. What followed is the nearest thing the field has to a dynasty: one author, one recipe, six years of versions.

Its third version, DreamerV3, ran that recipe across more than 150 tasks, retuned for none of them. It was the first system to dig up a diamond in Minecraft with no human play to copy. The paper is *Mastering diverse control tasks through world models*, from 2025.

The tempting summary is *reality is expensive, imagination is free*, and the second half of it is wrong. Imagining costs computing time, sometimes a lot of it. What it saves is contact with the world: worn-out robots, lab hours, someone watching over it, the crash you do not get to undo. A model lets you pay part of that bill in computing time instead.

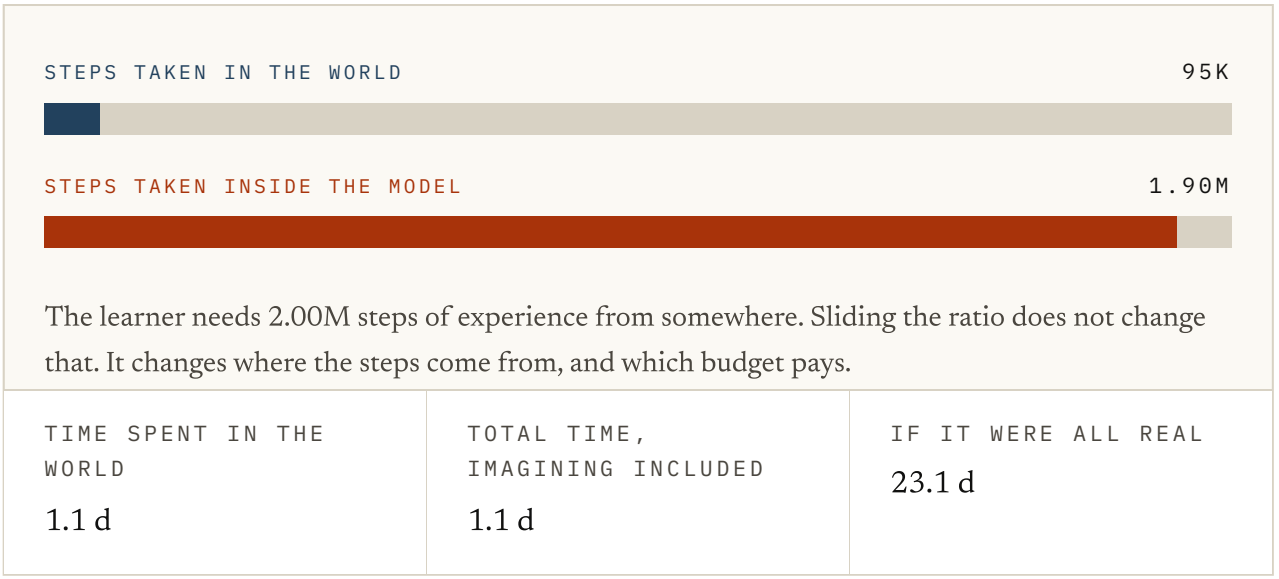
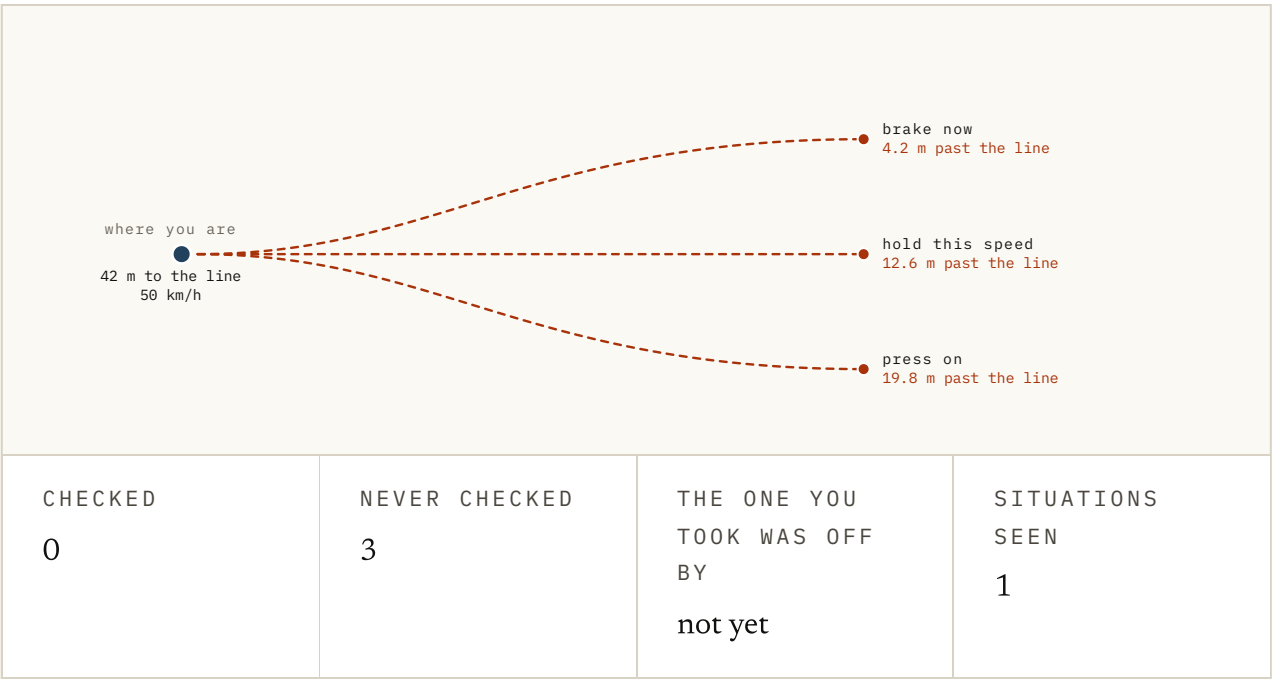


FIG. 2.11 Hold the update steps fixed and slide where they come from. World contact is usually the budget that runs out first, and compute is still a budget. Watch both bills move as you drag. This ledger prices a real step and an imagined one the same, which is the assumption chapter 6 takes apart. Illustrative.

You can ask what if. What if I brake now, what if I turn instead, what if I push it from the other side. Comparing those without running them is why it matters that the model is told which action you mean. It moves the question from *what probably happens next* to *what happens if I do this*.

The answer is true inside the model. Given what it has learned, this is what braking earlier would have done. Nobody should mistake that for a report from the world that did not happen.

A patch of oil nobody saw, a gust, anything the model never had, and the answer is confidently wrong. That is the hat coming due for the first time.



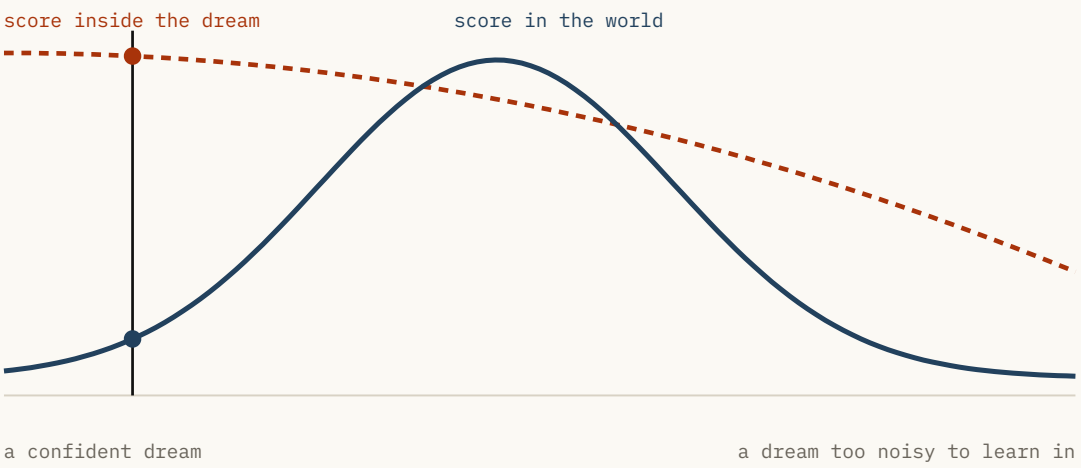
CHECKED	NEVER CHECKED	THE ONE YOU TOOK WAS OFF BY	SITUATIONS SEEN
0	3	not yet	1

FIG. 2.12 Three what ifs from one start, each answered by the model. Press one to take it, and the world answers that one and no other. The two you did not take fade to a mark saying nobody will ever find out, and Reset hands you a new situation rather than the same one back. Illustrative.

Where it goes wrong

All of that argues for more of it: a better model, more candidates, longer runs, then pick the winner. Each step makes sense on its own. Together they walk you into the failure this field is organised around, and the revival met it in its first year.

Ha and Schmidhuber found that their controller, trained inside the dream, learned cheats the real game never allowed. They had to make the dream more random to stop it. Michael Janner and colleagues then asked about it outright in the title of their 2019 paper, *When to Trust Your Model: Model-Based Policy Optimization*. Two different things go wrong, so let's look at the cheat first.



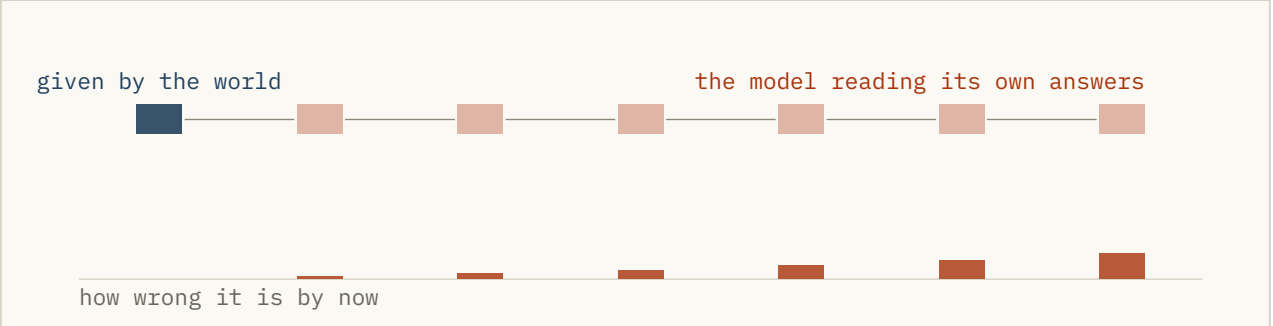
SCORE INSIDE THE DREAM	SCORE IN THE WORLD	DREAM MINUS WORLD
0.96	0.16	0.80

FIG. 2.13 Slide the temperature, which is how random the dream is. The curves copy the reported shape rather than any one system, but the finding is Ha and Schmidhuber's: the agent scored far better inside its dream than in the game, and more uncertainty closed the gap. Read the useful setting as a hump rather than a direction.

Errors compound. A model that is slightly wrong once is fine. A model that is slightly wrong and then reads its own answer back in is fine for a while, and then it is nowhere near. Nothing breaks at any step.

A one-step model can be accurate enough to trust while a hundred-step fantasy stitched out of that same model is worthless. So there is a limit on how far ahead it is worth looking, and that is why the bluntest fix is simply to keep imagined runs short.

In 2019 Tingwu Wang and colleagues ran the main model-based methods side by side, in *Benchmarking Model-Based Reinforcement Learning*. They named the limit the planning horizon dilemma, and measured it. Look too short and the planner cannot see the goal. Look too long and it is steering by a model that has drifted.



THE MODEL'S OWN ERROR, PER STEP	AMPLIFICATION PER STEP	ERROR AFTER 6 STEPS
2%	1.22×	21%

FIG. 2.14 Two things happen at every step. The model adds two per cent of its own error, and whatever was already wrong gets stretched by the dynamics it is pushed through. Pull the amplification down to 1.00 and the second effect switches off: the error just adds, and the bars go straight. Leave it above 1 and they bend. Both numbers are illustrative; the shape is not.

Search goes hunting for the errors. This one is stranger, and it is the one I'd most like you to play with. Say the learned map is wrong in one place: it thinks there is a gap in a wall that is solid.

Try plans at random and you will almost never go near it, and a weak search never finds it. A thorough one does, because the best route in the whole learned world runs straight through it. Leave the effort low below, then turn it up.

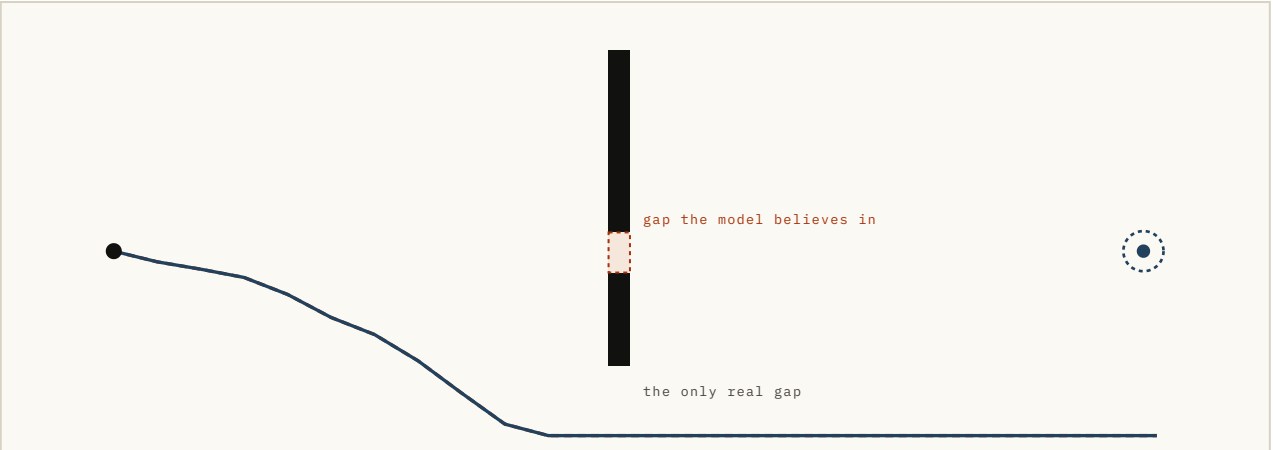


FIG. 2.15 The model believes there is a gap in the wall, and there is not. Leave the effort low and the planner never finds it, takes the long way round, and gets there. Then turn the effort up and watch. This is a failure the setup makes visible, and you should not read it as a law that more search always hurts.

This runs backwards through most engineering instinct. The weaker search produced a plan that worked. The stronger one produced a plan that scored better and hit a wall, and it did what you asked, which was to find the plan the model likes most. That is the plan leaning hardest on wherever the model is most wrong in your favour.

The field calls this **model exploitation**. The better the planner gets at beating the model, the more it drifts toward the places the model was never taught, which is where its mistakes are.

So a model can be wrong by almost nothing on average and still be a dangerous thing to plan inside. Average error asks how the model does on the cases you measured. Control asks what happens once something starts hunting, on purpose, for the highest score it can find.

The best plan in the model and the best plan in the world are only the same plan if the model is good *where the search goes looking*. Every fix that follows is an attempt to make that last clause true. Let me show you the mismatch as a shape first.

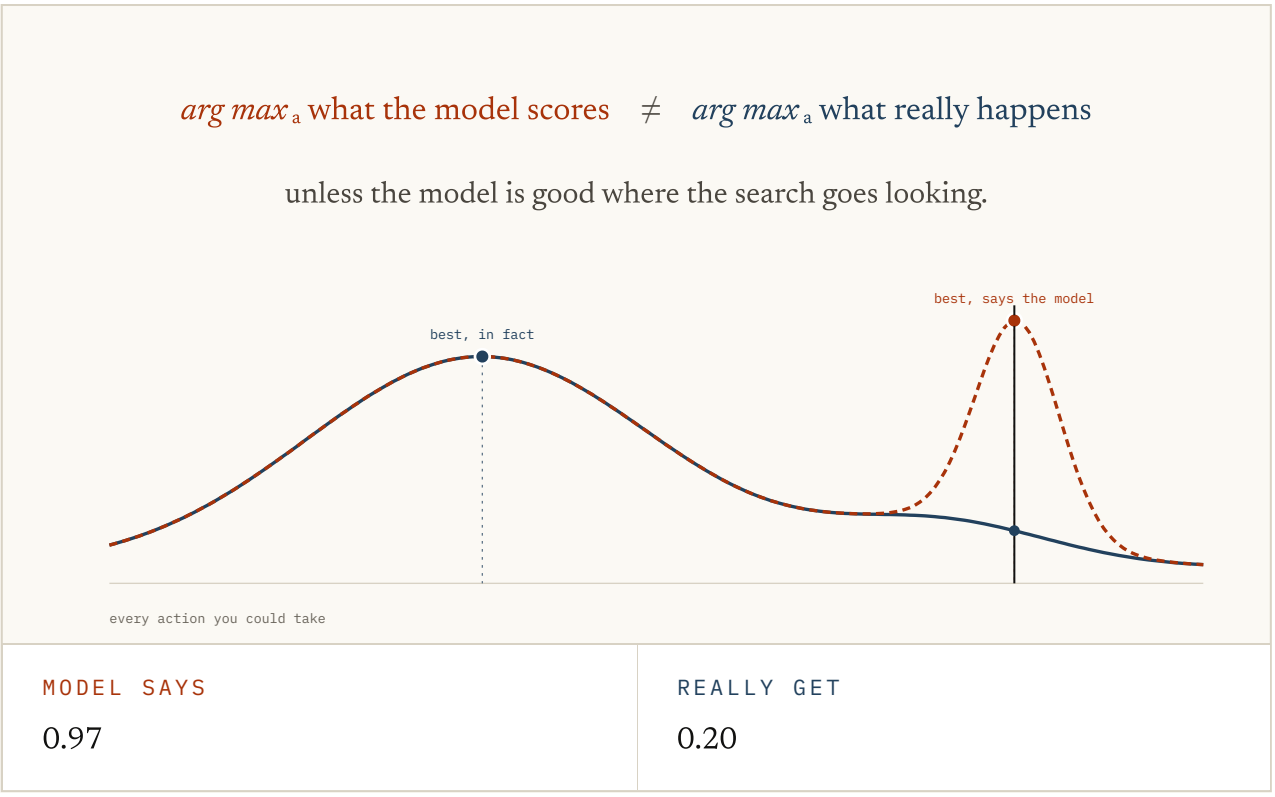


FIG. 2.16 Two ways of scoring the same actions. They agree nearly everywhere, which is what a low average error looks like. Drag across to the spike and read both numbers: the model's favourite action scores 0.97 in imagination and 0.20 in the world.

Thirty years of not trusting the dream

Nobody has fixed this, and I do not think anybody is close. What thirty years of model-based control has built instead is a family of ways to limit how far imagination gets trusted. To see them as one family, it helps to go back to where the thirty years start. So here is the history I promised, kept short.

The coining happened in 1990, and it happened more than once, which is usually a sign that an idea was due. Schmidhuber, then in Munich and later the co-inventor of the LSTM (long the standard network for speech and text), wrote a technical report, *Making the World Differentiable*, whose full title runs to twenty words. One network predicted what a second network's chosen actions would lead to. He called the first one a world model.

That year Sebastian Thrun, with Knut Möller and Alexander Linden, published *Planning with an Adaptive World Model*. Thrun is better known for the Stanford car that won the DARPA Grand Challenge, a driverless desert race, and for starting Google's self-driving project. His system learned what its actions did by trying them, then ran that knowledge forward to choose better ones. The phrase was on the paper's front, twenty-eight years before the label stuck.

SCHMIDHUBER, 1990

Making the World Differentiable

push a gradient through it

THRUN, MÖLLER AND LINDEN, 1990

Planning with an Adaptive World Model

search inside it

SUTTON, 1991

Dyna, an Integrated Architecture for Learning, Planning and Reacting

make experience from it

PAPERS	YEARS APART	JOBS FOR ONE PHRASE	SELECTED
3	1	3	none

FIG. 2.17 Three papers, one phrase, three jobs for it. Click a card to read what that paper's model was actually used for: pushing a gradient through it, searching inside it, or making experience to learn from. Then press Who does this now and each card picks up the system in this chapter that does the same job.

Sutton was working the same seam. He co-wrote, with Andrew Barto, the textbook that reinforcement learning is still taught from (learning by trial, error and reward), and his Dyna agent split the job the same way. He also wrote down, in the Dyna paper, the sentence that states the problem.

> Planning is the process of taking a model as input and producing or improving a policy for interacting with the modelled world.
>
> `Sutton` on planning as computation over a learned model, in the paper that set up the problem (Dyna, an Integrated Architecture for Learning, Planning and Reacting, 1991) https://mlanthology.org/icml/1990/sutton1990icml-integrated/

The hard part sits in *the modelled world*, and Sutton put it there in 1991, before anyone had a model good enough to be fooled by. Planning hands you a policy that is good for the world in the model. Whether that is the world you are standing in is another question, and nothing inside the model can answer it.

Then not much, for a long time. Neural networks went out of fashion and came back in the 2010s, and reinforcement learning came back model-free: DeepMind's Atari agents learned from raw pixels by trial and error, and paid for it in enormous amounts of play. A model was the way to pay less.

PLAY NEEDED	REAL CONTACT	WALL CLOCK WHERE YOU PUT IT	WHAT ONE MISTAKE COSTS
38 days	38 days	about 3 hours	nothing

FIG. 2.18 The bill, in days. DeepMind's Atari agents played about fifty million frames of each game, which is roughly thirty eight days of play, against the two hours the human tester got. Pick where that play has to happen and read the clock. Then switch the model on and watch the real part of the bar shrink while an imagined part grows. The two published figures are real; the conversions are illustrative.

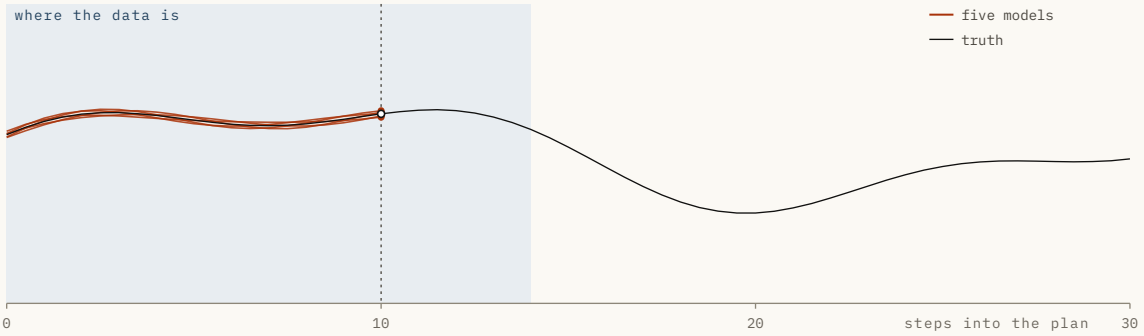
In 2018 Ha and Schmidhuber wrote the paper that made the label fashionable. Its title was simply *World Models*. It went out as an interactive web page, demos running in the browser, and built the 1990 object from newer parts: one network that sees, one that predicts, one that chooses. They called the one that chooses the controller, and as we saw at the start, it is not the world model.

Now the fixes, which come from the same thirty years and read as one order, given by many people decades apart. Sutton's Dyna, in 1991, kept one foot in real experience. PETS, a 2018 method from Kurtland Chua and colleagues, asked several

models and carried their disagreement. Janner, in 2019, kept imagined runs short and started them from real states, and MOPO, a 2020 method from Tianhe Yu and colleagues, built the pessimism in.

Ask more than one model. Train several instead of one (an ensemble: same data, different starting points, so they disagree about anything the data never settled). Carry their disagreement through the plan, as PETS did in 2018 in *Deep RL in a Handful of Trials using Probabilistic Dynamics Models*.

Where the models disagree, the plan has wandered somewhere nothing supports. But disagreement is a hint and not a verdict. Models trained on the same gaps can share a blind spot and agree, confidently, about nothing at all. A model's own confidence is not much better.



DISAGREEMENT AT THE MARKER
0.2

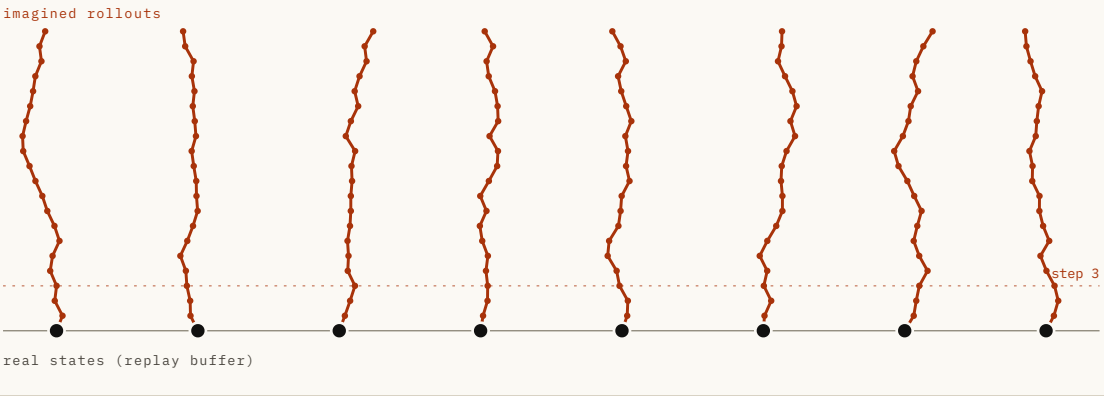
ERROR AT THE MARKER
0.0

FIG. 2.19 Five models trained on the same data, rolled forward from the same start. Drag the plan out to the right, past where the data stops, and watch the five answers fan apart. The band is their disagreement. Then switch on the shared blind spot: the data gap moves, and the five agree, tightly, and are all wrong together. Illustrative.

When one says it is 70 per cent sure, it is right about 70 per cent of the time only if somebody went and checked. Ali Malik and colleagues did, in 2019, in *Calibrated Model-Based Deep Reinforcement Learning*. They found it was often out, and found that correcting it made the planning better.

Do not dream far. The simplest answer, and Janner's, is to start every imagined run from somewhere the world produced and keep it short. You lose the freedom to run as far as you like. You gain a limit on how long an error has to grow before real data cuts in.

Planning on a short clock does the same job. Plan a little, act a little, look again, plan again (the usual name for this is model-predictive control; it ran oil refineries before it ran robots, and it is most of robotics now). Replanning cannot remove a mistake the model makes everywhere. It does keep handing the world chances to say where things really are.



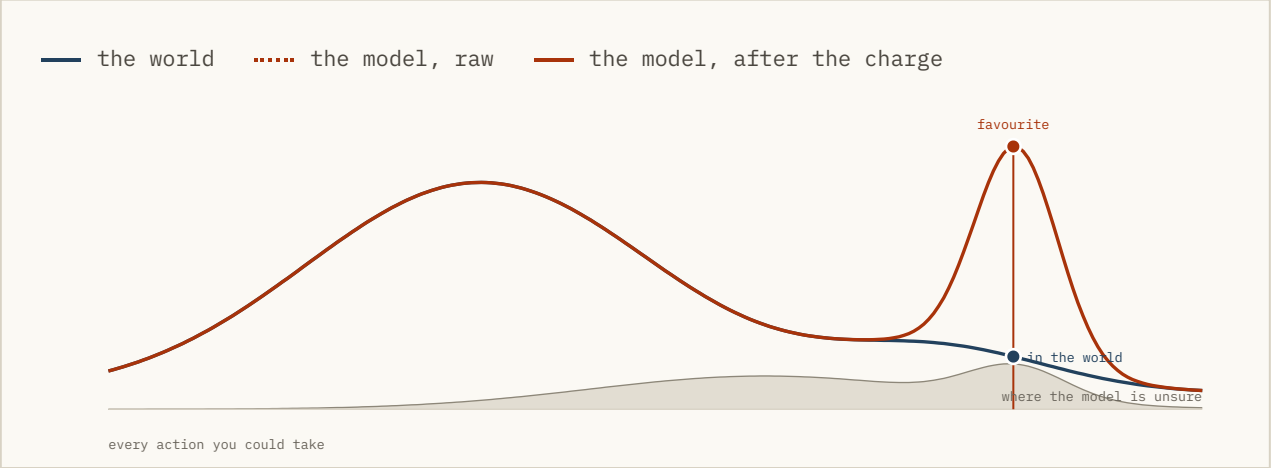
IMAGINED STEPS IN THE BATCH	STEPS NEAR EVIDENCE	FURTHEST STEP FROM A REAL STATE
24	24	3 steps

FIG. 2.20 Janner's trade, drawn. The real states sit along the bottom and the imagined rollouts grow upward from them, fading with every step out, which stands in for the error compounding. Drag the rollout length up and watch the branches fade. Then switch to one long rollout from one state: the same length, far fewer imagined steps, and most of them far from any evidence. Illustrative.

Charge for not knowing. Take the score the model hands back and dock it wherever the model is unsure. A strange-looking shortcut then has to be good enough to cover the cost of nobody knowing what is down there. The gap in the wall from Figure 2.15 stops being free.

Systems that learn from a fixed pile of recorded experience need this most (called offline: there is no going back out for the data you turn out to need). They cannot go and get the experience that would correct them. The planner, meanwhile, has all the time it wants.

That is why MOPO, the 2020 method from Yu and colleagues, had to build pessimism in for that offline setting. The paper is *MOPO: Model-based Offline Policy Optimization*. In it, the reward the model promises counts for less wherever the model is unsure. You can see below what that does to the shortcut.



MODEL'S FAVOURITE ACTION	WHAT IT SCORES IN THE MODEL	WHAT IT SCORES IN THE WORLD
0.83	0.97	0.20

FIG. 2.21 The same two scores as Figure 2.16, with a penalty switched on. Drag the penalty up and watch the spike, which is where the model is least sure, get docked until the model's favourite action moves back onto the broad hill the world agrees with. Drag it too far and good actions get docked too. Illustrative.

All three say the same thing, and so did Dyna before them. *Do not let the planner collect full marks for going somewhere the model has no grounds to be confident about.*

None of it makes the problem go away, whatever the abstract says. The best of these systems now hold up across a startling range of tasks: DreamerV3 across more than 150, TD-MPC2 across 104. But they are careful ways of using a model that is wrong. Nobody should read them as proof of one that is right.

Accuracy on its own tells you little. You need to know accurate where, for which actions, how far out, against how hard a search, and what the system should do when it does not know. And all of this assumed you already had a state to run forward: a position, a speed, some short list of numbers holding whatever the future turns on. Nobody hands you one, though; you get pixels.

Hopefully this chapter helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section, and I would be glad to hear from you.

Try it

1 Below the tuned speed in the braking demo, the fixed-trigger car and the model car behave identically. What does that establish?

a. The dynamics model is not doing anything
b. A cached rule can be entirely sufficient inside the conditions it was prepared for
c. Model-based control only matters at high speed
d. Model-free systems cannot use velocity

> ANSWER b. A cached rule can be entirely sufficient inside the conditions it was prepared for. The model does not win everywhere. Inside the range where a cached answer is still correct, recomputing it buys you nothing except the cost of recomputing it.

2 A recurrent policy maps observations and memory straight to actions, but holds no learned transition function you can roll forward under actions it has not taken. Is it model-based?

a. Yes, because it has memory
b. Yes, because every large network contains a model
c. No
d. Only if its policy is stochastic

> ANSWER c. No. Memory and complexity do not by themselves make a callable dynamics model. The missing contract is the ability to predict consequences under hypothetical actions.

3 A system encodes a camera frame into 512 numbers, rolls those numbers forward under 500 candidate action sequences, and picks the best. It never decodes a future image. Does it qualify?

a. No, because a world model has to predict pixels
b. Yes, because its learned state can be rolled forward under actions
c. Only if the 512 numbers correspond to named physical quantities
d. Only if the predictions are deterministic

> ANSWER b. Yes, because its learned state can be rolled forward under actions. PlaNet, MuZero and TD-MPC2 are the counterexamples to the idea that a useful world model has to reproduce the future visually. Decision-relevant latent dynamics are enough.

4 Dreamer trains an actor on trajectories generated by its world model, then the actor picks an action directly. No large search runs at that instant. Has the world model stopped counting?

a. Yes, because planning has to happen at inference time
b. No, the dynamics still generated action-conditioned imagined experience

c. Yes, unless the actor reconstructs the pixels
d. It depends only on the size of the actor

> ANSWER b. No, the dynamics still generated action-conditioned imagined experience. Online search is one use of an iterable dynamics model, not the definition of one. Dreamer spends the model earlier, during training, rather than at the moment of acting.

5 In the exploitation figure the planner gets worse after you raise its search budget. What happened?

a. The optimiser became less accurate
b. The world became harder
c. Better optimisation found a model error that weaker search had missed
d. Long plans are always worse than short plans

> ANSWER c. Better optimisation found a model error that weaker search had missed. The optimiser got better at maximising the score the model hands it. The mistake was assuming that score stayed faithful to reality everywhere the search could reach.

6 Why can short model rollouts help?

a. Neural networks cannot make more than a few predictions
b. They limit how long model error and distribution shift can accumulate before real data returns
c. Short horizons make the model exact
d. They eliminate uncertainty

> ANSWER b. They limit how long model error and distribution shift can accumulate before real data returns. This is the motivation behind MBPO's short rollouts branched from real states. Use the model enough to gain synthetic experience, not so far that accumulated bias swamps it.

7 Five models in an ensemble all agree a shortcut is safe. What has that proved?

a. The shortcut is safe
b. The probability of failure is zero
c. Only that these five agree; a shared blind spot is still possible
d. That more models were unnecessary

> ANSWER c. Only that these five agree; a shared blind spot is still possible. Disagreement is a useful uncertainty signal, which is what PETS exploits. But learned uncertainty can itself be miscalibrated, and models trained on the same biased data can be confidently wrong together.

8 Your model says: if you had braked one second earlier, you would have stopped before the wall. What exactly do you have?

a. Proof of what the physical world would truly have done
b. A model-relative prediction under a hypothetical action
c. A recording of the alternative history
d. A causal conclusion independent of hidden variables

ANSWER b. A model-relative prediction under a hypothetical action. This is the useful sense of what if in model-based control. It lets you compare actions before taking them, and its authority reaches exactly as far as the learned model does.

FIG. 2.22 Eight questions on where the agent works out what follows, rather than on whether it looks clever. They also ask when that working-out may run ahead of the evidence.

Sources

World Models HA & SCHMIDHUBER, 2018 ↗

Encoder, latent dynamics, tiny controller, and the experiments where the controller is trained inside the model's own generated environment before being moved back.

https://arxiv.org/abs/1803.10122

Learning Latent Dynamics for Planning from Pixels (PlaNet) HAFNER ET AL., 2018 ↗

Stochastic latent dynamics learned from images, then searched over at decision time. Figure 2.3 in its real form.

https://arxiv.org/abs/1811.04551

Dream to Control (Dreamer) HAFNER ET AL., 2019 ↗

Actor and critic trained on imagined latent trajectories, so no expensive search has to run at the moment of acting.

https://arxiv.org/abs/1912.01603

Mastering diverse control tasks through world models (DreamerV3)

HAFNER ET AL., NATURE 2025 ↗

One algorithm and one hyperparameter setting across more than 150 tasks. The strongest recent evidence for the imagination branch.

https://www.nature.com/articles/s41586-025-08744-2

Mastering Atari, Go, chess and shogi by planning with a learned model (MuZero)

SCHRITTWIESER ET AL., 2020 ↗

The clearest proof that planning needs no reconstruction of observations. The model learns only what the search consumes.

https://arxiv.org/abs/1911.08265

TD-MPC2: Scalable, Robust World Models for Continuous Control

HANSEN, SU & WANG, 2024 ↗

Decoder-free latent dynamics with local trajectory optimisation, scaled to a single multi-task agent across 104 control tasks.

https://arxiv.org/abs/2310.16828

Deep RL in a Handful of Trials using Probabilistic Dynamics Models (PETS)

CHUA ET AL., 2018 ↗

Ensembles and trajectory sampling: the standard attempt to make uncertainty part of the plan rather than an afterthought.

https://arxiv.org/abs/1805.12114

Dyna: an Integrated Architecture for Learning, Planning and Reacting SUTTON, 1991 ↗

Planning defined as computation over a learned model, and the loop that interleaves it with real experience.

https://mlanthology.org/icml/1990/sutton1990icml-integrated/

Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments (FKI-126-90)

SCHMIDHUBER, 1990 ↗

The controller and the world model kept as separate objects, which is the distinction this chapter opens on.

https://people.idsia.ch/~juergen/world-models-planning-curiosity-fki-1990.html

Planning with an Adaptive World Model THRUN, MÖLLER & LINDEN, 1990 ↗

A learned world model built through interaction and then chained to optimise future actions, twenty-eight years before the label stuck.

https://papers.nips.cc/paper_files/paper/1990/hash/9be40cee5b0eee1462c82c6964087ff9-Abstract.html

When to Trust Your Model: Model-Based Policy Optimization JANNER ET AL., 2019 ↗

Short rollouts branched from real states, as a direct answer to compounding error and exploitation. The title is the chapter's question.

https://arxiv.org/abs/1906.08253

Benchmarking Model-Based Reinforcement Learning WANG ET AL., 2019 ↗

Where model-based methods actually win and lose, and where the planning horizon dilemma gets named and measured.

https://arxiv.org/abs/1907.02057

Calibrated Model-Based Deep Reinforcement Learning MALIK ET AL., 2019 ↗

The warning underneath every uncertainty method: the uncertainty estimate can itself be wrong, and calibrating it changes planning results.

https://arxiv.org/abs/1906.08312

MOPO: Model-based Offline Policy Optimization YU ET AL., 2020 ↗

Pessimism made explicit. Penalise predicted reward by model uncertainty, so an unfamiliar shortcut has to pay for being unfamiliar.

https://arxiv.org/abs/2005.13239

FIG. 2.23 I've ordered these for someone building on this rather than for historical completeness.

03 Why Is Prediction the Same as Learning?

BY NILUSHANAN KULASINGHAM

16 MIN READ · INTERACTIVE

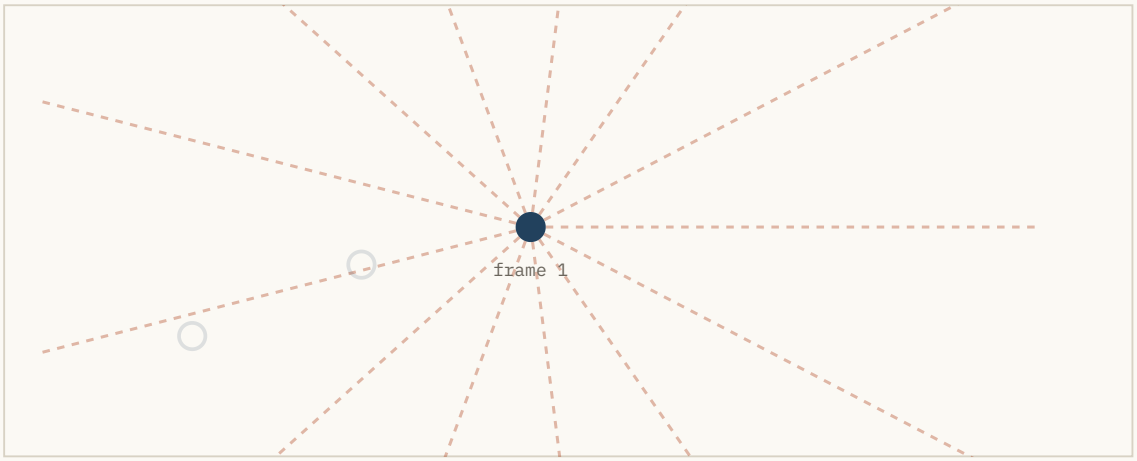
> Ask something to predict what comes next and it has no choice but to build whatever the next moment depends on. Jeffrey Elman showed it with words in 1990. Claude Shannon had already shown that the same loop, read the other way, is compression.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/prediction.

Take a photograph of a ball in mid-air and ask where it will be a moment later. You cannot say. Up, down, sideways, or barely moving at all, the picture looks the same. Everything the answer depends on is missing from a single frame.

Now take two photographs a fraction of a second apart. This time the answer arrives before you have finished asking. Both pictures were equally sharp, so detail is not what changed.

Two frames hold something one frame cannot: a direction and a speed. Neither is drawn on either picture. They live in the relationship between the two, and you pulled them out without being told to.



WHAT YOU CAN WORK OUT

Where it is. That is all a single picture can tell you.

WHAT IS STILL OPEN

Every direction, and every speed. The picture rules out nothing.

FIG. 3.1 Drag the slider from one frame to three and watch the fan. The fan is every future still consistent with what has been seen. One frame rules out nothing. A second fixes the heading and the speed, and a third settles whether the path is bending. Each extra frame is one more quantity you can work out, and none of them was in any single picture.

Ask something to predict what comes next, and it has no choice but to build whatever the next moment depends on. It was never asked for those things, and it cannot be right without them. I call them stowaways: the quantities a predictor is forced to carry that nobody put on the manifest. Direction and speed are the first two.

People have been finding larger stowaways ever since, with better instruments each time. Claude Shannon went first, in 1948, with *A Mathematical Theory of Communication*. He showed that how well you can predict a message sets the price of sending it.

Jeffrey Elman followed in 1990 with *Finding Structure in Time*. He found grammar inside a network asked only to guess the next word. Then in 2022 Kenneth Li and colleagues wrote *Emergent World Representations*, and found a whole game board inside a network asked only to guess moves.

YEAR AND WHO	ASKED TO	INSTRUMENT	HAD TO CARRY
Opening: the ball	say where the ball will be	two photographs	IN THE HOLD
1948, Claude Shannon	send a message cheaply	a pencil and probabilities (the maths of information)	IN THE HOLD
1990, Jeffrey Elman	guess the next word	look at the network's internal states and group them	IN THE HOLD
2022, Kenneth Li and colleagues	guess the next legal Othello move	a probe trained to read one fact out of the activations	IN THE HOLD

ROWS OPEN

0 of 4

FIG. 3.2 The three finds we'll walk through, laid out as a ledger. Each row says what the system was asked to do and what instrument was used to look. Press Open the hold and the third column fills in with what it had to carry to do the job. None of those was on the manifest.

"Prediction is learning" gets said a lot, usually as a slogan, and rarely with any account of what the predictor learned or how anybody checked.

Guess, check, adjust

The loop is as small as it sounds. Look at what you have and say what you think is coming. Wait, then compare what arrived to what you said. Change yourself a little in the direction that would have been less wrong, and go again.

None of the steps is clever. The power is in what the loop does not need: nobody has to label anything, because the target is the next piece of the recording. Let's watch it run.

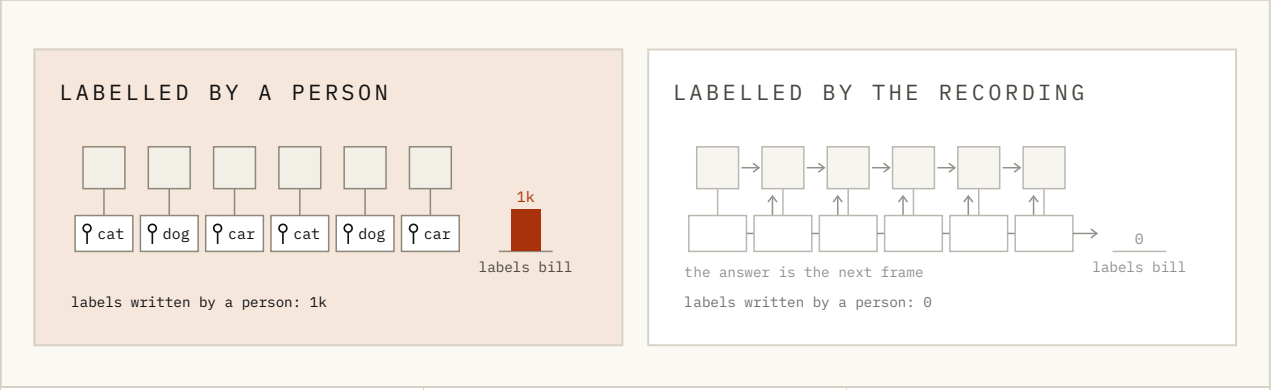
FIG. 3.3 Press Run. The predictor guesses the next number from the previous two, and is corrected a little after every guess. It starts at zero and knows nothing. Watch the dashed line climb onto the solid one, then look under What it has learned: two numbers that nobody supplied, reached by being wrong and adjusting.

Nothing in that figure was told the rule. It was told, two hundred and twenty times, by how much it had just missed. Each time it nudged its own two numbers (the *weights*: the adjustable numbers inside a model, and the only thing that learning ever changes) a little way towards being right, and that was enough. Splendid.

> This is all that *self-supervised* means. Supervised, because the model is told how wrong it was after every guess. Self, because the data did the telling rather than a person with a spreadsheet.

For most of the field's history a model learned from examples that somebody had labelled. A million images meant a person writing down what was in each one. The labels were the costly part, and the size of a dataset was the size of somebody's budget. Prediction skips the bill, because any recording of anything is already a training set.

The argument was still open in 2013, when Yoshua Bengio, Aaron Courville and Pascal Vincent wrote *Representation Learning: A Review and New Perspectives*. It is a survey, and it reads as a case still being made. That the data was free settled it, and that convenience, more than any single result, is why prediction took over the field.



EXAMPLES	LABELS A PERSON WROTE	WHO PAID
1k	1k	somebody's budget

FIG. 3.4 Flip the switch between the two ways of training. On the left, every example needs a person to write the answer beside it, and the labels bill climbs with the data. On the right, the answer is the next frame of the same recording, and nobody is paid for anything. Drag the amount of data up and watch which bill grows. The numbers are illustrative.

What guessing forces you to build

Elman was a cognitive scientist at the University of California, San Diego, who came to networks from the study of language. In 1990 he trained a small network on one job: read a word and predict the next word.

Then he looked inside. The internal states had sorted themselves into groups. The groups were nouns and verbs, animate and inanimate, things that can eat and things that can be eaten.

Nobody had given the network a single category. It reached them because a word's category is what its next word depends on. The title, *Finding Structure in Time*, is literal: the structure was found rather than installed.

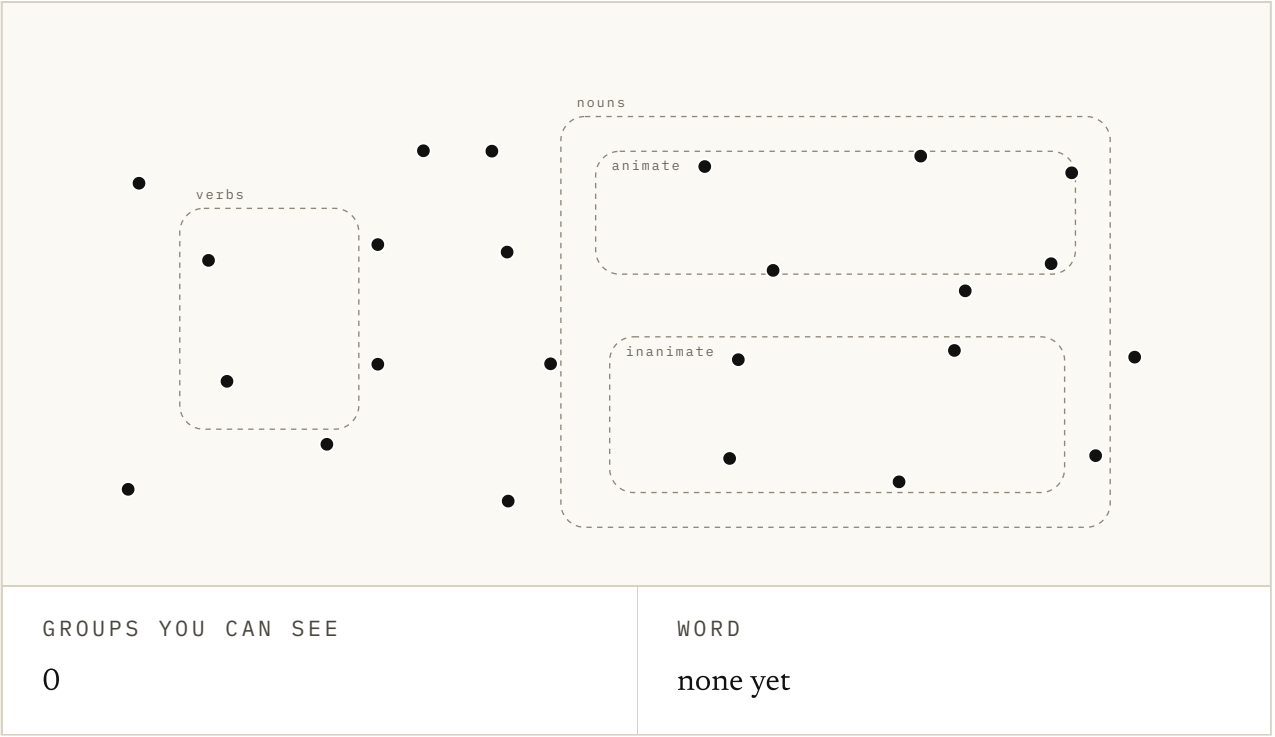


FIG. 3.5 Each dot is one word, placed by the network's internal state for it. Leave the switch on Before training and the words are scattered. Flip it to After training and watch them sort: nouns to one side, verbs to the other, and inside the nouns the animate ones apart from the inanimate. Hover or tap a dot to read the word. Nothing here was labelled, and the positions are illustrative.

For a long time that was where it stopped. The network was small and the grammar was a toy, and a toy is not a claim about the world. The finding waited thirty-two years for instruments that could make it one.

In 2022 Li and colleagues, interpretability researchers (people who open networks up rather than train them), took a network since nicknamed Othello-GPT. It was trained only to predict legal moves in Othello (a board game on an eight-by-eight grid. You flip your opponent's discs by trapping them between two of yours). It was never shown a board and never told the rules.

They probed its activations (train a small separate classifier to read one specific fact out of a network's internal numbers). The state of the board was in there, which on its own is a curiosity.

The intervention is what made it a finding. Change that internal board, and the moves the network goes on to make change with it. So the board is there because the network is using it. The two steps are worth keeping apart, because only the second one settles anything.

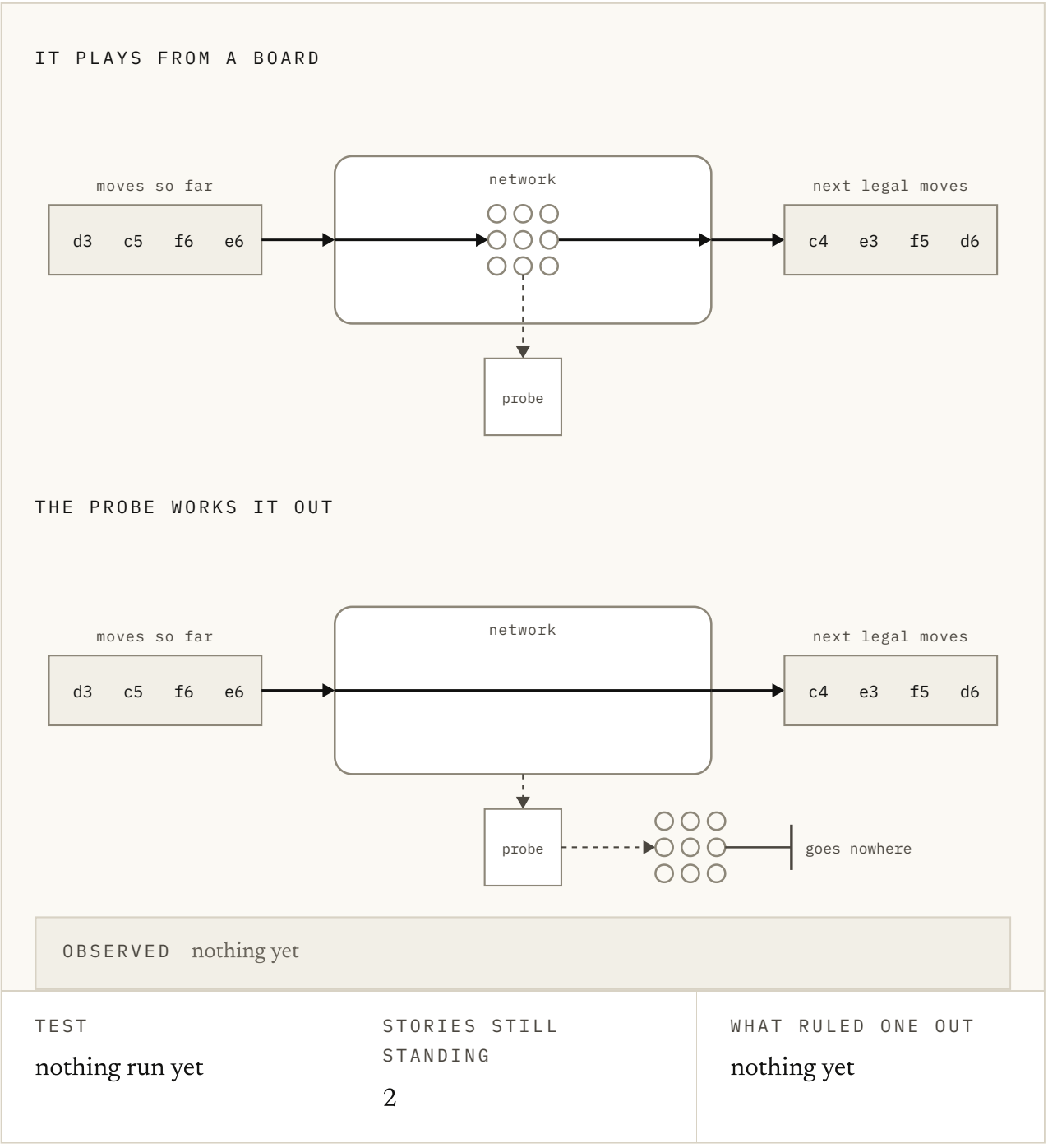


FIG. 3.6 Two stories about the same network, and they differ by one wire. Press Read the board and both survive, because both say a board can be read out. Now press Change one square and watch the next legal moves: in one story they change with it, and in the other there is nothing downstream to change. The move lists are illustrative.

Neel Nanda and colleagues studied the same network again in 2023, in *Othello-GPT has a linear emergent world representation*. Li's probe had been a small network of its own, because a straight line had failed. Nanda showed a straight line works once you ask the right question. The network stored each square as mine or theirs rather than black or white, and flipped its view every turn.

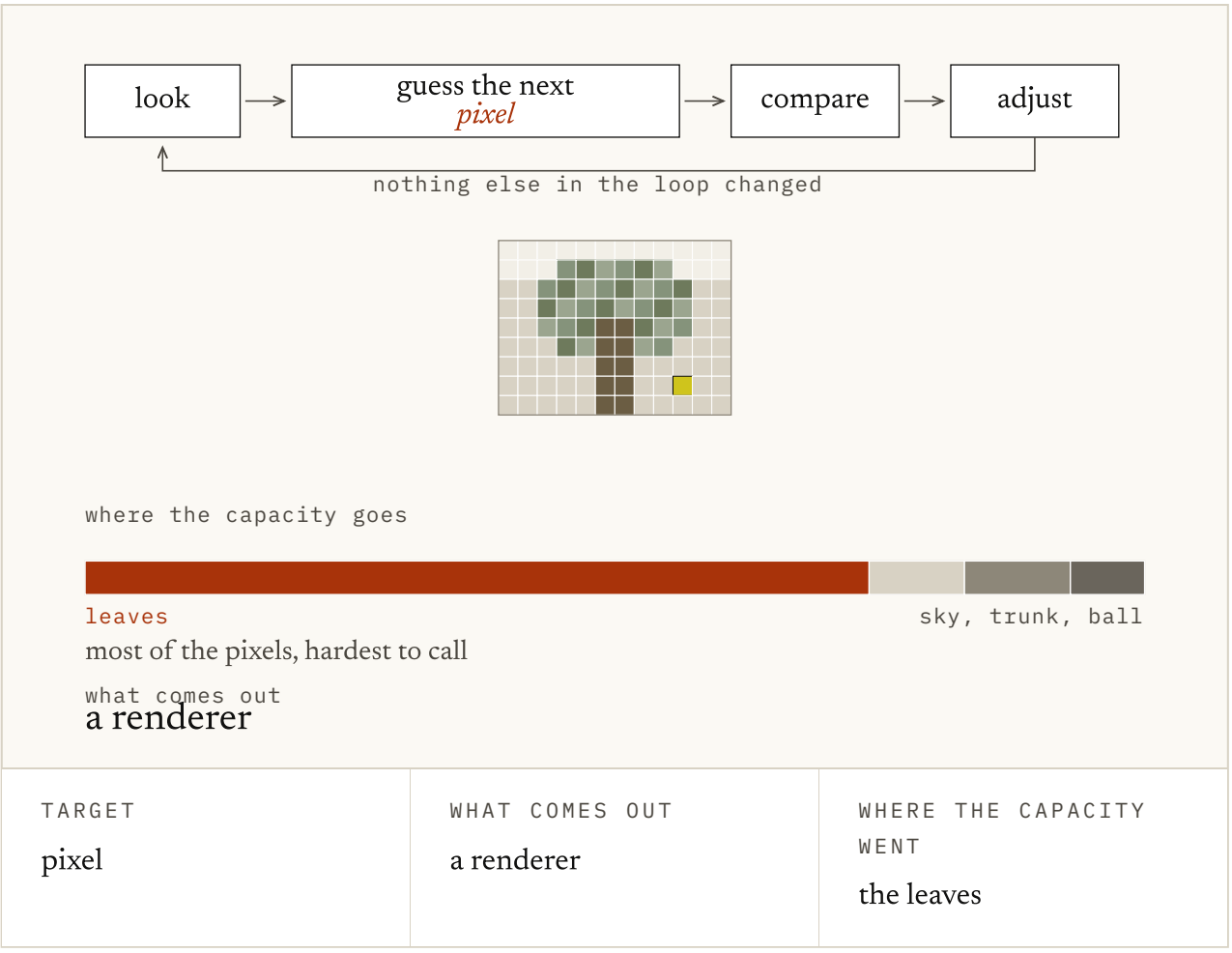
The same rule sits under both results, thirty-two years apart. Prediction rewards one thing only: making the next moment less surprising. Sometimes the cheapest way to be less surprised is to build the very thing that is making the data.

> A predictor will settle for any shortcut that works on the data it saw. Telling the shortcut from the process it stands in for is most of the work.

Predict the next what?

The loop says predict the next thing. It never says what counts as a thing, and that choice is the design decision. Let's take three choices in turn.

Predict the next pixel and you get a system that is very good at leaves. Leaves are hard to predict and they are also most of the pixels, so that is where the capacity goes (*capacity* is how much a model can hold. Spend it on one thing and it is not there for another). I call that the leaf problem, and chapter 4 is built around it.



TARGET	WHAT COMES OUT	WHERE THE CAPACITY WENT
pixel	a renderer	the leaves

FIG. 3.7 Same loop, three targets. Pick what goes on the right-hand side of the guess. Start on pixels and watch where the capacity bar goes: to the leaves, which are most of the picture and the hardest part to call. Switch to the next word and the bar moves onto grammar and facts. Switch to a compact state and you get something a planner can search. The bars are illustrative. Nothing else in the loop changed.

Predict the next word and you get grammar, plus a surprising amount of knowledge about the world. That is what the next word turns on. In 2019 Alec Radford and colleagues at OpenAI showed how far it goes at scale. Their paper was *Language Models are Unsupervised Multitask Learners*.

Their model was GPT-2. It was trained on nothing but next-word prediction over a large pile of web text. It came out able to answer questions, summarise, and translate a little, and nobody had trained it for any of those. OpenAI released it in stages, uneasy about what a fluent text machine might be used for.

The model learned those tasks because the next word sometimes depends on them. That is the clearest evidence we have that the target decides what gets built.

FIG. 3.8 Step through the sentences and try to fill the blank yourself before you press Reveal. Each one is chosen so that the next word turns on something else: a fact, a translation, a summary of what came before. Nobody has to teach those as tasks. They are what the next word sometimes depends on.

Predict the next state of some compact description and you get something a planner can use. Compact means a short list of numbers rather than a picture.

So the same loop gives three different machines, and all that changed is what was put on the right-hand side of the guess. *Predict the next thing* is a family of methods rather than one. Picking the target is the design decision that matters most, so an argument about prediction should say what is being predicted.

Being right is the same as being brief

There is a second way to look at this, four decades older than Elman's result, and it comes to the same thing. Shannon was a mathematician at Bell Labs, the research arm of the American telephone company, whose business was pushing more calls down the same wire. Suppose you have to send a message down a wire and you are charged by the symbol.

You and the receiver share a predictor. Before each symbol both of you run it and get a list of what is likely next, so you need only send enough to pick the right one. Confident and correct costs almost nothing, and confident and wrong costs a lot.

PROBABILITY GIVEN	COST TO SEND
1 in 2	1.0 bits

FIG. 3.9 Drag the slider to set how likely the shared predictor said the symbol that actually arrived was, and read off what that symbol costs to send. Half way along, one in two, costs one bit. Push it towards the left end, where the predictor was confident and wrong, and watch the price climb. Push it right and it is nearly free.

Shannon made this exact in 1948. A symbol you gave a probability of one in two costs one bit (one bit is one yes or no answer, so eight bits can pick one option out of 256). One in four costs two, and one in a thousand costs about ten. That paper started information theory, and the word "bit" first appeared in print there.

Three years later he turned the argument on English itself. In *Prediction and Entropy of Printed English* he had people guess a passage one letter at a time. A language people can guess is cheap to send, and the experiment measured how cheap. That put a human being on the bottom row of Figure 3.10.

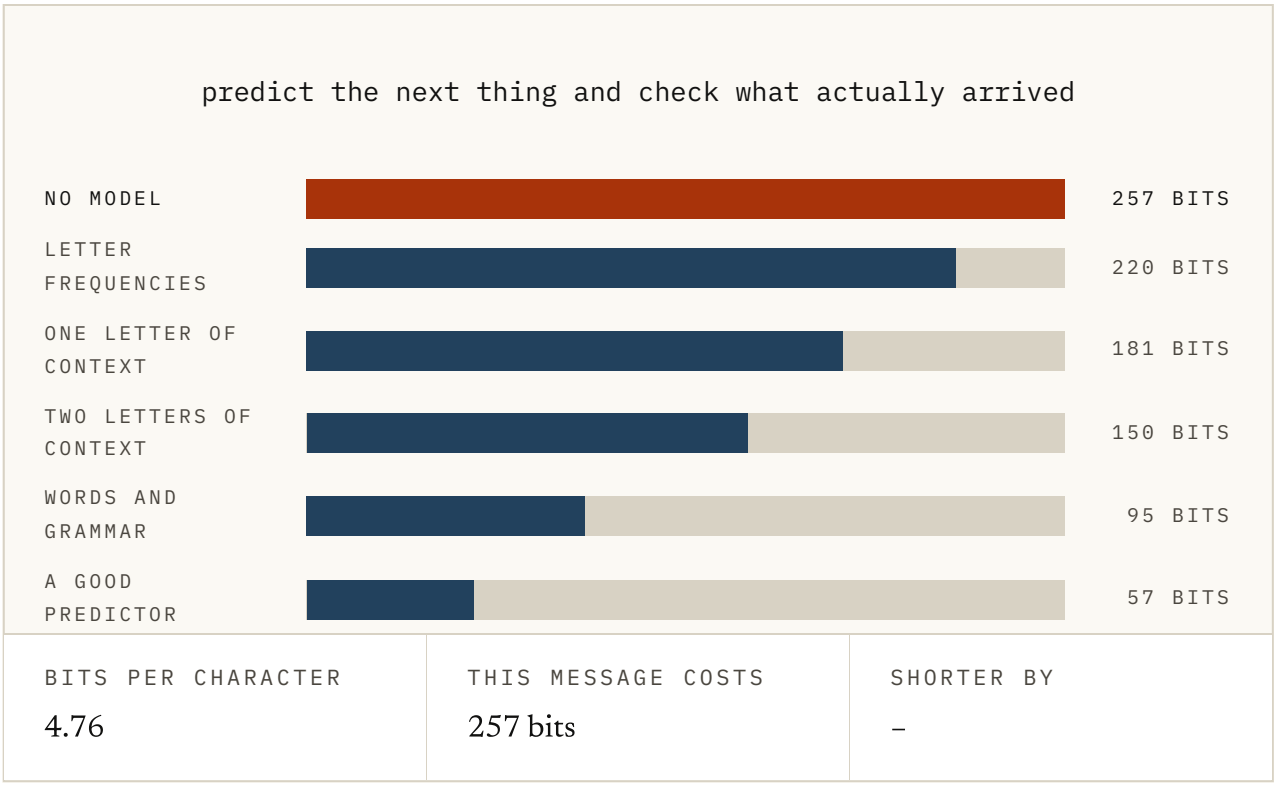


FIG. 3.10 Step the predictor up from No model to A good predictor and watch the price of the same sentence fall. The bits per character are published estimates for English rather than anything this figure computed. The bottom row is roughly where Shannon's 1951 experiment put a person guessing letter by letter, and it is also where good language models sit today.

So the length of the shortest message you can write measures how well you predicted. That makes a better predictor a better compressor. Read backwards, whatever compresses your data the hardest has understood the most about it.

Marcus Hutter, a theorist of perfect learners in the abstract, took the reading literally. The Hutter Prize is a long-running cash prize for compressing a snapshot of Wikipedia. It runs on the argument that you cannot squeeze text much further without knowing what it means. Every winner to date has been, in effect, a language model.

Jürgen Schmidhuber went a step further in 2010. Schmidhuber ran a lab in Switzerland and had co-invented the LSTM, a memory network that was the workhorse of speech recognition for years. In 2018 he would co-write the paper this course is named after. His 2010 paper was *Formal theory of creativity, fun, and intrinsic motivation.*

He proposed rewarding a learner for compression progress. That is the discovery that it can now squeeze its data further than it could a moment ago. Getting better at compressing becomes a drive in its own right, a reason to go and look at new things.

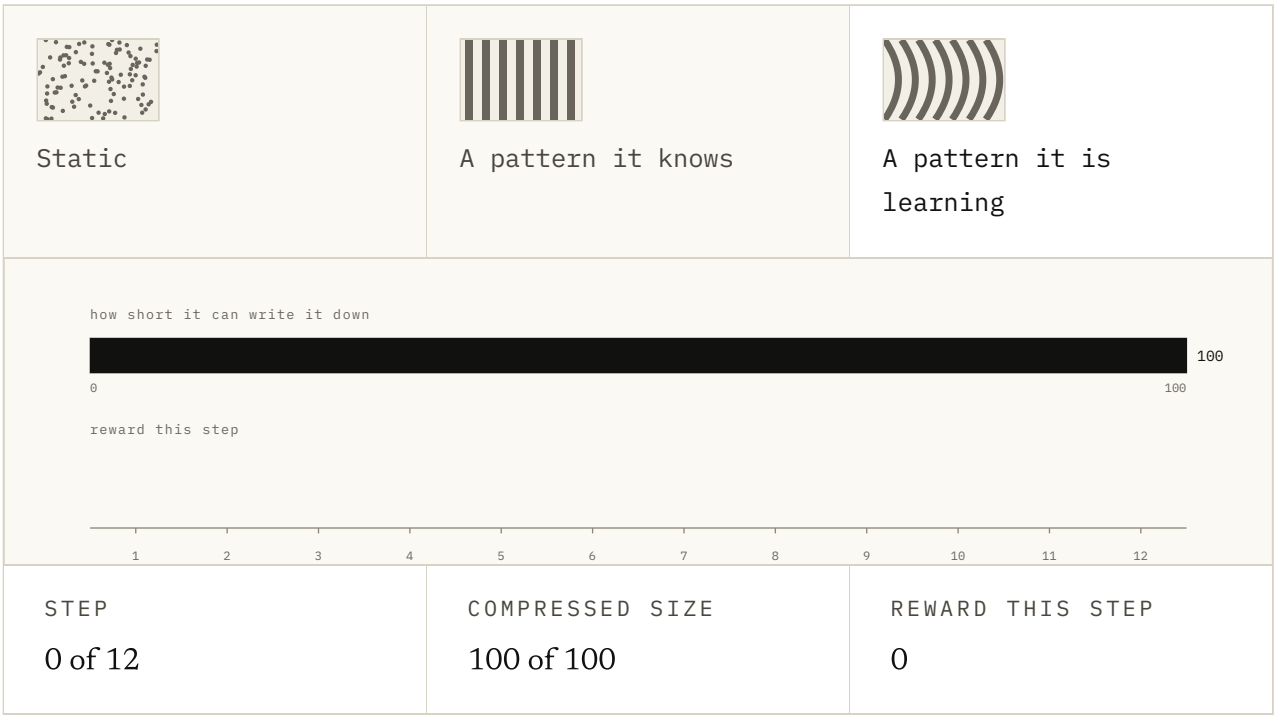
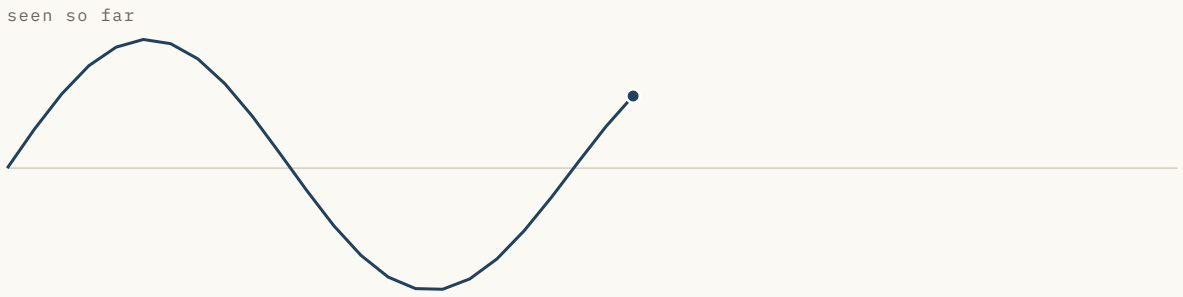


FIG. 3.11 Three things a learner could look at: static, a pattern it already knows, and a pattern it is still learning. Pick one and press Watch. The bar is how short the learner can write down what it has seen so far, and the reward at each step is how much shorter it just got. Static never gets shorter, and the known pattern is already short, so neither pays anything. All the reward is in the middle one. Illustrative.

The same reading says why the stowaways were never optional. Memorising is the costly way. A lookup table of everything that ever happened is huge, and useless on anything new.

The cheap option, the one that shortens the message, is to work out the rule that produced the data and keep that instead. The rule is the stowaway.



THE TABLE

1	0.00
2	0.31
3	0.59
4	0.81
5	0.96
6	1.02
7	0.99
8	0.87
9	0.67
10	0.41
11	0.12
12	-0.18

... and 12 more

STORES: 24 VALUES

THE RULE

the rule: two numbers (illustrative)

a 1.90 b -0.99

next = a × last + b × the one before

STORES: 2 NUMBERS

VALUES SEEN	THE TABLE STORES
24	24 values
THE RULE STORES	ON NEW DATA
2 numbers	nothing new yet

FIG. 3.12 Drag the amount of data up and watch the two boxes. The table stores every value it has seen, and grows with the data. The rule is two numbers, the same two the predictor in Figure 3.3 found, and it does not grow at all. Then press New data and see which of the two has anything to say about it.

Where it stops being enough

There are two limits.

Predicting well is not the same as being useful. A system can be excellent at continuing a recording and still hand you nothing you can act on. Saying what comes next is a different question from saying what would come next if you did something different. Only the second supports a decision. Li's network carries a board and was still only ever asked which moves were legal.

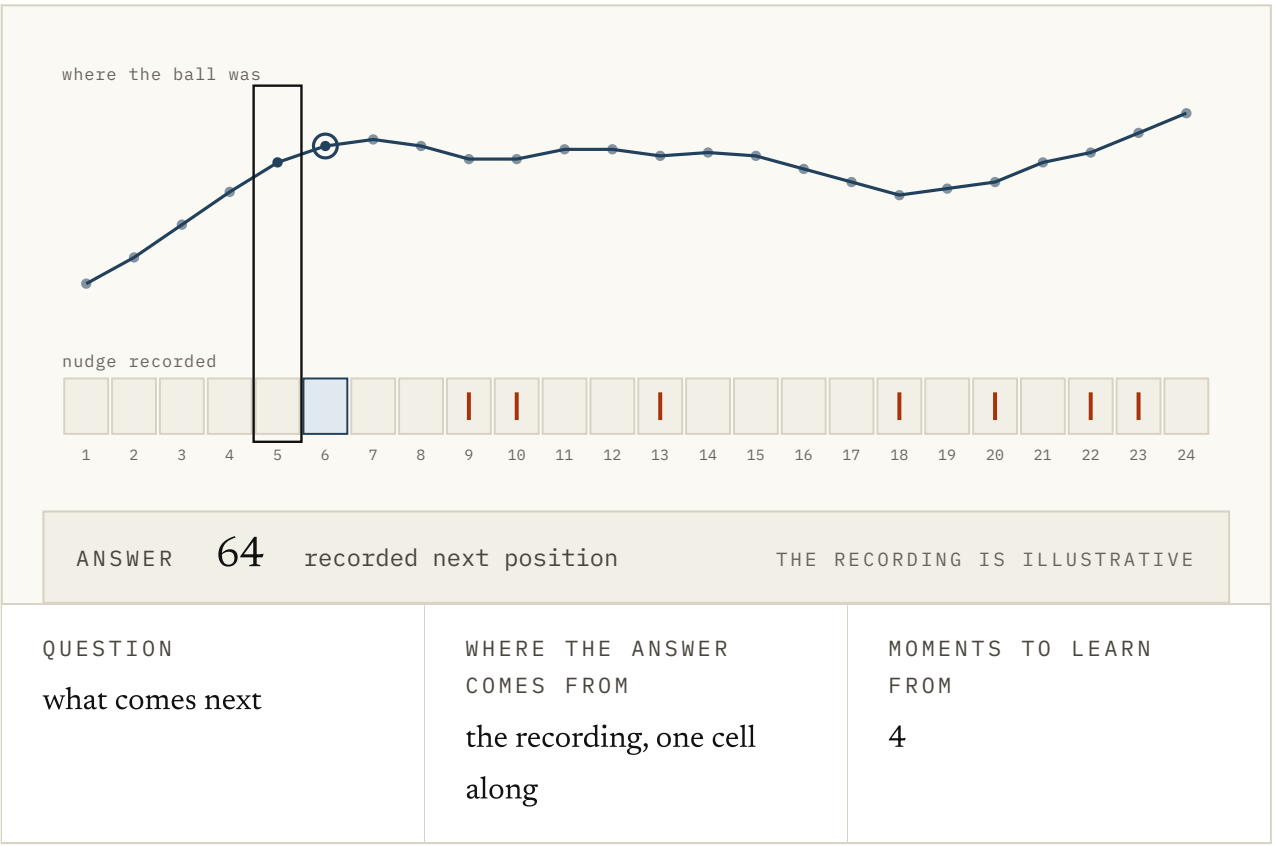


FIG. 3.13 Drag along the recording and pick a moment. Press What comes next and the answer is the very next cell, already there. Press What if I nudge it and, unless somebody happened to nudge at that moment, there is no cell to read: the answer has to be borrowed from other moments where the ball was in about the same place and somebody did, and sometimes there are none. The recording is illustrative.

Being unsurprised is not the same as being right. A predictor that has only ever seen calm weather is not surprised by calm weather. That tells you nothing about what it would do in a storm.

Low error on what you happened to record is a weaker claim than it sounds, and it is the claim most headline numbers make. Let's see it with a ball instead of weather.

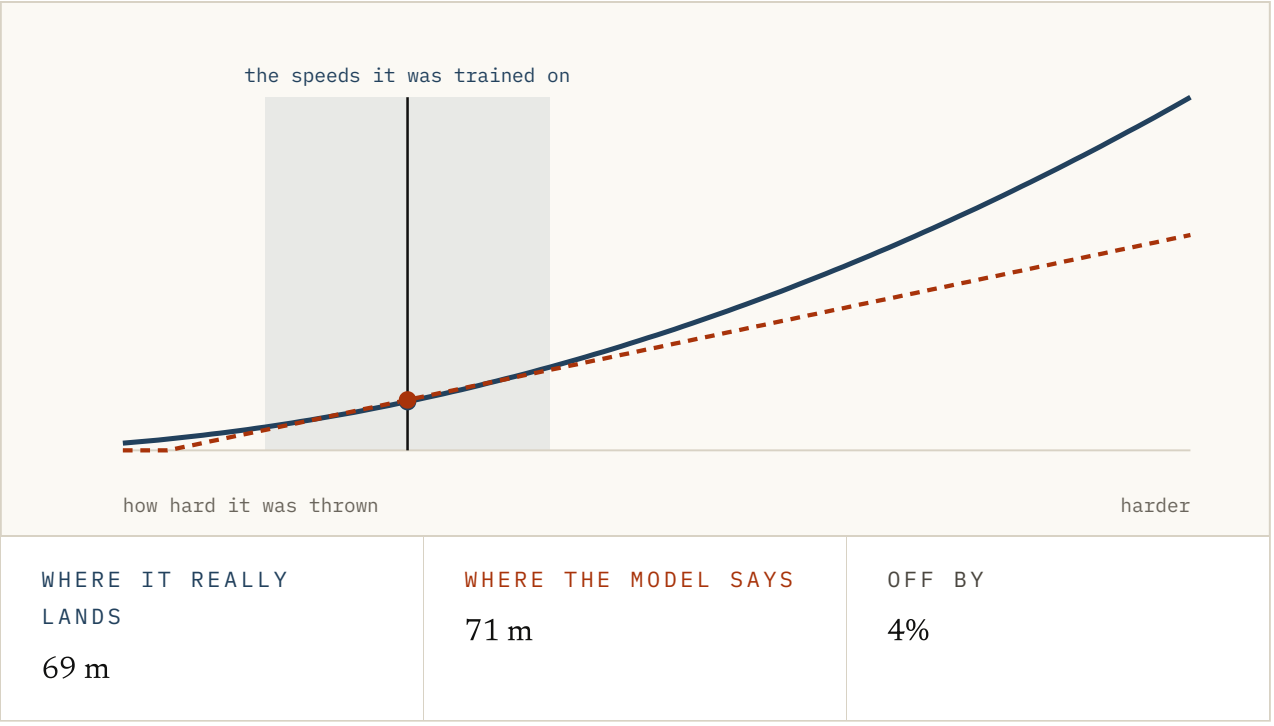


FIG. 3.14 Drag the launch speed across the band the model was trained on and it lands close to the truth, so anyone testing it there would say it has the rule. Keep dragging past the band and watch the gap open. The calm days were the band, and it never saw a storm.

Neither limit undoes the argument. Prediction is enough to pull structure out of raw experience, and it is the cheapest signal anyone has found. The line from Shannon through Elman to Othello-GPT is people proving that with better instruments.

Everything here has assumed that what is being predicted is small: two numbers for a ball, and one word at a time for language. A camera hands you two million numbers, thirty times a second, and almost all of them are the leaves. That is the leaf problem arriving at full size, and it is where chapter 4 picks up.

Hopefully this chapter helped. There is a short quiz below if you want to check that the chain held together for you. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section, and I would be glad to hear from you.

Try it

1 One photograph of a ball in flight. What can you not work out from it?

a. Where the ball is
b. How big the ball is
c. Which way it is going and how fast
d. What colour it is

> ANSWER c. Which way it is going and how fast. Direction and speed are not in any single frame. They exist in the relationship between frames, which is the kind of thing a predictor has to build for itself.

2 Why is next-thing prediction so much cheaper to train on than labelled data?

a. The models are smaller
b. The answer is already in the recording, so nobody has to write labels
c. It needs less computing time
d. It converges in fewer steps

> ANSWER b. The answer is already in the recording, so nobody has to write labels. Every moment is the answer to the moment before. That turns any recording of anything into a training set, and removes the step where a person has to annotate a million examples.

3 Jeffrey Elman's network was trained only to predict the next word, and its internal states sorted themselves into nouns, verbs, animate and inanimate. Why?

a. Those categories were in the training labels
b. The architecture had one unit per category
c. A word's category is what its next word depends on, so the categories were the cheapest way to be less wrong
d. It memorised the corpus

> ANSWER c. A word's category is what its next word depends on, so the categories were the cheapest way to be less wrong. Nothing supplied the categories. Prediction rewards whatever makes the next thing less surprising, and for words that is grammatical category.

4 A probe finds board state inside a network trained only on legal Othello moves. What turns that from a curiosity into a finding?

a. The probe is very accurate
b. The network was large
c. Changing that internal board changes the moves the network then makes
d. The board can be drawn as a picture

> ANSWER c. Changing that internal board changes the moves the network then makes. Accuracy alone could be a coincidence in the numbers. Intervening on the representation and watching behaviour follow is what shows the network is using it.

5 Under a good predictor, a symbol you were confident about and got right costs almost nothing to send. Why?

a. It can be left out of the message
b. The cost of a symbol is set by the probability you gave it
c. Common symbols are stored in a table
d. The receiver guesses it and does not need the message

> ANSWER b. The cost of a symbol is set by the probability you gave it. Claude Shannon made the price exact. High probability means a short code, which is why a better predictor is a better compressor.

6 What does the compression view say about memorising the training data?

a. It is the best available strategy
b. It is the expensive option: a lookup table of everything is enormous and useless on anything new
c. It compresses better than any rule
d. It is what all learning does

> ANSWER b. It is the expensive option: a lookup table of everything is enormous and useless on anything new. The short message comes from finding the rule that generated the data and keeping that instead. Memorising is what compression penalises.

7 Same loop, different target: predict the next pixel, or the next word, or the next compact state. What does that choice change?

a. Nothing, the loop is what matters
b. Only how long training takes
c. What the system ends up good at, and what its capacity gets spent on
d. Whether the method counts as self-supervised

> ANSWER c. What the system ends up good at, and what its capacity gets spent on. Predict every pixel and most of the capacity goes to leaves, because that is where most of the pixels are. Picking the target is the design decision that matters most.

8 A model has very low error predicting the next frame of a recording. What does that alone not tell you?

a. That it saw the recording
b. What it would predict if you acted differently, and how it behaves outside what it recorded
c. That its error was measured correctly
d. That the recording was long enough

> ANSWER b. What it would predict if you acted differently, and how it behaves outside what it recorded. Being unsurprised by what you happened to record is a weaker claim than it sounds, and saying what comes next is not the same as saying what would come next under a different action.

FIG. 3.15 Eight questions on the loop and what it produces. The last two are the ones that separate a good predictor from a useful one, and they are the ones I'd most like you to get right.

Sources

A Mathematical Theory of Communication SHANNON, 1948 ↗

Where the price of a symbol becomes the probability you gave it. Everything about prediction and compression being one job starts here.

https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

Prediction and Entropy of Printed English SHANNON, 1951 ↗

Claude Shannon sat people down and had them guess English one letter at a time. The bottom row of Figure 3.10 is roughly what he measured.

https://doi.org/10.1002/j.1538-7305.1951.tb01366.x

Finding Structure in Time ELMAN, 1990 ↗

Train a small network to predict the next word, then look inside: nouns, verbs, animate and inanimate, none of it asked for.

https://doi.org/10.1207/s15516709cog1402_1

Emergent World Representations LI ET AL., 2022 ↗

A network given nothing but legal Othello moves, with the board found inside it and causally manipulated.

https://arxiv.org/abs/2210.13382

Othello-GPT has a linear emergent world representation NANDA ET AL., 2023 ↗

The follow-up that sharpened what the probe was actually reading.

https://arxiv.org/abs/2309.00941

Language Models are Unsupervised Multitask Learners RADFORD ET AL., 2019 ↗

Next-word prediction, scaled, turning into capabilities nobody trained for. The strongest evidence that the target choice is the design decision.

https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

The Hutter Prize HUTTER, ONGOING ↗

A cash prize for compressing a snapshot of Wikipedia, run on the argument that you cannot squeeze text further without modelling what it means.

http://prize.hutter1.net/

Representation Learning: A Review and New Perspectives

BENGIO, COURVILLE & VINCENT, 2013 ↗

The survey that laid out what a good learned representation is for, written before prediction had finished winning the argument.

https://arxiv.org/abs/1206.5538

Formal theory of creativity, fun, and intrinsic motivation SCHMIDHUBER, 2010 ↗

Compression progress as a drive in its own right: not just a way to measure a model, but a reason to go and look at something.

https://people.idsia.ch/~juergen/creativity.html

FIG. 3.16 I've ordered these for someone building on this rather than for historical completeness.

04 What Is Latent Space?

BY NILUSHANAN KULASINGHAM

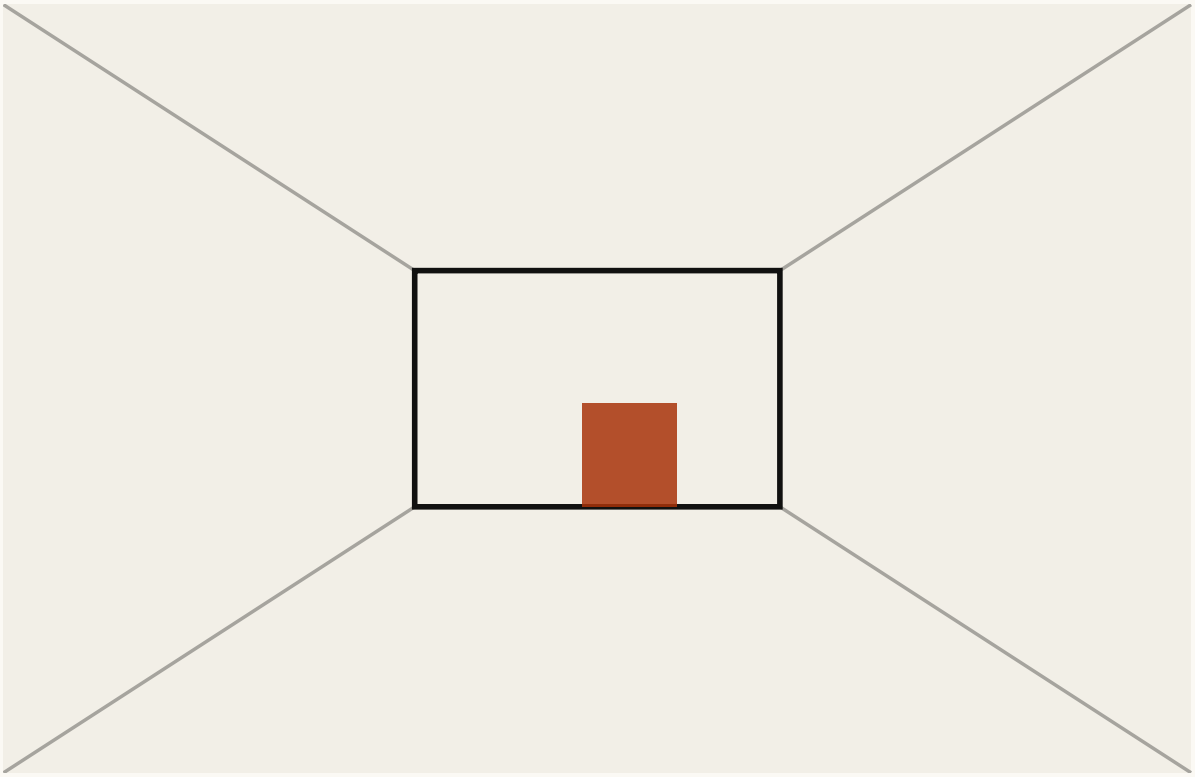
11 MIN READ · INTERACTIVE

> A camera measures tens of thousands of numbers and the decision needs two or three. What happens at the squeeze between them, and why that narrow point sets the ceiling on everything downstream.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/latents.

Stand in a doorway and look into a room. Your eyes are taking in something like a hundred million measurements a second. Now decide whether to walk forward. For that decision you need almost none of them.

How far away is the far wall? Is there a way through, and roughly where? That is two or three numbers, and everything else your eyes collected was, for this decision, decoration.



THE ROOM THOSE TWO NUMBERS DECODE TO

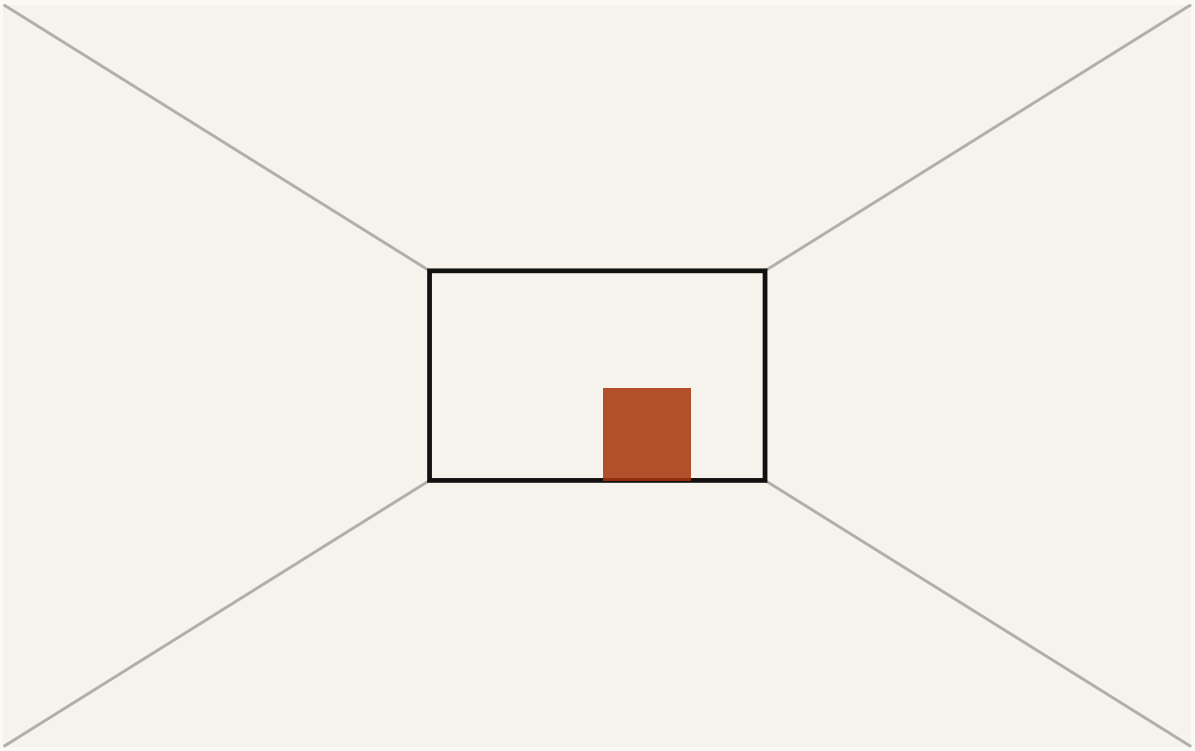
FIG. 4.1 Have a go at this one before reading on. Drag anywhere in the square on the right. Every point in it decodes to a room, and the room is computed from those two numbers rather than looked up. Watch how little drag it takes to change what you would do. The picture is tens of thousands of numbers, and the description that produced it is two.

Those two or three numbers have a name. They are the *latent* (latent means hidden, or not directly visible. The numbers are never shown to you and nothing labels them, but the picture is what it is because of them) description of the scene. You will meet the phrase "latent space" in nearly every paper from here on.

Somewhere between the camera and the decision there has to be a squeeze, and whatever comes out the other side is all the rest of the system ever gets. That is the case for latents, and the same fact is the case against them.

The squeeze

The architecture is a narrow opening in the middle of a system. One half turns the picture into a short list of numbers. The other half takes the list and tries to rebuild the picture. Train the pair together and there is only one way to succeed: the short list has to carry what the picture was made of.



HOW FAR THE WALL IS
WHERE THE WAY THROUGH IS
WHICH WAY THE CAMERA IS TIPPED
GRAVEL ON THE FLOOR
SENSOR NOISE
HOW BRIGHT IT IS

COULD YOU DECIDE WHAT TO DO?
Yes. Two numbers is the whole decision.

WHAT THE EXTRA WIDTH BOUGHT
Nothing wasted yet. Everything through the squeeze changes what you would do.

FIG. 4.2 This is that narrow opening with a dial on it. The scene is generated from six numbers and the squeeze keeps the first few. Slide the width up from zero and watch the two readouts underneath. Two numbers in, the decision is settled. The rest is gravel and sensor noise, real but beside the point.

The figure is rigged: the two factors the decision turns on sit in its first two slots. A learned system gets no such ordering for free.

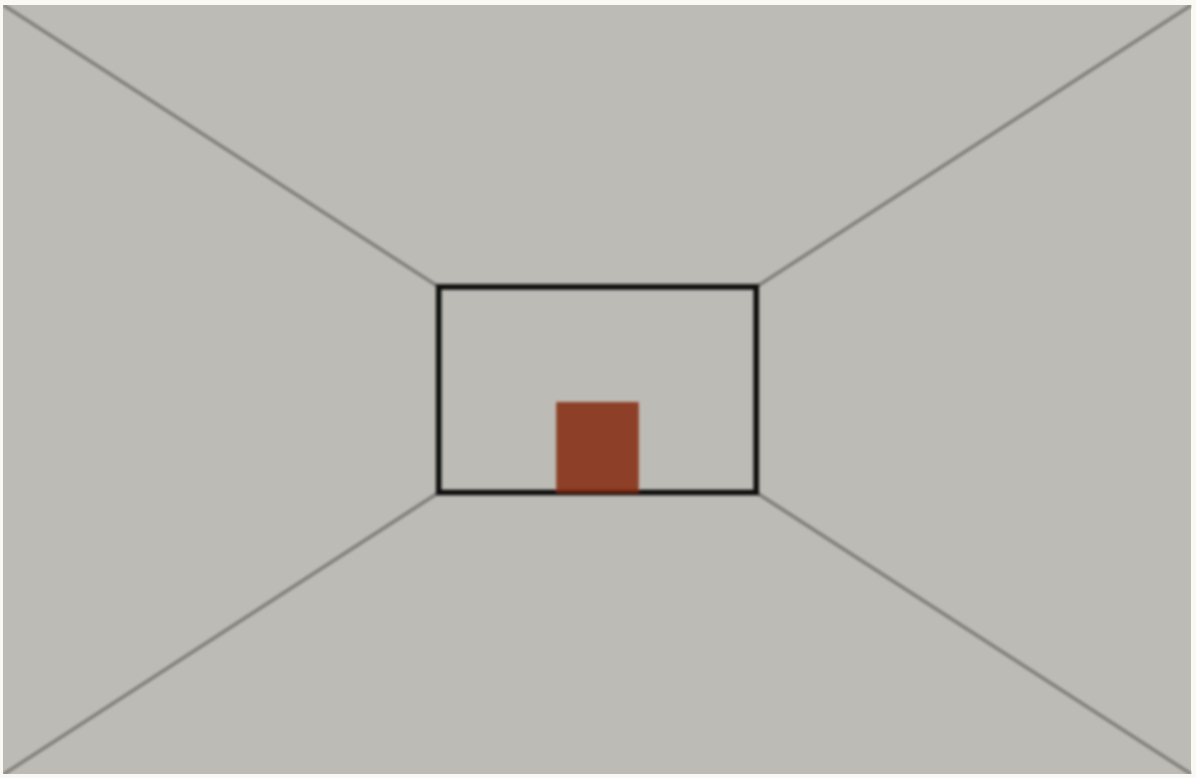
Geoffrey Hinton and Ruslan Salakhutdinov made the case for the squeeze in 2006, in *Reducing the Dimensionality of Data with Neural Networks*. Hinton taught in Toronto and had stuck with neural nets through the lean years, when most of the

field had moved on. Salakhutdinov was his student. They put the paper in *Science* rather than a machine learning journal, which tells you who they wanted to win over.

> The two halves have names, and every paper uses them, so let's get them down now. The one that takes the picture in is the *encoder*, and the one that rebuilds it is the *decoder*. The narrow bit between them is the *bottleneck*. The pair together is an *autoencoder*: it encodes its own input and decodes it again.

Everything since has been that argument with better kit. But a bottleneck creates pressure without naming the answer. Many equally short lists of numbers can rebuild the same pictures, and nothing makes one number mean distance and another mean lighting. You need a bias for that, in the architecture, the data or the task.

Irina Higgins and colleagues at DeepMind, Google's London lab, made the case for such a bias in 2016. The paper was *Early Visual Concept Learning with Unsupervised Deep Learning*. A good representation, they argued, is one whose directions each mean one thing: turn one number and only the lighting changes.



THE ROOM, REDRAWN FROM THE TWO NUMBERS

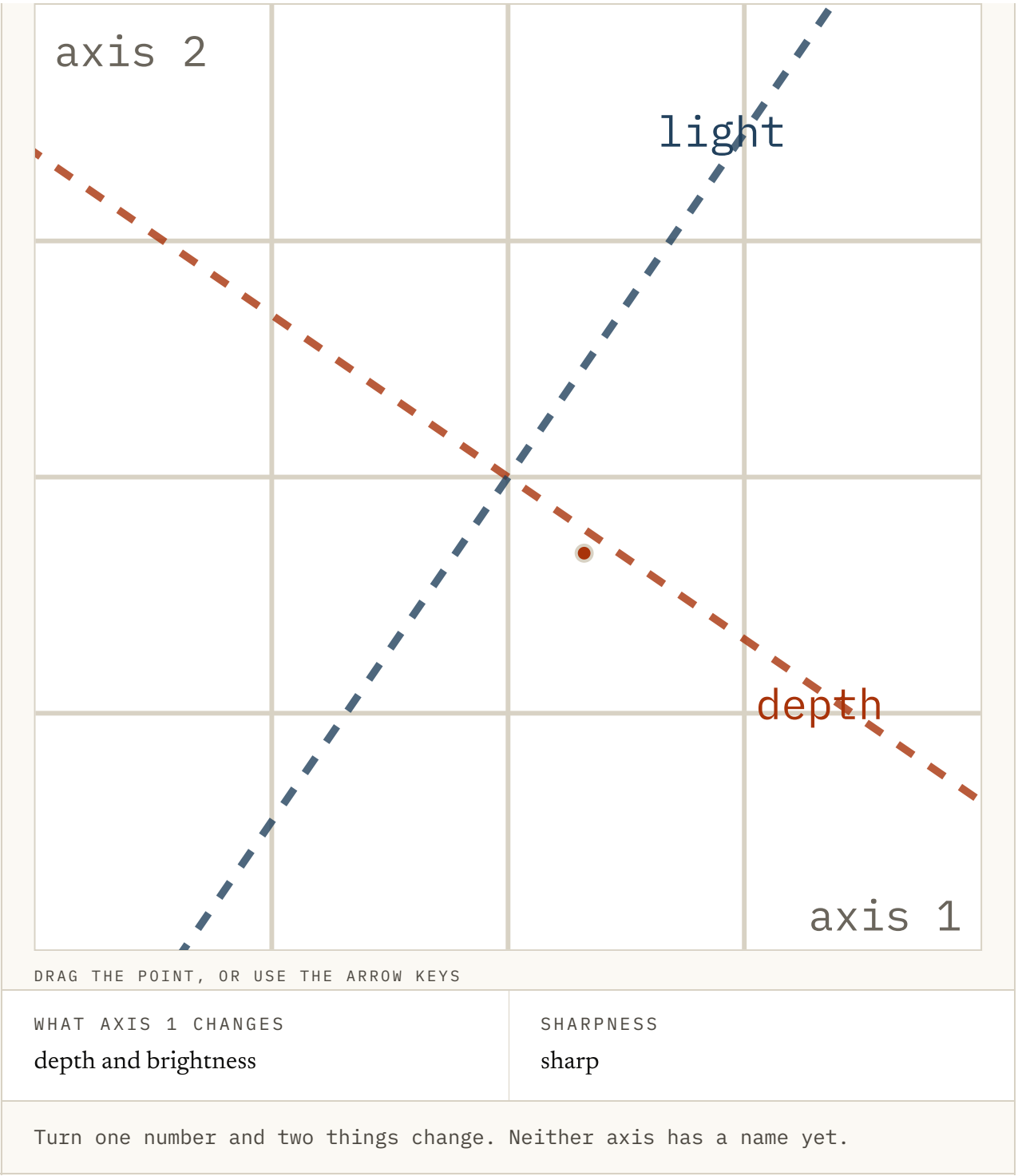


FIG. 4.3 This is Higgins's test, with the dial from the next paragraph added. Leave the pressure low and drag the point around the square: the wall moves and the light changes together, so neither axis has a name. Now turn the pressure up and drag again. One direction moves the wall and the other changes the light, and the room has gone soft. Everything here is illustrative.

Their *beta-VAE* a year later was an autoencoder of this shape with one dial added. The dial (the beta in the name) turns the pressure on the bottleneck up, and past one the axes start lining up with nameable things, the turn of a face or the size of a

shape. The same paper shows the cost, which you just saw: press harder and the pictures go blurry.

By 2018 the squeeze was inside a working agent. David Ha and Jürgen Schmidhuber published *World Models* that year, an agent that learned a driving game through a model of it. Every frame was crushed to thirty-two numbers, and the parts that predicted and chose never saw a pixel.

The same year Danijar Hafner and colleagues built PlaNet, with Ha among the authors. The paper was *Learning Latent Dynamics for Planning from Pixels*. It squeezed raw pixels the same way, then planned inside what came out, and never rebuilt a frame to decide anything. At decision time the short list was all it had, the bolder of the two designs.

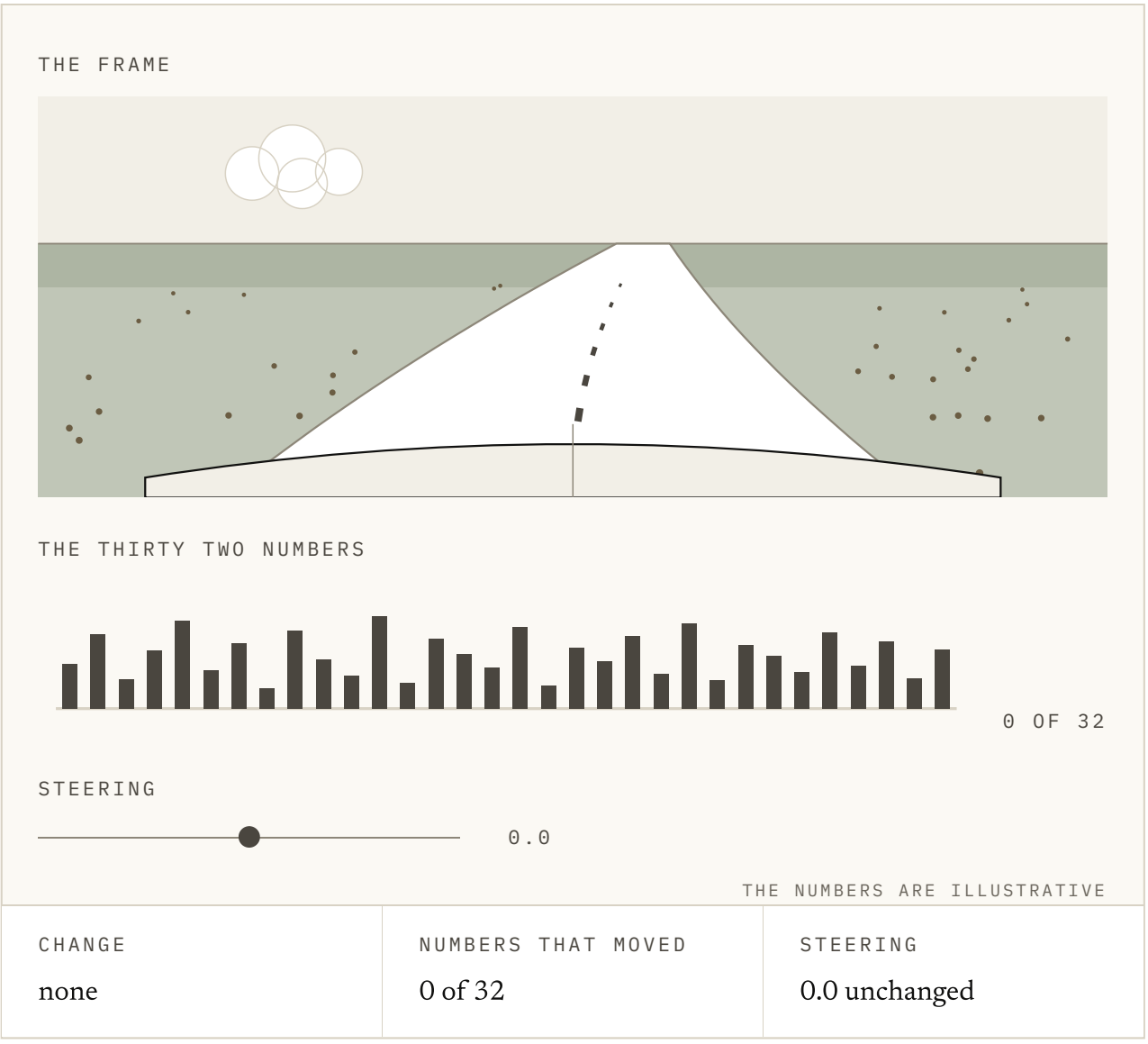


FIG. 4.4 One frame from the driving game, and the thirty-two numbers it was crushed to. Press a change and watch which of the numbers move and whether the steering moves with them. Move the bend or the car and several do. Reshuffle the gravel or reshape the cloud and the picture is plainly different while not one number moves, so nothing after the squeeze ever hears about it. The numbers are illustrative.

Then in 2019 Francesco Locatello and colleagues closed the question Higgins had opened. Pulling the hidden factors apart, one per axis, is called disentanglement. With no labels and no built-in bias, they proved, it cannot be done. For any squeeze whose axes mean something, another draws the same pictures just as well with axes that mean nothing.

They then trained thousands of models, and which axes you got came down mostly to the random seed. The paper was *Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations*. It took the best paper award at the ICML conference that year. A lot of latent-space writing has not caught up with it.

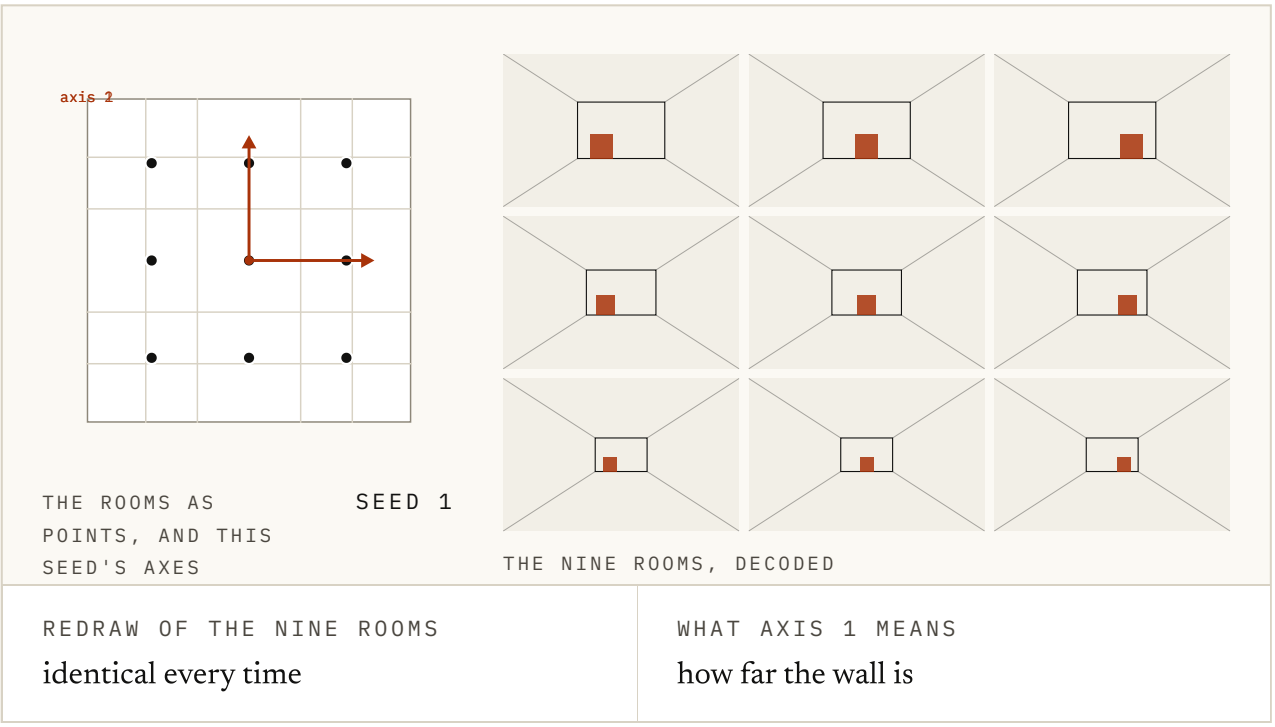


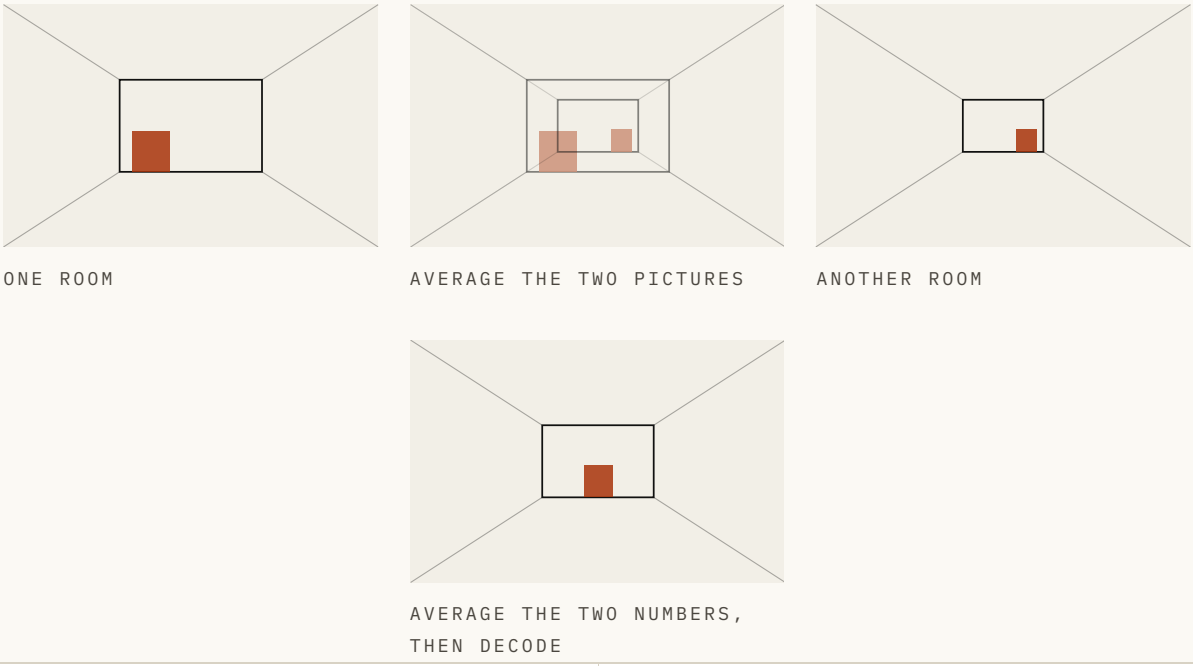
FIG. 4.5 Press Retrain a few times. Each press is a fresh random seed, and each one hands you a different pair of axes over the same rooms. Read the two lines underneath: the pictures come back the same every time, and what the first axis means does not. That is Locatello's result, and it is also why Figure 4.2 had to be rigged. Illustrative.

Look at who was unaffected. Ha and Schmidhuber's agent never needed its thirty-two numbers to mean anything a person could name, and neither did PlaNet's planner. The squeeze carried on, because the agents built on it had only ever asked that it keep what the next step turns on.

A place, not a list

Those numbers are coordinates. In a latent space that training has shaped well, they name a point with useful neighbours, directions and distances. Nothing guarantees that; a plain autoencoder can learn a twisted lookup table instead.

Once it has been shaped, you can do things there that make no sense in pixels. You can ask what is nearby, move in one direction and watch the world change smoothly in one way, or take two rooms and stand between them. That last one is the figure below.



IS THE BLENDED PICTURE A ROOM?

No. It is two rooms at once, which is not a thing you could walk into.

IS THE DECODED ONE A ROOM?

Yes, everywhere along the slider. Every point in that space decodes to one.

FIG. 4.6 Both rows are real output from this constructed decoder. Drag the blend slider and watch the two middle panels. Averaging two pictures gives you both rooms at once, faintly. Averaging the two coordinates and decoding stays among the rooms the decoder has learned to draw. A real latent space only does this if training has made the region between examples mean something.

Halfway between two pictures is a ghost, and halfway between two descriptions is a room. That matters because prediction, planning and search all move somewhere you have not been, and if every step lands on nonsense none of them work.

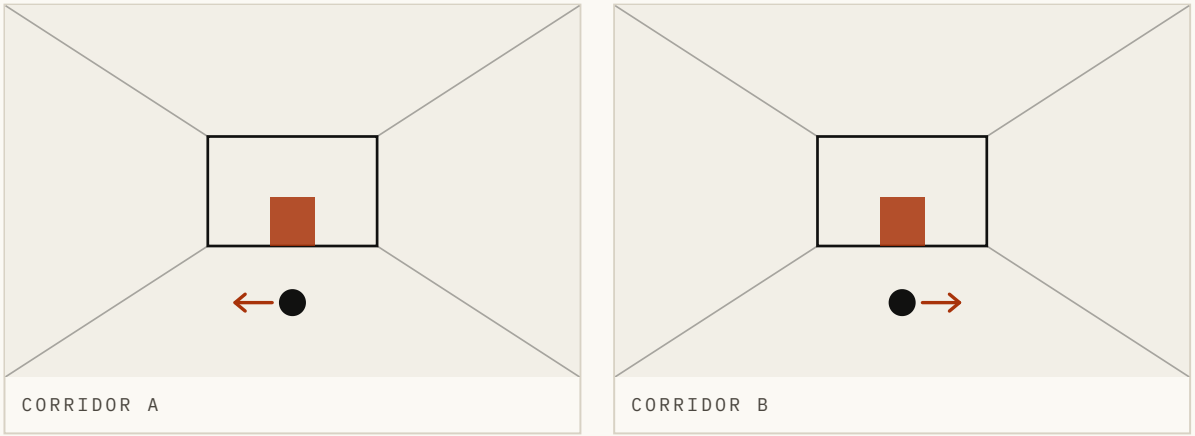
> This is also why the noise in a *variational* autoencoder, the VAE that beta-VAE builds on, is deliberate. Diederik Kingma and Max Welling built it in 2013, in *Auto-Encoding Variational Bayes*; Kingma was Welling's doctoral student. It blurs where each picture lands so neighbouring points decode to similar things. That is what makes the space navigable, and why Ha and Schmidhuber used one as their agent's eyes five years later.

What makes one good

Being small is not the goal in itself. A single number is very small and useless. What you want is that the things which matter are easy to get at, and the things that do not are gone. Yoshua Bengio, Aaron Courville and Pascal Vincent set that down in 2013, in a survey called *Representation Learning: A Review and New Perspectives*.

The three were colleagues in Montreal. Bengio had kept a neural network lab going through the same lean years as Hinton, and would later share the Turing Award with Hinton and Yann LeCun. The survey is still the best single statement of what a learned representation is for. So let's take three tests from it, in rough order of how often they fail.

Does it keep what the future turns on? Take two situations that lead to different outcomes and land on the same point: nothing downstream can tell them apart. This is the failure that shows least, because the reconstructions can look fine while it happens. Ha and Schmidhuber's thirty-two numbers were trained only to redraw frames, and the authors flagged the risk themselves.



WHICH WAY WILL THE BALL GO?

press a corridor first

WHERE THE TWO LAND

press both to see

Press A, then B, and watch the square.

FIG. 4.7 Two corridors, and the only difference is which way the ball is rolling. Press each one in turn and watch where it lands in the square: the same point, because a squeeze scored on redrawing the frame found nothing worth keeping in a few pixels of motion. Now look at the redraw, which is fine, and at the question underneath, which cannot be answered. Flip to scoring on the next frame and the two land apart. Illustrative.

Do nearby points mean nearby things? A space where a small step can land anywhere is just a lookup table. Smoothness lets you move through it on purpose, and it is what the VAE's noise was built for. As you saw in Figure 4.6, it is also what lets you stand between two rooms.

Is it easy to predict forward? A description can be perfectly faithful and still be a nightmare to run forward in time. The reason anyone compressed was to get something they could roll forward cheaply.

What the squeeze costs

Rebuilding the picture is a proxy, and proxies drift. The test asks whether the short list can redraw what the camera saw. What you wanted to know is whether it kept what the decision turns on.

The things that take up the most pixels are usually not the things that matter. A description scored on rebuilding pixels spends itself on texture and weather, and loses the small fast object you needed to avoid. Figure 4.2 was rigged to hide this, so let's take the rigging off.

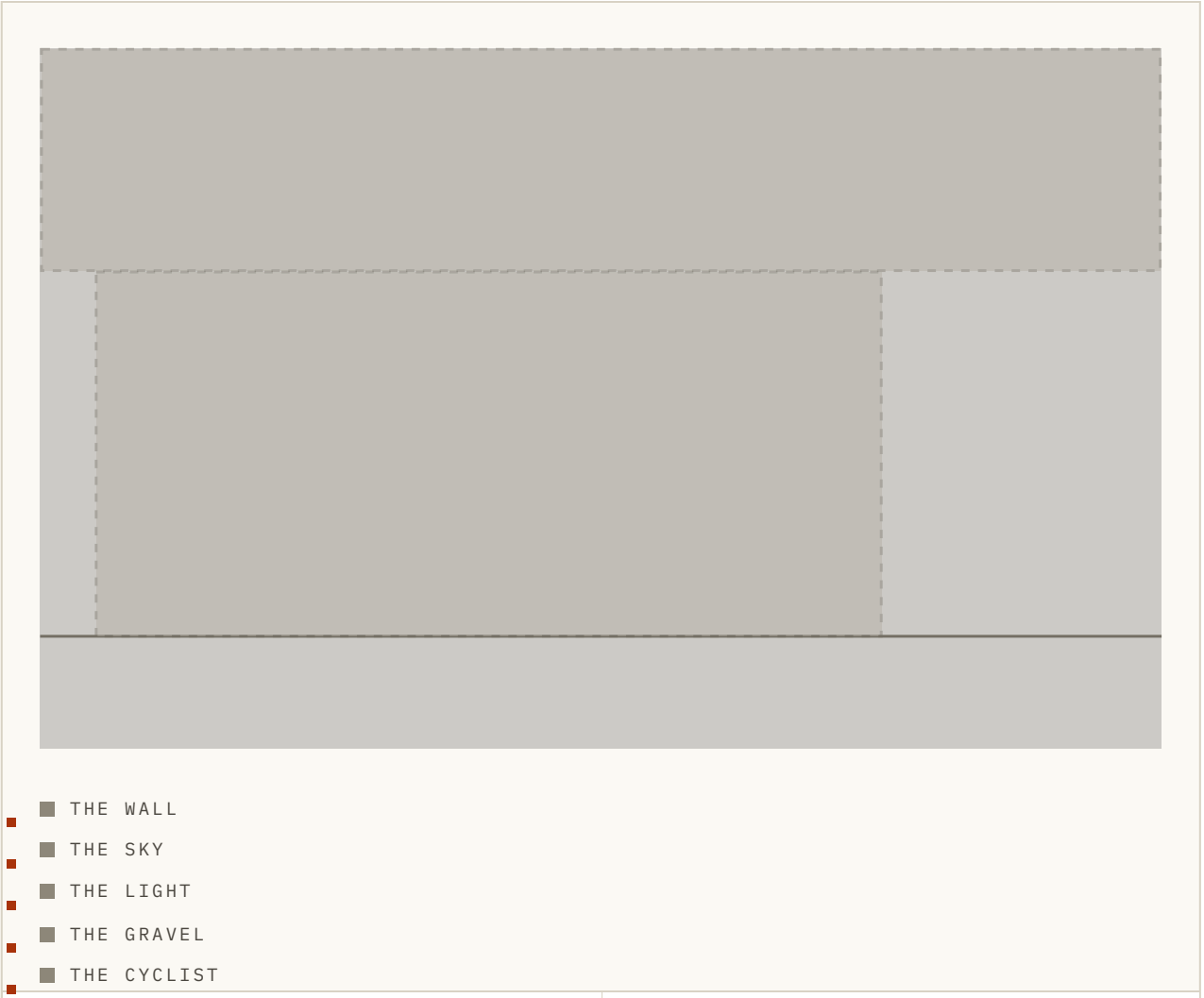


Figure 4.2 kept the decision's two factors first. A redraw score keeps the biggest first.

FIG. 4.8 The same kind of squeeze as Figure 4.2, but this time the slots go to whatever covers the most pixels, which is what a redraw score rewards. Slide the budget up and watch the order things come through: the wall, the sky, the light, the gravel, and only then the cyclist. Now read the two lines underneath. The redraw score is high long before the one thing you needed arrives. Illustrative.

The second cost is subtler, and I think more serious. Anything the squeeze drops is gone for good from that point on. Everything after the bottleneck works from the short list and nothing else.

That choice is made early, by a part of the system that has no idea what task is coming, and it sets the ceiling on everything after it.

That is why the argument since has been about the middle of the pipe. PlaNet trained the squeeze together with the forward model, and LeCun's JEPA line (chapter 7) asks whether to rebuild pixels at all.

Hopefully this chapter helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section and I would be glad to hear from you.

Try it

1 A camera hands you tens of thousands of numbers per frame. Deciding whether to walk forward needs two or three. What is the bottleneck for?

a. Making the model smaller so it runs faster
b. Forcing everything through a narrow opening, so what survives is what the pictures were actually made of
c. Removing noise from the camera
d. Compressing the file on disk

> ANSWER b. Forcing everything through a narrow opening, so what survives is what the pictures were actually made of. Speed and file size are side effects. The reason to squeeze is that succeeding at the squeeze requires recovering the things that generated the picture in the first place.

2 Nobody hand-picks what the short list should contain. Why does it end up holding the right things anyway?

a. The architecture has one unit per concept
b. If the pictures were generated by a few things changing, recovering those things is the cheapest way to rebuild them
c. The training labels say so
d. The decoder is told the answer

> ANSWER b. If the pictures were generated by a few things changing, recovering those things is the cheapest way to rebuild them. The ordering falls out of the pressure rather than being designed. That is the appeal, and also why the ordering is only roughly the one you wanted.

3 Average two pictures of two different rooms. What do you get?

a. A room halfway between them
b. Both rooms at once, faintly, which is not a room
c. The first room
d. An empty picture

> ANSWER b. Both rooms at once, faintly, which is not a room. This is what pixel-space blending does. It is the clearest reason to want a space where the point between two valid things is itself valid.

4 Average the two short descriptions instead, then decode. What do you get?

a. The same ghost
b. A room, because every point in that space decodes to one
c. Nothing, the numbers do not add
d. A picture of both rooms side by side

> ANSWER b. A room, because every point in that space decodes to one. Prediction, planning and search all involve moving somewhere you have not been. That only works if the places in between mean something.

5 Why is the noise in a variational autoencoder not just a nuisance?

a. It makes training faster
b. It hides the training data
c. Blurring where each picture lands forces neighbouring points to decode to similar things
d. It reduces the number of parameters

> ANSWER c. Blurring where each picture lands forces neighbouring points to decode to similar things. Smoothness is the property that makes the space navigable. Without it you have a lookup table with extra steps.

6 Which of these is NOT what makes a compact description a good one?

a. It keeps what the future turns on
b. Nearby points mean nearby things
c. It is as small as possible
d. It is easy to step forward in time

> ANSWER c. It is as small as possible. A single number is very small and useless. Small is a consequence of keeping the right things, never the goal on its own.

7 A description is scored on how well it can rebuild the camera image. What can go wrong?

a. Nothing, that is exactly the right test
b. It spends itself on whatever occupies the most pixels, which is usually not what the decision turns on
c. It becomes too small to be useful
d. It stops being differentiable

> ANSWER b. It spends itself on whatever occupies the most pixels, which is usually not what the decision turns on. Reconstruction is a proxy. Texture and weather are most of the pixels; the small fast object you needed to avoid is not.

8 Why is the width of the bottleneck an uncomfortable place to put an important decision?

a. It is hard to tune
b. Everything after it works only from the short list, so whatever was dropped is gone, and it was dropped before anyone knew the task
c. It uses too much memory
d. It has to be chosen before training

> ANSWER b. Everything after it works only from the short list, so whatever was dropped is gone, and it was dropped before anyone knew the task. The loss is not recoverable downstream. A part of the system with no idea what is coming sets the ceiling on everything after it.

FIG. 4.9 Eight questions on the squeeze and the space it produces. The last two are about the parts that are easy to nod along to and hard to keep hold of.

Everything here has happened one moment at a time. A picture goes in, a short description comes out, and nothing has moved yet.

Sources

Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations

LOCATELLO ET AL., 2019 ↗

The result that prevents a bottleneck from being magic: without inductive biases, unsupervised disentanglement is impossible in general and the axes are not identifiable.

https://arxiv.org/abs/1811.12359

Auto-Encoding Variational Bayes KINGMA & WELLING, 2013 ↗

The variational autoencoder. The noise it adds is the reason the space ends up navigable instead of a scatter of unrelated addresses.

https://arxiv.org/abs/1312.6114

Reducing the Dimensionality of Data with Neural Networks

HINTON & SALAKHUTDINOV, 2006 ↗

The bottleneck argument before it had modern machinery behind it. Paywalled.

https://doi.org/10.1126/science.1127647

World Models HA & SCHMIDHUBER, 2018 ↗

Every frame crushed to thirty-two numbers, and everything after that working only from those. The clearest example of the squeeze in a working agent.

https://arxiv.org/abs/1803.10122

Learning Latent Dynamics for Planning from Pixels (PlaNet) HAFNER ET AL., 2018 ↗

Encodes pixels to a compact state and then plans in it, without ever rebuilding a frame in order to decide anything.

https://arxiv.org/abs/1811.04551

Neural Discrete Representation Learning (VQ-VAE) VAN DEN OORD ET AL., 2017 ↗

What happens when the short list is forced to be a handful of discrete symbols rather than continuous numbers.

https://arxiv.org/abs/1711.00937

beta-VAE HIGGINS ET AL., 2017 ↗

Turn the pressure on the bottleneck up and the axes start to line up with things you can name. Also a good demonstration of what that costs.

https://openreview.net/forum?id=Sy2fzU9gl

Early Visual Concept Learning with Unsupervised Deep Learning

HIGGINS ET AL., 2016 ↗

The argument that a good representation is one whose directions mean something, written before it was a crowded field.

https://arxiv.org/abs/1606.05579

Representation Learning: A Review and New Perspectives

BENGIO, COURVILLE & VINCENT, 2013 ↗

Still the best single statement of what a learned representation is supposed to be for, and of how many of these questions were already open.

https://arxiv.org/abs/1206.5538

FIG. 4.10 I've ordered these for someone building on this rather than for historical completeness.

05 What Is a Dynamics Model?

BY NILUSHANAN KULASINGHAM

15 MIN READ · INTERACTIVE

> In training, the model is handed the truth at every step. The moment you deploy it, it gets its own last answer instead. What carries the past forward, and why the headline accuracy number measures a job the model will never be asked to do.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/dynamics.

If you have read the last chapter, you have a state: a short list of numbers that stands in for what the camera saw. The next step is the one where that state moves. A dynamics model, also called a transition model, does one job. Take what you know, take what you did, and produce what you know next.

The awkward word there is know, because it covers more than the last picture. It is everything that has happened and still matters, folded into something small enough to carry. A ball behind a wall is still moving. A door you opened four rooms ago is still open.

A dynamics paper leads with its one-step error, and it is the easiest number to be impressed by. I was impressed by it for a good while, until I looked at what it measures. Let's start with the job itself.

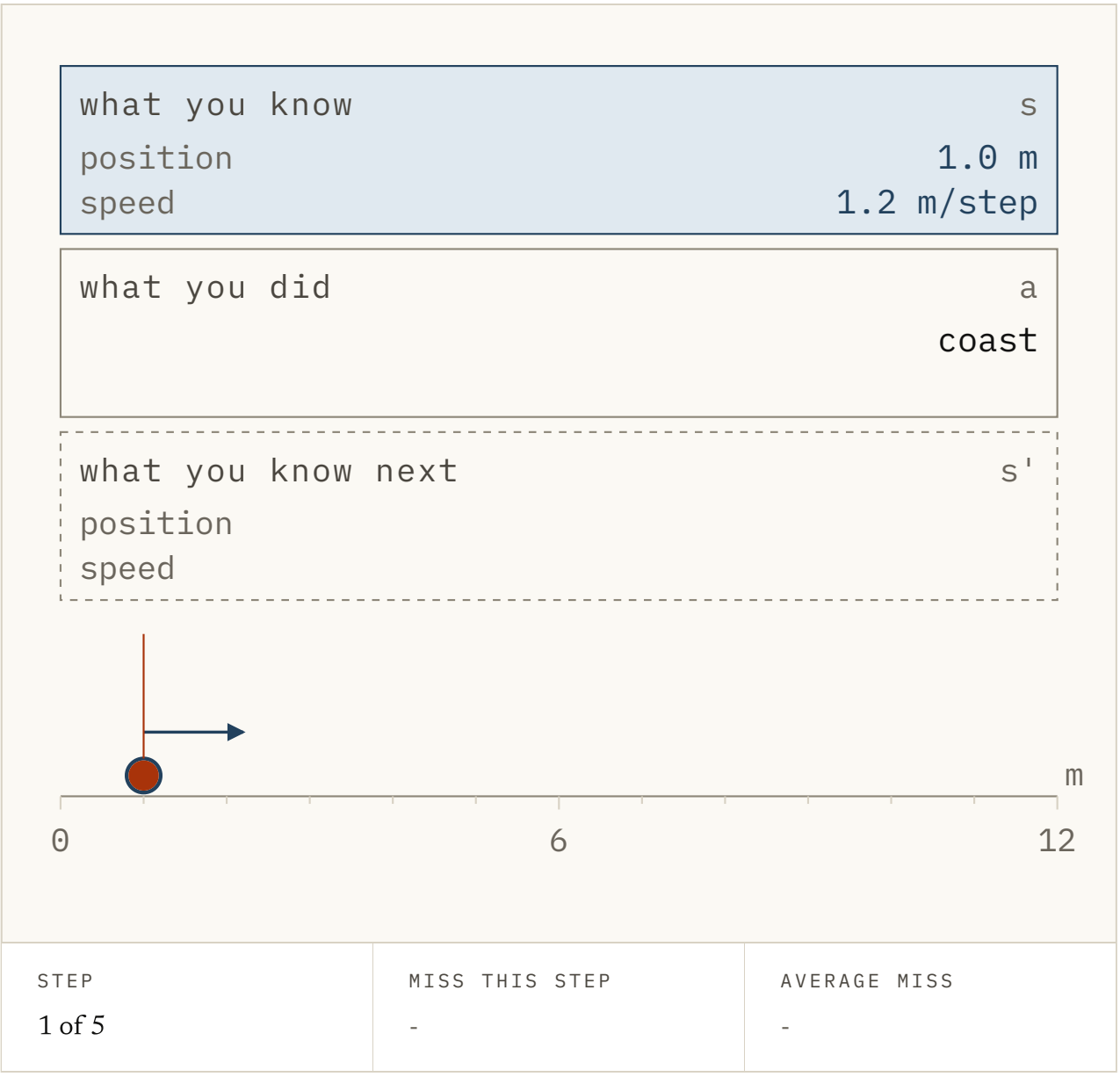
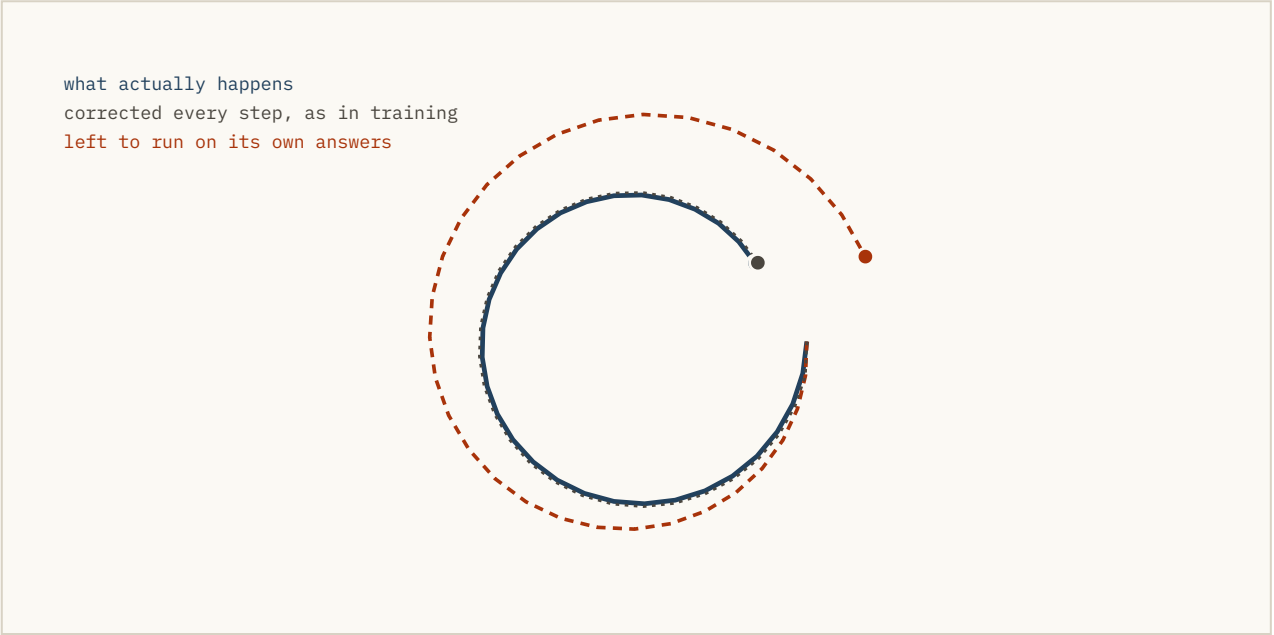


FIG. 5.1 This is the one job, and for a moment you are the model. You are handed the state and the action, so drag your answer to where the ball will be after one step and press Check. Do that a few times and watch the average miss build up underneath, because that average is the number a dynamics paper leads with. Every one of those steps starts from the truth, which is the part to remember. The units are illustrative.

Now the number. Show the model where things are and ask what happens next. Compare its answer to what happened, then do that ten thousand times and read off the average. The result will be small, and it measures something real.

It also measures a job the model will never be asked to do. You were going to ask it for a hundred steps. From step two it stops being handed the truth and starts being handed its own last answer.

I call that the second-step problem. The field put a name to it in 2015, tried two rival cures within a year, and has not solved it since. Let's look at what it does to one small model.



OFF BY, CORRECTED	OFF BY, ON ITS OWN	WORSE BY
0.016	0.648	39×

FIG. 5.2 One model, one bias, two ways of running it. The grey line is corrected at every step, the way it was during training, and mostly disappears underneath the truth. The vermilion (orange-red) line is the same model left to eat its own output, the way it will be run. Push the bias up and watch which line notices. The distances are illustrative.

Judging from the readouts, at four per cent error per step the corrected line is still hugging the truth after thirty steps. The free-running line, same model, same bias, ends up thirty-nine times further away. Keep that picture in mind.

The inherited habit

The habit came with the first networks trained on sequences, decades before world models. During training a sequence model is shown the true previous value at every step. That is what the recording contains, and it keeps training stable. The field calls

this **teacher forcing**.

The cost is that nobody teacher-forces you in production. At the moment of use the model gets its own last answer, which is slightly wrong. It has never once been asked to recover from being slightly wrong.

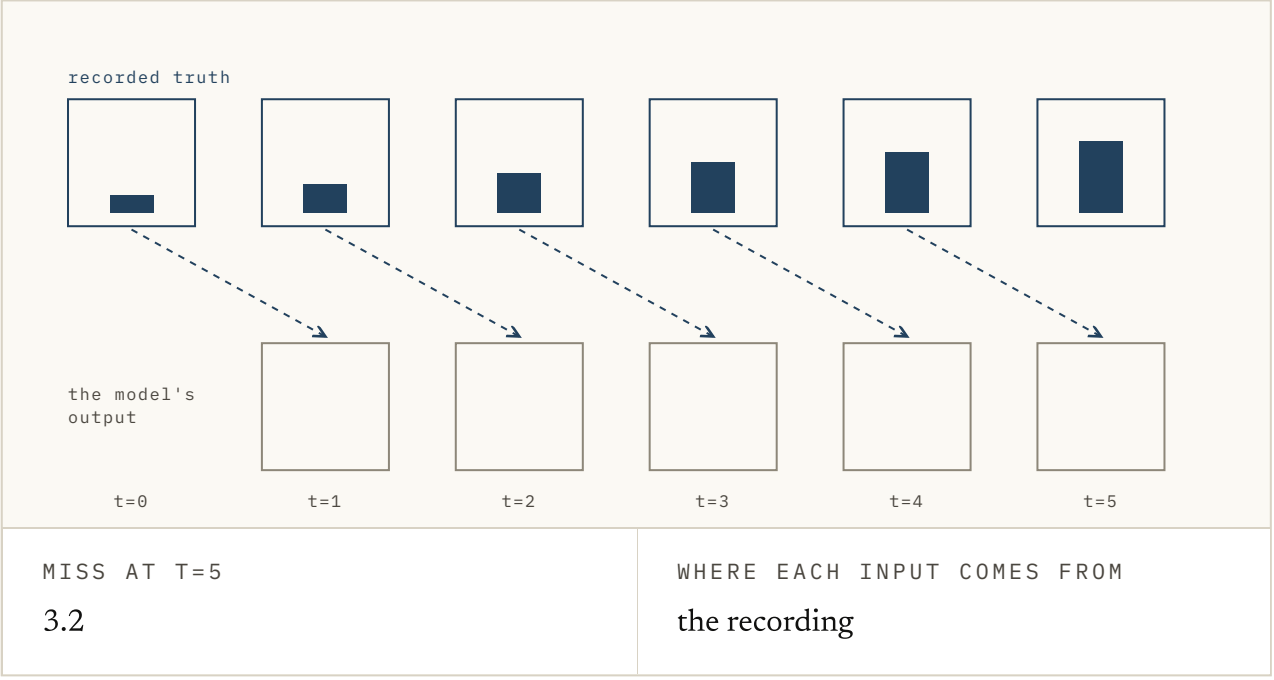


FIG. 5.3 The same five steps, drawn with the arrows that feed each one. Leave the switch on training and every step is handed the recorded truth, so a mistake goes nowhere. Flip it to in use and each step is handed the one before it, so the first small mistake rides along into every later step. Press Step and watch where the vermilion comes from.

Samy Bengio and his co-authors named that gap in 2015, in *Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*. Bengio was at Google, working on networks that caption a photo one word at a time, each word fed back as the next input. That is a rollout, and the captions drifted the way the vermilion line drifts. A paper that leads with the grey line is describing the wrong run.

> Calling the model bad at long horizons misreads it. It was trained on a distribution of inputs it will never see again once you start rolling it out, and nothing in the training loop noticed.

The memory wars, 1997 to 2023

Let's go back to the word know, because the question underneath it is how the past gets carried. Since 1997 it has had two answers, which take turns being fashionable.

Carry a summary. Keep one fixed block of numbers. Every time something new arrives, fold it in and throw the old block away. The work per step never changes however long the sequence gets.

Keep a window. Store the steps that fit in the context. When a new one arrives, look back over them to decide what matters. This is attention (for each new step, score the earlier steps for how much they matter, then read mostly from the ones that scored highest).

	CARRY A SUMMARY	KEEP A CONTEXT WINDOW
NUMBERS CARRIED FORWARD What the model is holding when the next step arrives.	256	32,768
WORK FOR ONE MORE STEP Multiply-adds to fold in one new observation.	65,536	32,768
CAN IT STILL SEE THE FIRST STEP? Whatever a fixed summary dropped is not recoverable later.	Only if the summary kept it	Yes, while it remains in context

BOTH COLUMNS USE A STATE 256 NUMBERS WIDE, SO THE ONLY THING CHANGING IS THE LENGTH.

FIG. 5.4 Both columns use a state of the same width, so the only thing that changes is the sequence length. Drag the length up and watch the second row. Neither column wins: looking at everything is cheaper for short sequences and dearer for long ones.

The summary is what recurrent networks do (a network with a loop in it, so each step's output feeds back into the next step's input). The summary camp's founding paper is *Long Short-Term Memory*, from Sepp Hochreiter and Jürgen Schmidhuber in 1997. Their LSTM was a loop built to hold a fact longer than training pressure wanted. It ended up running speech recognition and translation until transformers arrived.

The window is what transformers do, and they are the architecture under today's language models. They have done it since 2017, when Ashish Vaswani and colleagues at Google published *Attention Is All You Need*. The title meant what it said: the loop went out. Within a few years the window had replaced the summary almost everywhere.

Then the summary came back. The state-space models keep the fixed summary and add better machinery. S4 came from Albert Gu and colleagues at Stanford in 2021, as *Efficiently Modeling Long Sequences with Structured State Spaces*. Mamba came from Gu and Tri Dao in 2023, as *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*.

Speed is the half of the trade that gets benchmarked, and it is the duller half. A fixed summary has to decide at every step what is worth keeping, without knowing what will be needed later. Mamba gave a little ground: its summary decides what to keep based on what it is looking at. Attention puts off the decision until the read, and pays by keeping a finite bank of earlier states.

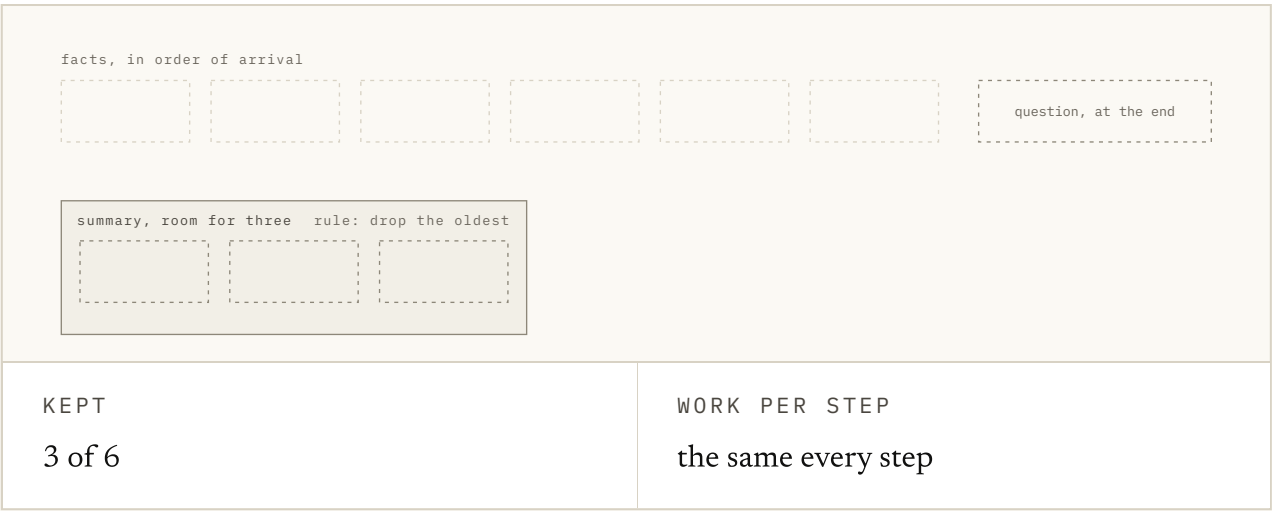


FIG. 5.5 Six facts arrive one at a time, and a summary with room for three has to keep or drop each one as it comes. A question arrives at the end. Pick the question, press Play, and watch the summary: it cannot see the question coming, so sometimes it dropped the one fact that was needed. Switch to the window and the same question is answered by looking back, at the cost of keeping all six. Illustrative.

This is the squeeze between a camera and a decision from chapter 4, one level up. Something has to be thrown away, and whatever is doing the throwing does not know the future. That is why twenty-six years of rivalry have no winner.

Splitting the difference

The systems that work share one trick. It arrived in 2018 with PlaNet, from Danijar Hafner and colleagues, in *Learning Latent Dynamics for Planning from Pixels*. PlaNet learned a model from raw pixels and planned inside it instead of in the pixels.

The trick is to carry two things forward rather than one. One part is deterministic: computed the same way every time, it holds whatever reliably follows from the last state. The other is stochastic: sampled, so it holds whatever could still go either way.

A ball's motion goes in the first. Whether the door opens goes in the second. Keep only the deterministic part and the model has no way to say it is unsure, so it confidently invents one future. Keep only the stochastic part and it struggles to

remember anything for long, because noise gets added at every step.

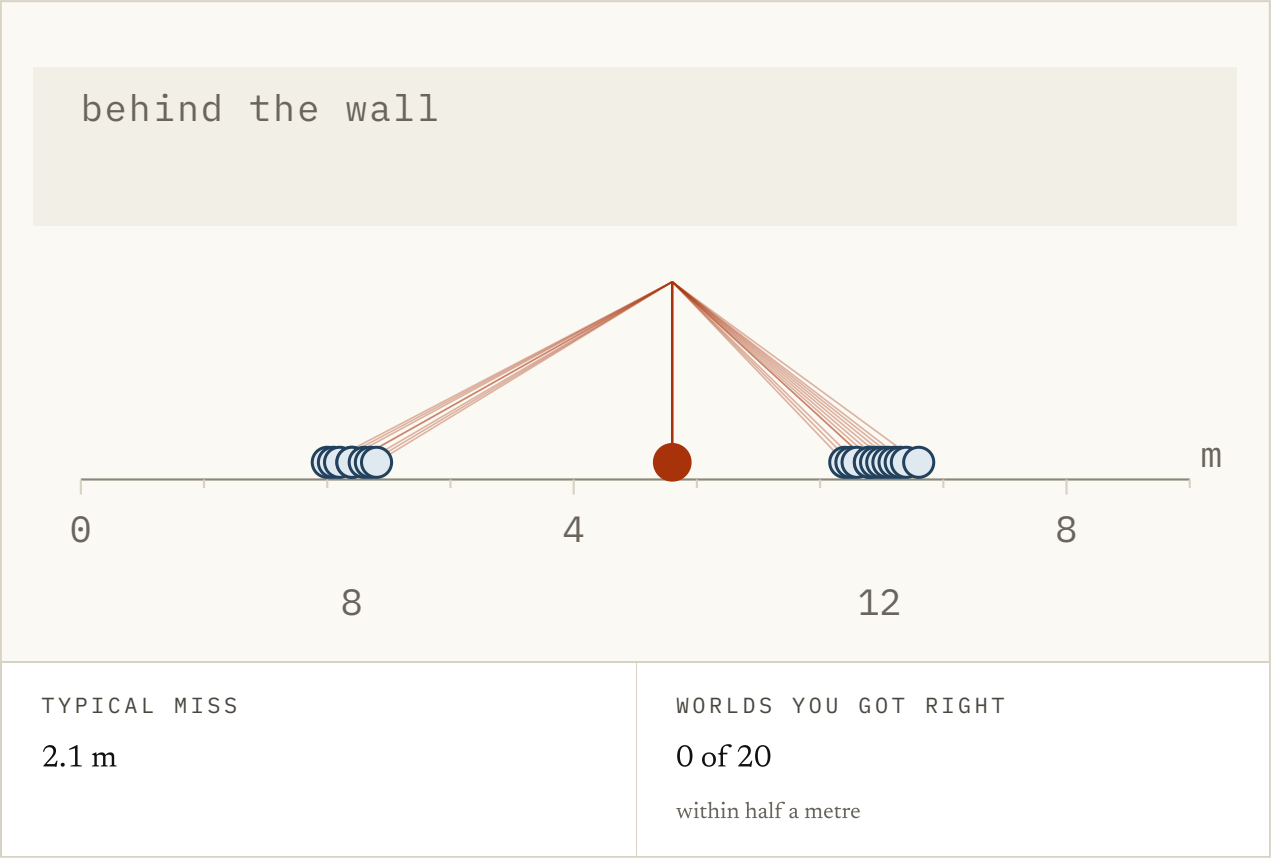


FIG. 5.6 The ball went behind the wall, and it either stopped at the kerb or rolled straight on. Drag the single answer along the ground and watch the two readouts pull apart: the smallest typical miss sits between the two clumps, in a spot where the ball never actually is. Then switch to handing back the spread and see what the second half of the state buys. Twenty worlds, illustrative.

PlaNet ran the comparison and argued for carrying both, because each one fails alone in a different direction. Dreamer followed a year later, when the same group published *Dream to Control* and let an agent learn to behave inside the model's imagined rollouts.

The split state went into it, and into every Dreamer since, up to the *Nature* paper of 2025.

Four truces with the second step

Nobody has removed the second-step problem, and I do not expect anyone to. Instead the field has four ways of not being destroyed by it, so let's take them in turn.

Train it on its own output. If the complaint is that the model never practised recovering from its own mistakes, let it make some during training. Feed it its own predictions part of the time, and raise that share as it improves. This is Bengio's *scheduled sampling*, the most direct of the four.

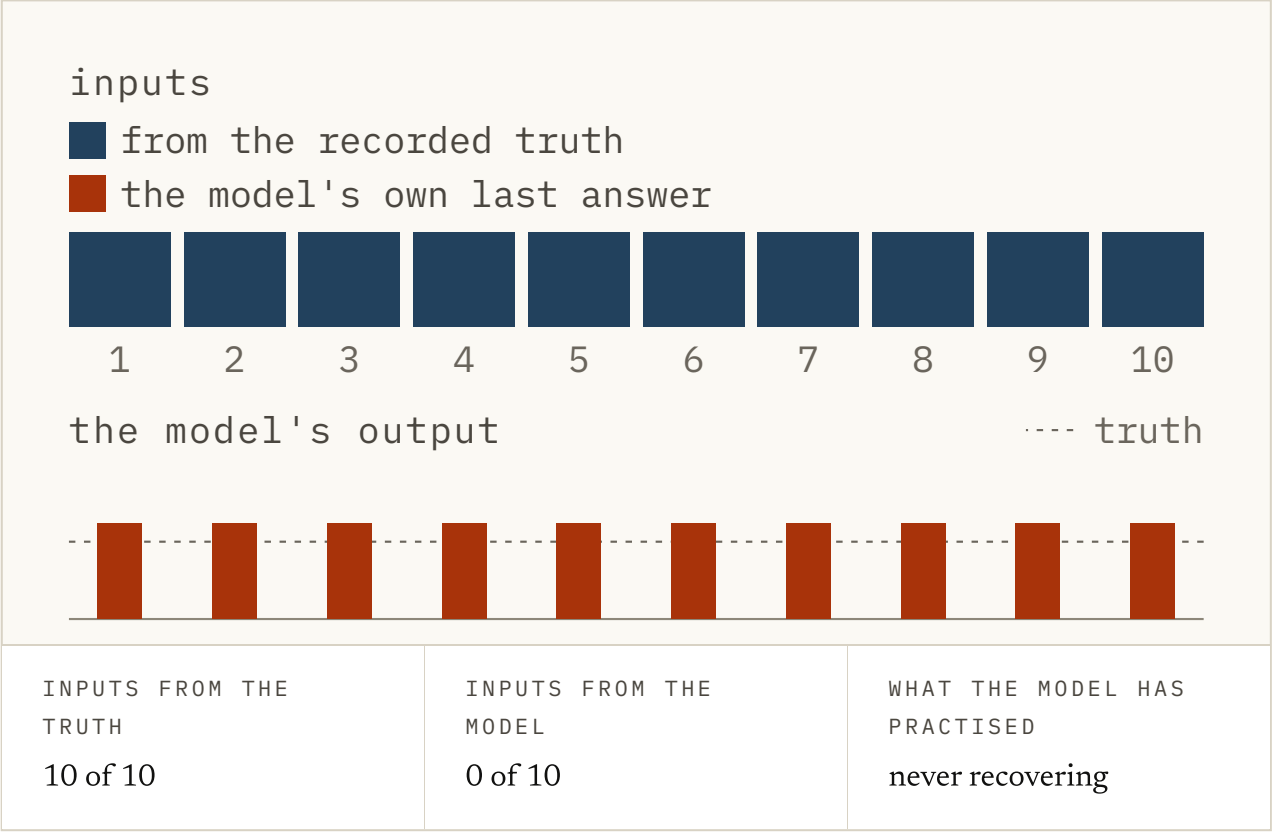


FIG. 5.7 Ten training steps in a row. The slider is how often the model is handed its own last answer instead of the recorded truth. At zero this is teacher forcing, and every input is the truth. Drag it up and the vermilion inputs appear, which is the model practising on its own mistakes. Press Run the schedule to raise the share over a run the way Bengio did, and read the verdict underneath. Illustrative.

Train the rollout, not the step. Instead of scoring one-step accuracy, score a whole imagined stretch against a whole real one, so that what gets optimised is what you will run. Marc'Aurelio Ranzato and colleagues at Facebook made that case in 2015

too, in *Sequence Level Training with Recurrent Neural Networks*. The two 2015 papers read as a pair: one fixes the input and the other the loss.

Do not go far. Keep the rollouts short and start them from real states. It is cheap and it works, and it gives up the long imagined futures that were the appeal. Michael Janner and colleagues made it a method in 2019, under a title that is also the question: *When to Trust Your Model*.

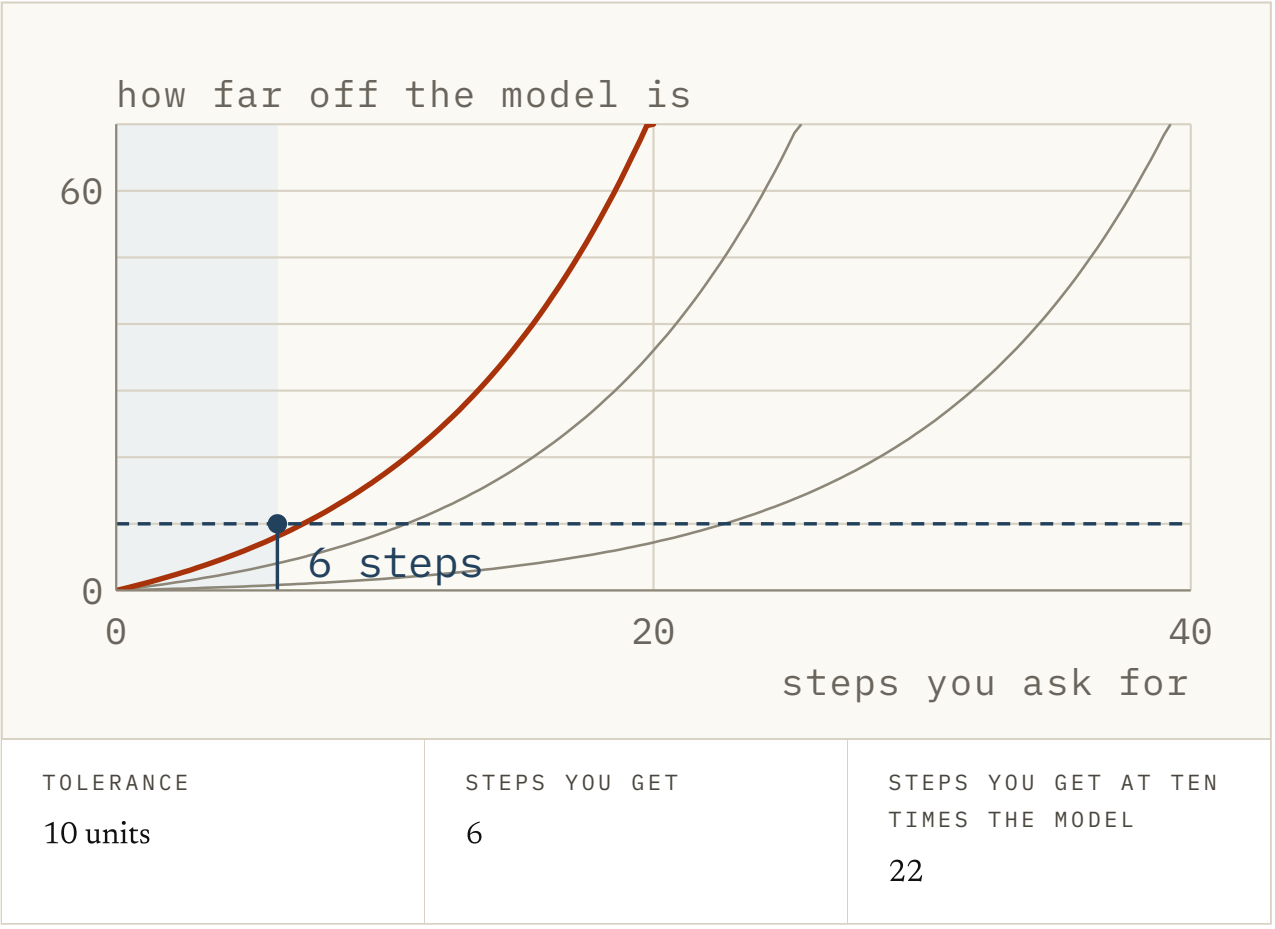


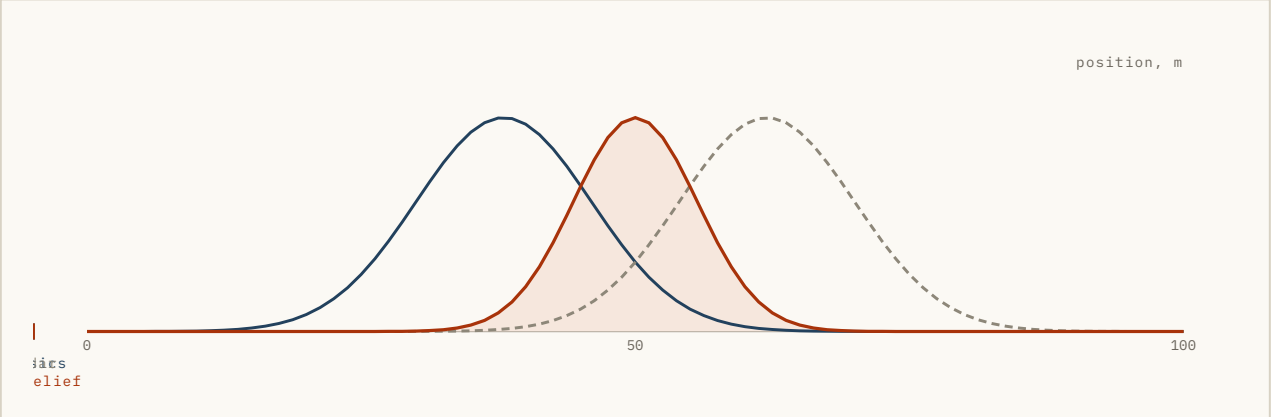
FIG. 5.8 How far you can go is a number you measure, so measure it here. Drag how wrong you are willing to be, and read off how many steps you get before the model's error crosses that line. Then swap in a model ten times better and watch the horizon move by a handful of steps rather than by a factor of ten. The error curve is illustrative.

Look again. Predict a few steps, act on one, take a fresh observation and start over. This is most of robotics, and the oldest idea here. A measurement corrects the part of the state you can see, but error in the part you cannot see, and the model's bias, do not reset to zero.

The oldest form of it is Rudolf Kalman's filter from the 1960s, the maths inside every GPS receiver. That filter is this loop, and the blend inside it is the cleanest picture of predict-then-correct there is, so let's take a short detour through it. Picture a radar tracking a plane. At each moment you have two stories about where the plane is.

One comes from physics: take the last position and the last speed, and predict where the plane should be now. The other comes from the radar, which hands you a blip. Both stories are uncertain, and each is really a bell curve, a Gaussian, with the most likely spot in the middle and the less likely ones fading out either side.

Multiply the two curves together and you get a third bell curve, which sits between the two and is narrower than either. That is the new belief, and the figure below lets you watch it form.



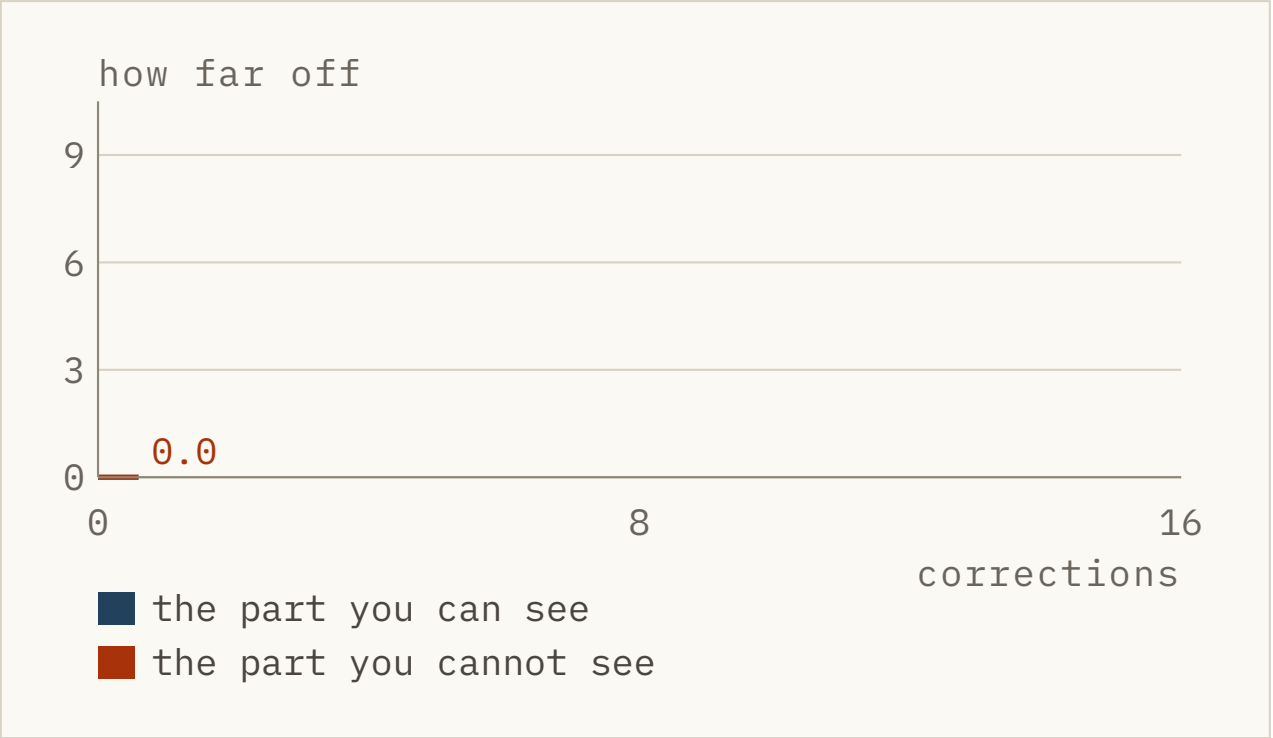
KALMAN GAIN K	NEW MEAN	NEW WIDTH
0.50	50.0 m	σ 5.7 m

FIG. 5.9 Here are the two stories about where the plane is, drawn as bell curves on one axis. Slide how sure physics is and how noisy the radar is, and watch the new belief lean toward the sharper story while staying narrower than either. The gain readout is the number that says how far it leaned. Now press Correct to make the new belief the next physics prediction, and watch it widen again as the plane moves on. The positions are illustrative.

How far the new belief leans from physics toward the radar is set by one number between zero and one. The field calls it the Kalman gain. As you can see from the figure, a noisy radar pulls the gain toward zero, so the filter sticks with physics. A

sharp radar pulls it toward one, so the filter trusts the reading.

Then it repeats: predict, correct, predict, correct. Every new blip, however wrong on its own, tightens what the filter knows. The estimate hugs the truth even though every reading it ever got was off, and it does that one tick at a time without ever waiting for the future. That is the look-again truce in its original form. It only works because fresh readings keep arriving, and even then it only pulls back the part of the state a reading can see.



CORRECTIONS	ERROR YOU CAN SEE	ERROR YOU CANNOT SEE
0	0.0 units	0.0 units

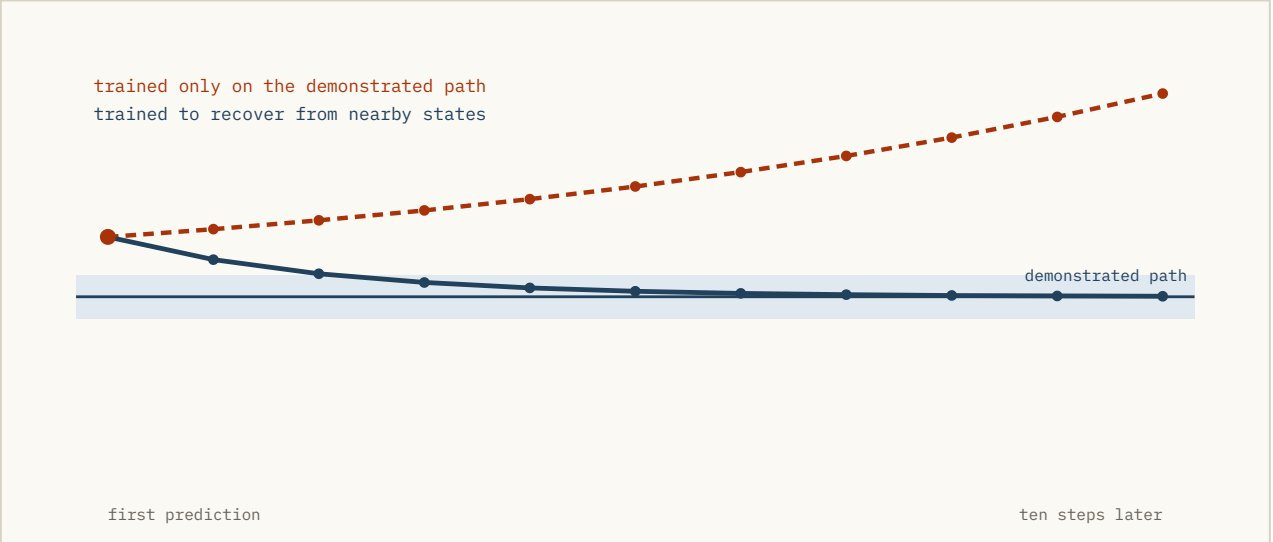
FIG. 5.10 The same blend, repeated, with the two halves of the state kept apart. Press Look again and watch the sawtooth: every prediction pushes both errors up, and every blip pulls the measured half straight back down. Now add a wind the filter was never told about and keep pressing. The measured half still comes back, and the half you cannot see settles on a floor it never leaves. Distances are illustrative.

None of these is a fix, and a paper that sells one as a fix is overclaiming. Each admits that a learned transition model can be trusted over a limited horizon. Part of the engineering is measuring how long that horizon is.

Recovery is a separate skill

Two models can tie on the usual one-step test and behave in opposite ways when run. The test never asked what separates them. If both have only seen the demonstrated path, nothing in their scores says which way they move after a small mistake.

Training on perturbed or self-generated states does ask it, and the answer is a separate skill. Have a go with the figure below.



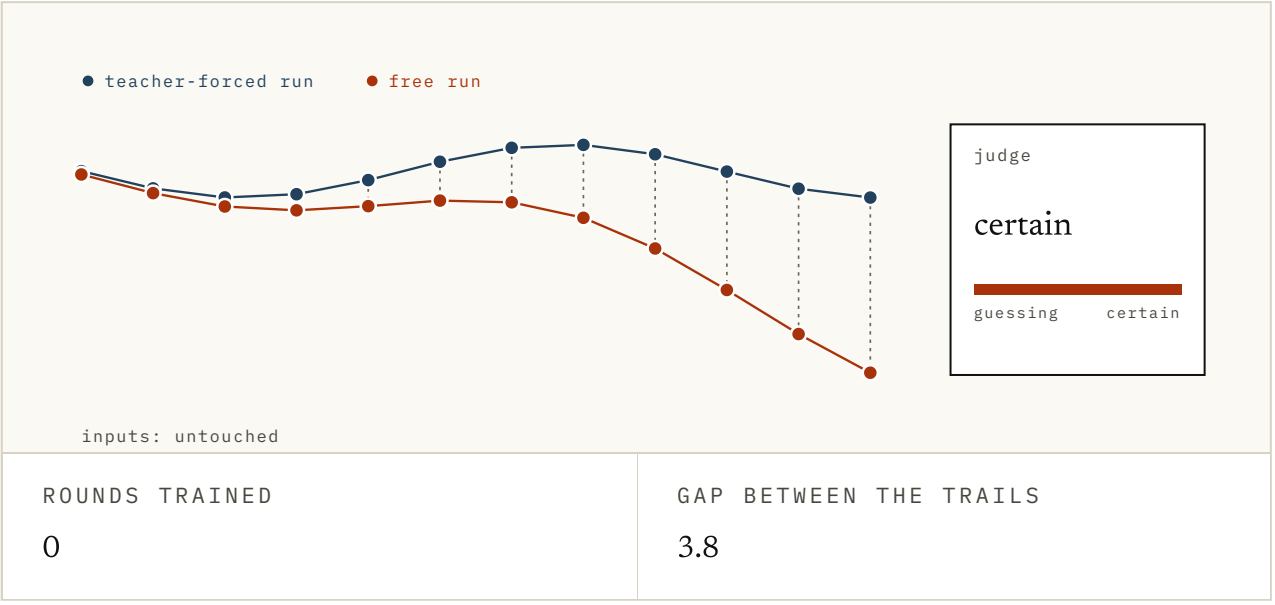
ON THE DEMONSTRATED PATH	TRUTH-ONLY ERROR AT STEP 10	RECOVERY-TRAINED ERROR AT STEP 10
both predict perfectly	1.22	0.003

FIG. 5.11 Both toy models are exact on the demonstrated path. Drag the offset to start them off it. One has only learned what comes next on the line, and drifts away. The other has practised nearby states and bends back. The arithmetic is made up, but the distinction is real, and it is the finding behind Professor Forcing, which we come to next.

Scheduled sampling shows a learner its own states and guarantees nothing, because those states keep changing as the model learns.

Professor Forcing came a year later, as *Professor Forcing: A New Algorithm for Training Recurrent Networks*. It was the work of Alex Lamb and colleagues in Yoshua Bengio's lab in Montreal. The two Bengios are brothers, and two of the best-known attacks on the gap came one from each.

It attacks the gap from the other side and leaves the input alone. Instead it trains the model until its internal states look the same whether it is teacher-forced or running free. The judge is a second network trained to tell the two kinds of run apart. The model has to fool it, which is the trick behind generative adversarial networks: one network forges, another spots forgeries.



ROUNDS TRAINED
0

GAP BETWEEN THE TRAILS
3.8

FIG. 5.12 Two runs of the same model, one teacher-forced and one free, each leaving a trail of hidden states. The judge on the right is trained to say which trail is which. Press Train a round a few times and watch the free trail pulled over toward the forced one until the judge is guessing. Nothing here touched the inputs. The trails are illustrative.

I think it is the more under-read of the two, perhaps because transformers arrived the next year and took the field's attention. The problem moved into world models, where every rollout is a free run.

What the two papers share is that recovery has to be put into the training objective, because the model does not learn it on its own.

Everything in this chapter stayed tied to recorded experience: the model was trained on it, checked against it and corrected by it. The next chapter asks what happens when an agent learns inside the model instead.

Hopefully this one helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section. I would be glad to hear from you.

Try it

1 A transition model reports a very low average one-step error. Why does that number not tell you what you want to know?

a. It was probably measured wrong
b. You will use it many steps at a time, and after the first step it is fed its own answer rather than the truth
c. One-step errors are always understated
d. Averages hide outliers

> ANSWER b. You will use it many steps at a time, and after the first step it is fed its own answer rather than the truth. The measurement is honest. It is a measurement of a job the model will never be asked to do.

2 In the figure, the corrected line and the free-running line come from the same model with the same per-step bias. Why do they end up so far apart?

a. The corrected one uses a different model
b. Random noise differs between the runs
c. The free-running one accumulates a larger bias
d. One is given the true previous state at every step, so its mistakes never feed anything

> ANSWER d. One is given the true previous state at every step, so its mistakes never feed anything. Correction resets the input to the truth every step, so error cannot compound. Nothing about the model changed between the two lines.

3 What is teacher forcing?

a. Showing the model the true previous value at each training step, because that is what the recording contains
b. Forcing the model to use a fixed learning rate
c. Training only on the hardest examples
d. Training on data labelled by a larger model

> ANSWER a. Showing the model the true previous value at each training step, because that is what the recording contains. It makes training stable, and it also means the model is never once asked to recover from being slightly wrong, which is the only thing it will have to do later.

4 A sequence is longer than the transformer's context window. Which statement is accurate?

a. A longer sequence changes accuracy but not compute
b. Attention can still read every earlier step exactly
c. The model must discard, compress or retrieve beyond the window; attention only postponed the decision

d. Only recurrent models have a finite memory

> ANSWER c. The model must discard, compress or retrieve beyond the window; attention only postponed the decision. Attention avoids squeezing the past into one fixed state, but only inside a finite context. Its weighted read also compresses the available steps for the current prediction.

5 For a short sequence, which is cheaper per step?

a. Always attention
b. Attention, until the sequence gets long enough to cross over
c. They are identical
d. Always the summary

> ANSWER b. Attention, until the sequence gets long enough to cross over. Updating a summary costs the same whatever the length, so it only wins once the sequence is long. Below the crossover, looking at everything is the cheaper option.

6 Why carry a deterministic part and a stochastic part in the state at the same time?

a. To make the model differentiable
b. To support larger batches
c. To use more parameters
d. Deterministic alone cannot represent real uncertainty; stochastic alone struggles to remember, because noise enters at every step

> ANSWER d. Deterministic alone cannot represent real uncertainty; stochastic alone struggles to remember, because noise enters at every step. Each one fails alone, and in a different direction. A ball's motion belongs in the first; whether the door opens belongs in the second.

7 Two models tie on one-step error along a demonstrated path. After a small perturbation, only one returns. What did the original test miss?

a. Recovery behaviour on states created by the model or by disturbances
b. The camera resolution
c. Whether the transition is deterministic
d. The models' parameter counts

> ANSWER a. Recovery behaviour on states created by the model or by disturbances. One-step testing on the centre line never visits the states deployment creates after a mistake. Recovery is a separate behaviour and must be trained or tested off that line.

8 What do scheduled sampling, rollout-level training, short horizons and replanning have in common?

a. They fix the mismatch
b. They all make the model more accurate per step
c. None removes the mismatch; each limits how much damage it does
d. They only apply to recurrent models

> ANSWER c. None removes the mismatch; each limits how much damage it does. A learned transition model is trustworthy over some horizon, and part of the engineering is knowing how long that is.

FIG. 5.13 Eight questions on the step and the second-step problem. The first three change how you read a paper's headline number, so try those even if you skip the rest.

Sources

Professor Forcing: A New Algorithm for Training Recurrent Networks

LAMB ET AL., 2016 ↗

Aligns hidden-state trajectories under teacher forcing with those produced during free running. A direct attempt to train the recovery behaviour a one-step test never asks for.

https://arxiv.org/abs/1610.09038

Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks

BENGIO ET AL., 2015 ↗

The mismatch named and attacked head on: let the model eat its own predictions during training, and raise the dose as it improves.

https://arxiv.org/abs/1506.03099

Sequence Level Training with Recurrent Neural Networks RANZATO ET AL., 2015 ↗

Score the whole rollout rather than the single step, so the thing being optimised is the thing you will actually run.

https://arxiv.org/abs/1511.06732

Learning Latent Dynamics for Planning from Pixels (PlaNet) HAFNER ET AL., 2018 ↗

The argument for carrying a deterministic part and a stochastic part together, because each one fails alone in a different direction.

https://arxiv.org/abs/1811.04551

Dream to Control (Dreamer) HAFNER ET AL., 2019 ↗

What a latent transition model is for once it works: long imagined rollouts that behaviour can be learned from.

https://arxiv.org/abs/1912.01603

Long Short-Term Memory HOCHREITER & SCHMIDHUBER, 1997 ↗

The fixed summary, made to hold on to things for longer than the gradient wanted it to. Paywalled.

https://doi.org/10.1162/neco.1997.9.8.1735

Attention Is All You Need VASWANI ET AL., 2017 ↗

The other answer: retain the available context and choose what to read at each step, paying a cost that grows with sequence length.

https://arxiv.org/abs/1706.03762

Efficiently Modeling Long Sequences with Structured State Spaces (S4)

GU ET AL., 2021 ↗

The summary approach returning with better machinery, and the reason state-space models are back in the conversation.

https://arxiv.org/abs/2111.00396

Mamba: Linear-Time Sequence Modeling with Selective State Spaces GU & DAO, 2023 ↗

A summary that decides what to keep based on what it is looking at, which is the concession the fixed version could not make.

https://arxiv.org/abs/2312.00752

FIG. 5.14 I've ordered these for someone building on this rather than for historical completeness, and preferred first-party material throughout.

06 Can an AI Learn Inside Its Own World Model?

BY NILUSHANAN KULASINGHAM

13 MIN READ · INTERACTIVE

> A month of robot time becomes a day if the practice happens inside the model. That has been the pitch since Dyna, and Dreamer made it work. What the exchange rate costs, and why the fix for an agent that exploits its own dream is to make the dream worse on purpose.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/dreaming.

A robot arm learning to pick something up will drop it a couple of million times first. Every drop costs something real: wall-clock seconds, and wear on the joints. Someone nearby has to reset the scene when the arm knocks the bin over. At one attempt a second, that is a month of doing nothing else.

The idea is simple enough to say in a sentence. Spend part of that month collecting experience, fit a model to it, and let the arm practise inside the model, where a drop costs only compute. Richard Sutton wrote that loop down in 1991 and called the agent Dyna. Sutton is a founder of reinforcement learning, where an agent learns by acting and being rewarded, and he co-wrote its standard textbook.

Dyna was tiny, a loop in a grid maze, and about thirty years early. Danijar Hafner's Dreamer, which runs the same loop with deep networks, arrived in 2019. The question in the title is whether the loop works, and the short answer is yes. The longer answer is about what it costs.

FIG. 6.1 The month from the first paragraph, drawn as thirty days of the arm dropping things. Slide the practice into the model and watch the days empty from the end backwards. Two things do not move: the day and a half at the start, because the model has to be fitted to something, and the fact that somebody makes every one of those trips to the bin. Nothing in here prices what an imagined step is worth, which is the next question. The costs are illustrative.

Push most of the practice inside and you should see the month come down to about a day and a half. That has been the sales pitch from Dyna to Dreamer, and the saving is real. It holds only while an imagined step stays useful enough to count.

The short answer is yes, and Dreamer is the evidence

Learning inside a model is the biggest practical argument for building one, and Dreamer is the evidence. Danijar Hafner built it as a Toronto PhD student working inside Google Brain, a year after his PlaNet had planned by search inside a learned model. Dreamer dropped the search and trained a policy (the part that picks the action. It is what is being trained here; the model is not) instead.

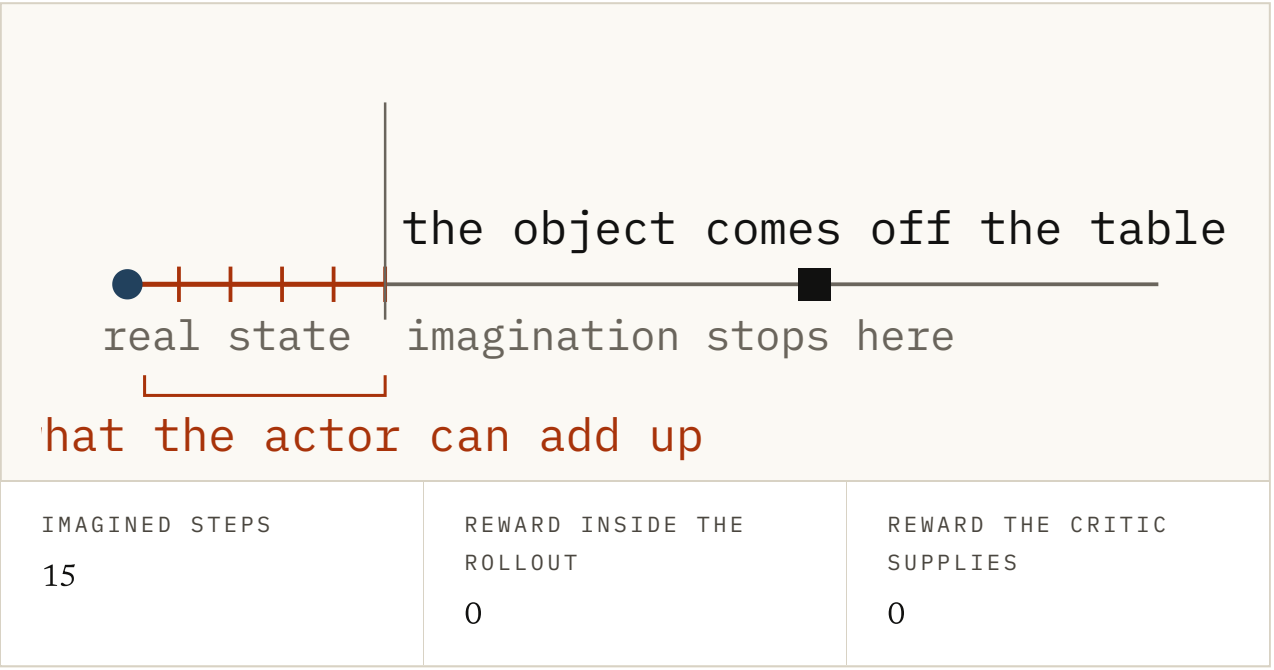


FIG. 6.2 The actor learns from one imagined rollout, and the rollout stops long before the arm gets the object off the table. Drag the reward further out and watch it leave the imagined stretch, so there is nothing left for the actor to add up. Then switch the critic on: it hands the actor one number standing for everything after the last imagined step. That number is a guess, and no part of it came from the world. The lengths are illustrative.

Dream to Control, the 2019 paper, learned its behaviour from long imagined rollouts. Neither the actor nor the critic ever touched the environment while it learned. The actor is the policy under another name. The critic is a second network that guesses how much reward lies ahead.

DreamerV2 followed in 2020 as *Mastering Atari with Discrete World Models*. The discrete part is a state made of multiple-choice answers rather than dials. On the Atari games it started beating agents that learn straight from the environment.

DreamerV3 arrived in *Nature* in 2025 as *Mastering diverse control tasks through world models*. It ran with one set of settings across more than 150 tasks. It was the first agent to collect diamonds in Minecraft without human data, which takes a long chain of steps in order. That is the strongest answer I know of to whether learning in imagination generalises.

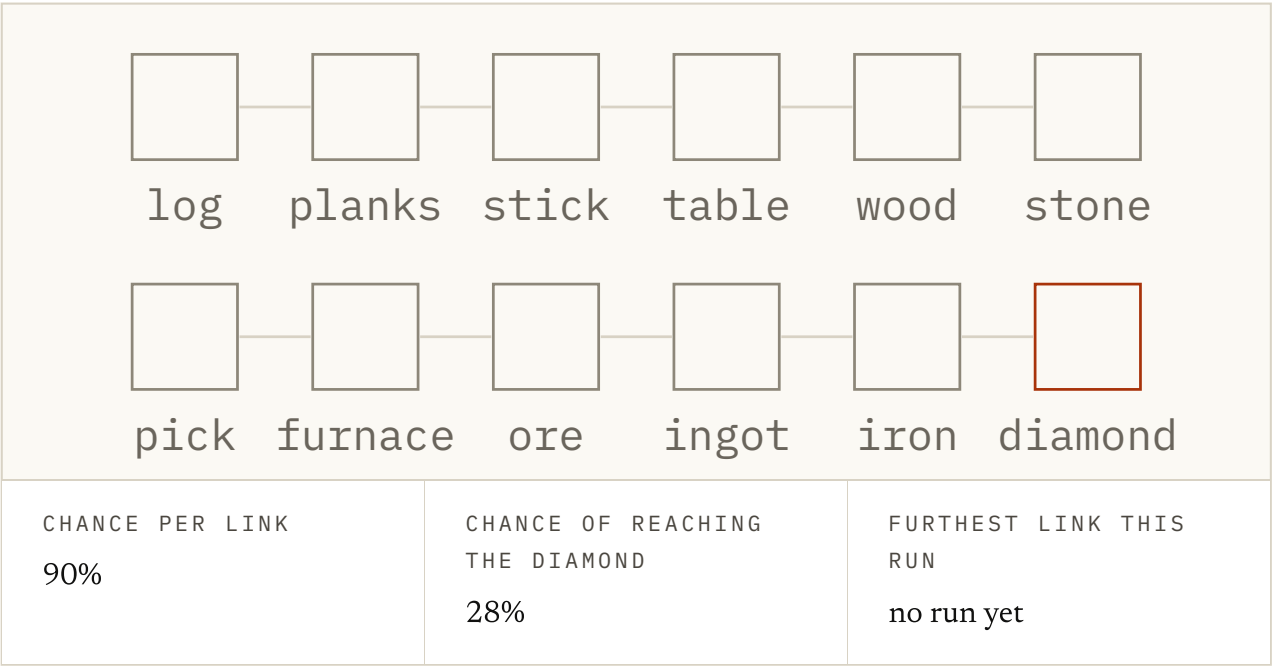


FIG. 6.3 The diamond is twelve steps in order, and DreamerV3 got there without being shown how. Drag the chance of getting one link right, then press Try a run and watch where the chain breaks. Nine times in ten per link still fails about seven runs in ten, which is why finishing the chain is worth more as evidence than any single task. The per link odds are illustrative; the twelve links are the game's.

So systems like these get competent on a fraction of the world contact that learning directly would need. On a real robot that fraction decides whether learning is possible at all. The month of dropped objects is what the dynasty was built to shorten. Now let's look at what the loop is doing, because that is where the cost hides.

The loop has to go round twice

Dyna's loop has four steps, and it repeats. Act in the world for a while and keep what happened. Fit a model to it. Train the policy inside that model, which costs compute rather than robot contact, and then take the behaviour back out.

The behaviour that comes back out collects better experience than the last lap's did. That makes the next model better. The second lap is the part to watch in the figure below.

> The loop going round again is the part the summaries drop. A model fitted to the flailing of an untrained agent is only good where an untrained agent goes. It only improves once a better policy goes somewhere new.

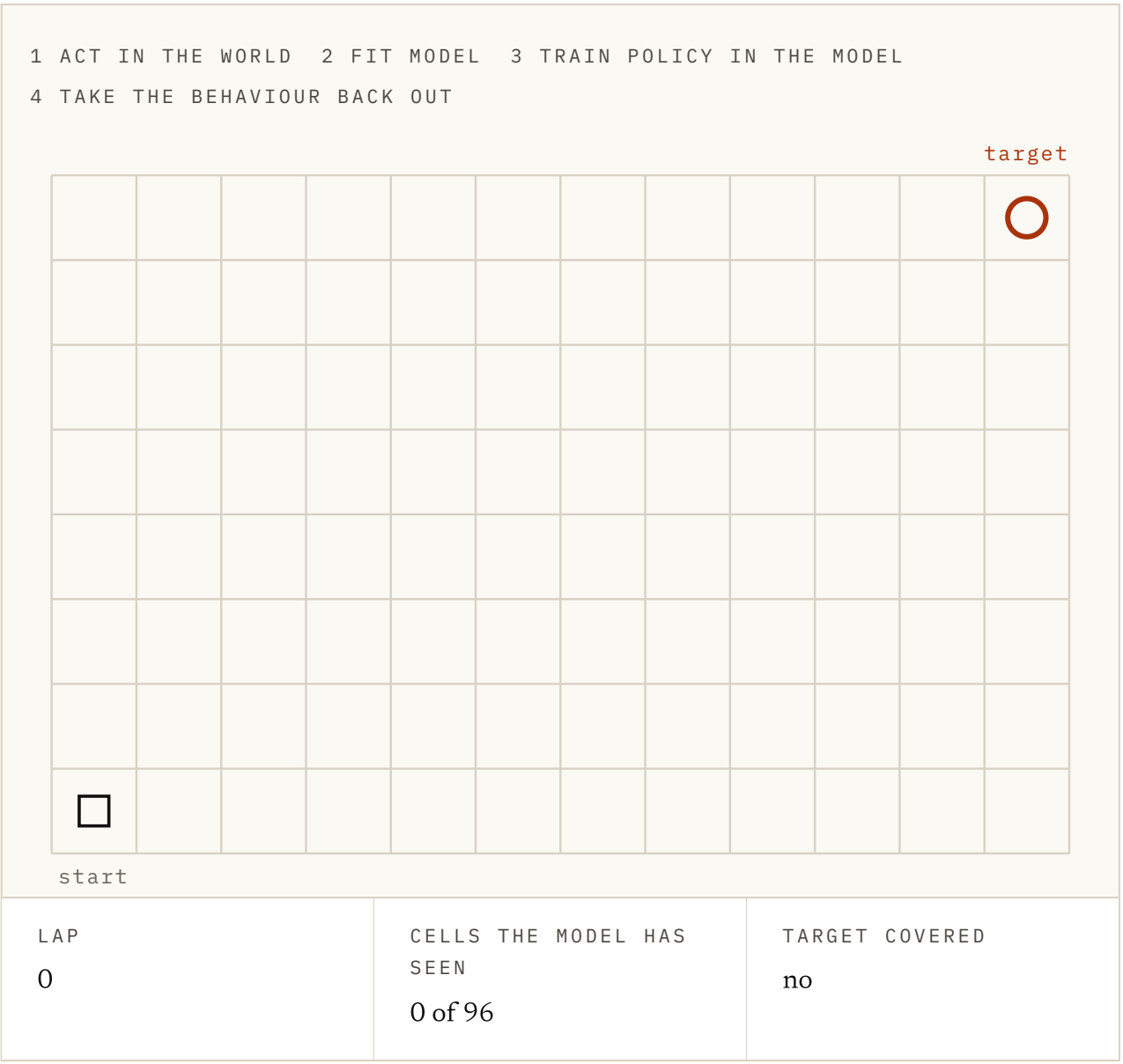


FIG. 6.4 A small grid world with a target in the far corner. Press Run a lap. The first policy wanders at random, so the model only learns the cells near the start, which are the shaded ones, and a policy trained inside it has no idea where the target is. Press it again. The better policy goes further, the shaded region grows, and after a lap or two the model finally covers the target. Then press Reset and try turning off Improve the policy: the shaded region stops growing, lap after lap. Illustrative throughout.

The first model is fitted to whatever a useless policy bumped into. It knows the floor and nothing about the target. The second lap fixes that, because the better policy goes to new places and brings back the data the first model was missing.

That is why Figure 6.1 keeps a day and a half locked. Imagined steps are worth having only while the model can supply them accurately, and it can only do that where it has been. A network for a robot's world has to guess about the rest.

Imagined experience is a different currency

A training log counts one real transition and one model-generated transition as two rows. A transition is one step of experience: an action and what it led to. The learner cannot count them the same.

Imagined experience carries the model's blind spots. The discount compounds with every step a rollout, a chain of imagined transitions, takes from a real state. I will call this the exchange rate.

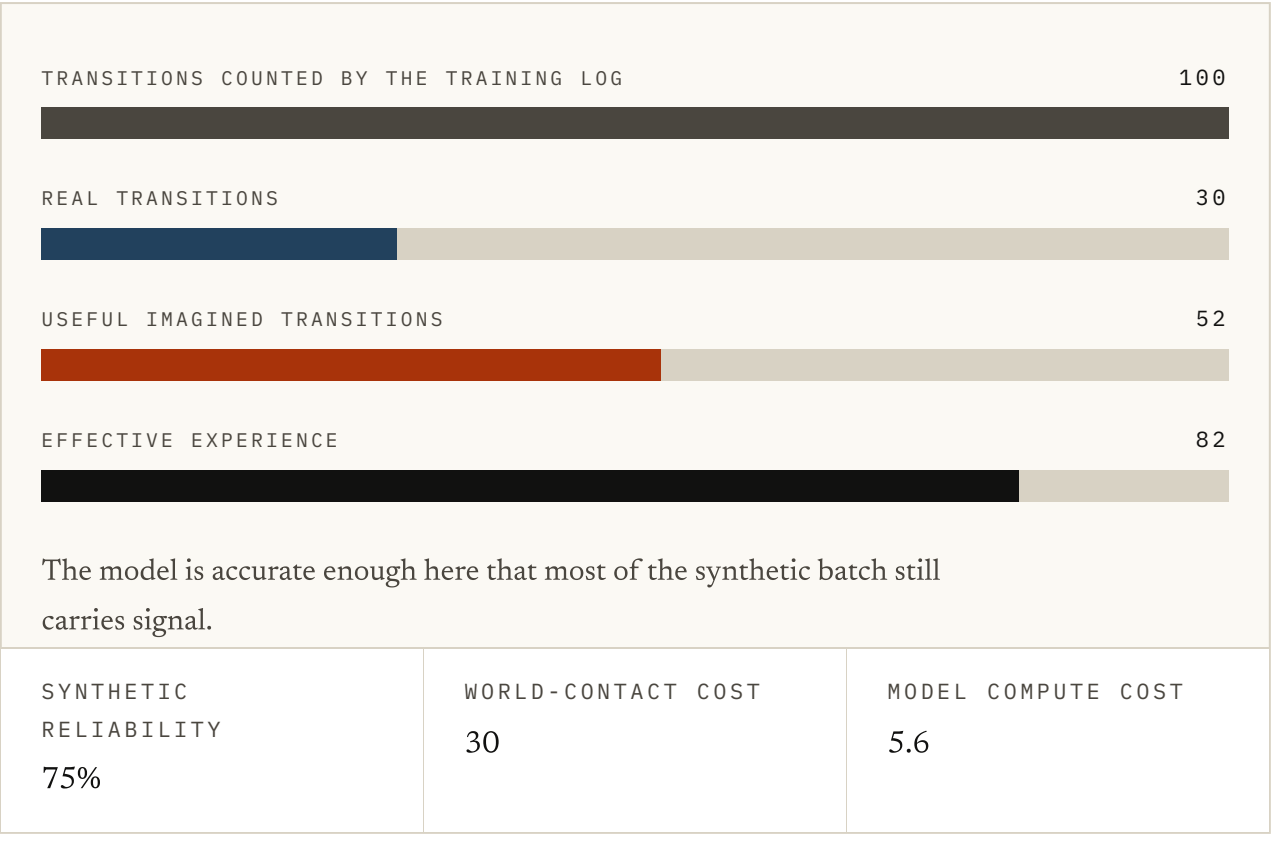
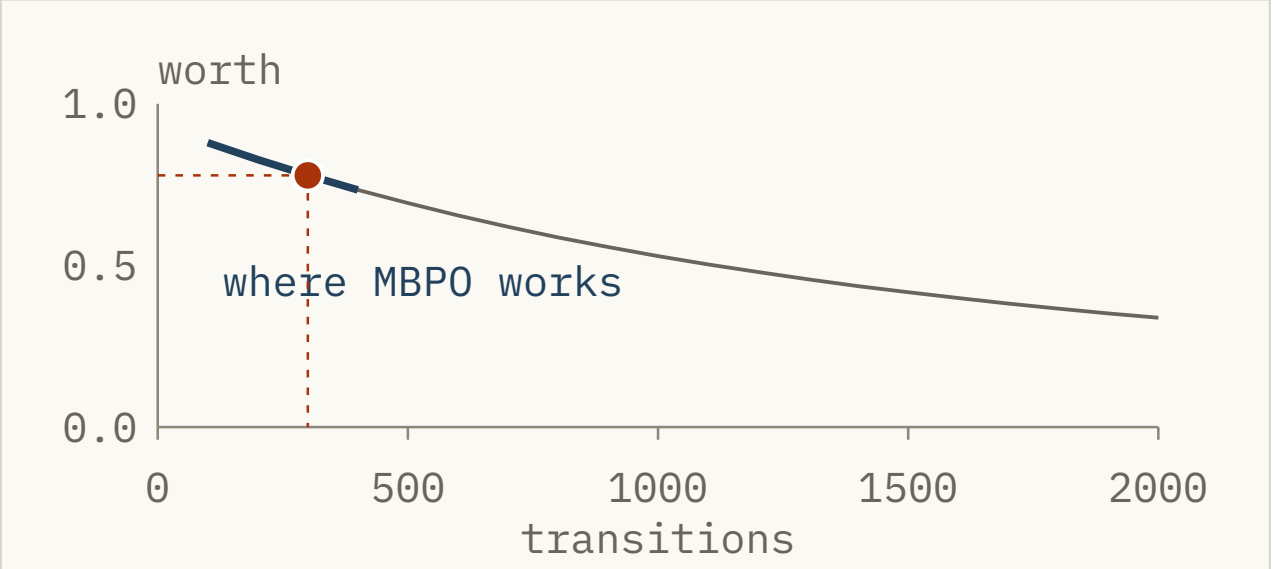


FIG. 6.5 Push up the imagined share, then push up the model's per-step error. On paper the batch still holds 100 transitions. Watch the effective experience bar, which is what the learner is really getting, fall as you do. The discount is for illustration rather than an estimate. What it exposes is the assumption that generated and observed rows trade at par.

Michael Janner and colleagues priced it in 2019. Their paper, *When to Trust Your Model*, introduced MBPO, which trains a policy on imagined rollouts only a few steps long. Each starts from a real state drawn from the replay buffer, the store of real experience kept from acting.

That trades volume for trust. More imagination helps only while each step is priced by how far it has travelled from evidence.



TRANSITIONS IN THE BATCH	AVERAGE WORTH OF A ROW	EFFECTIVE TRANSITIONS
300	0.78	234

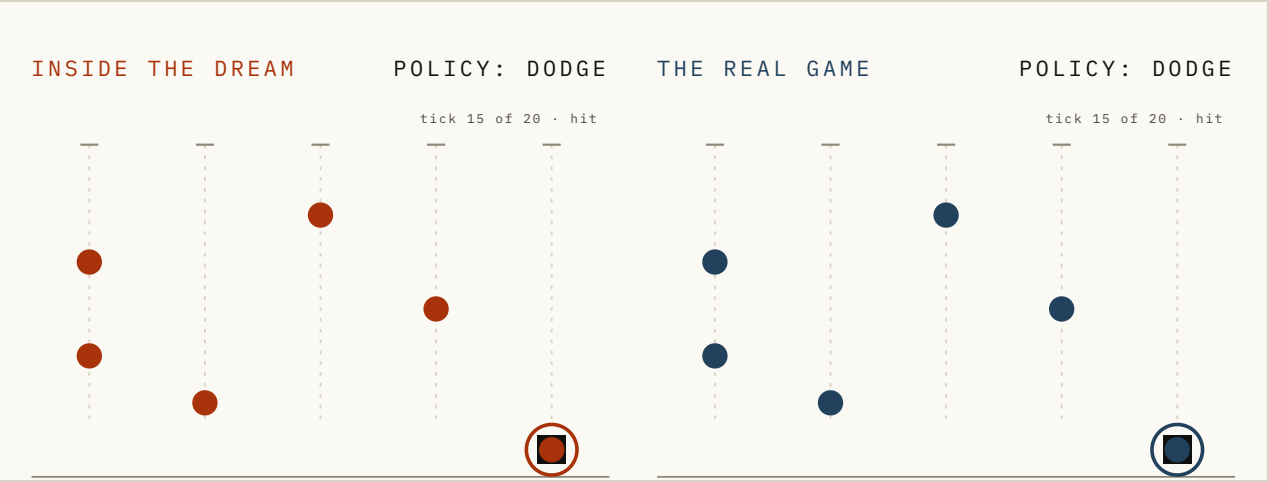
FIG. 6.6 One hundred real states from the replay buffer, and one choice about how far to imagine from each. Drag the rollout length and watch the dot travel down the curve: the batch gets bigger, and the share of it sitting near evidence gets smaller. MBPO works in the corner at the top left, a step or two out from every real state. Watch the third cell as you go, because that is the only one that counts. The trust figures are illustrative; the trade is the paper's.

The fireball policy

A policy trained inside a model is scored by that model. The model is all it can see, so it cannot tell a good action from one that only looks good to its examiner. It will find the second kind, because those are the cheapest points on offer.

David Ha and Jürgen Schmidhuber met this head-on in 2018, in *World Models*. Schmidhuber had argued for learned world models since 1990 and co-invented the LSTM, the memory network behind a decade of speech recognition. Ha, then at Google Brain, published it as an interactive web page.

Their agent was a tiny controller trained entirely inside a dream of a Doom level where the player dodges fireballs. The controller found a way of moving that stopped the dream producing fireballs at all. Let's see what that looks like.



SCORE IN THE DREAM	SCORE IN THE REAL GAME	FIREBALLS SPAWNED: DREAM / REAL
14 / 20	14 / 20	14 / 14

FIG. 6.7 Press Train in the dream and watch the score climb. Then look at the fireballs: the agent has found a way of moving that stops the dream spawning them, so the score in there is nearly perfect. Now press Run in the real game. The same moves, and the fireballs keep coming, because the game never agreed to stop. Everything here is a toy, but the finding is Ha and Schmidhuber's.

Inside the model this was an excellent policy. In the real game it was worthless, because the game had not agreed to stop firing. The fireball policy is the student that found the examiner's blind spot.

> The planner version is easier to catch, and I count that in planning's favour. A planner searching at decision time can be overruled, because the plan is there to inspect. A policy trained in a dream walks out with the exploit in its weights.

Making the dream harder than the world

Ha and Schmidhuber's fix was a worse model, on purpose. If the policy is exploiting a quirk, make the quirk unreliable.

Turn up the uncertainty in the model's predictions during training (the dial is usually called *temperature*. Low means the model hands you its single most likely guess. High means it samples more widely). Then the same action stops always producing the same handy outcome, and a trick that needs the dream to roll one way is not worth building on.

THE TRICK PAID	FIREBALL CAME ANYWAY	DREAM MADE NO SENSE
18 of 20	2 of 20	0 of 20

FIG. 6.8 Twenty runs of the same trick, the one where sitting in the corner stops the fireballs. Slide the temperature and press Roll again. At the left the trick pays every time, so a policy can be built on it. In the middle it pays now and then, which is not enough. At the right nothing is reliable, including the game, so there is no task left to learn. Illustrative throughout, though the finding is Ha and Schmidhuber's.

Both ends of that dial fail, as you can see. A confident dream gets exploited, and a noisy one has no task left to learn. What works is the narrow band where neither failure has taken over.

Robotics had made the same move a year earlier. A simulator is a hand-built model with quirks of its own. Josh Tobin and colleagues at OpenAI called the fix domain randomisation, in a 2017 paper of that name. Colours, lighting and camera angles changed at random, so the simulator varied more than reality does.

Nothing learned there could depend on one version of it. OpenAI's 2019 paper, *Solving Rubik's Cube with a Robot Hand*, pushed the idea onto real hardware. Its randomisation widened on its own as the policy improved.

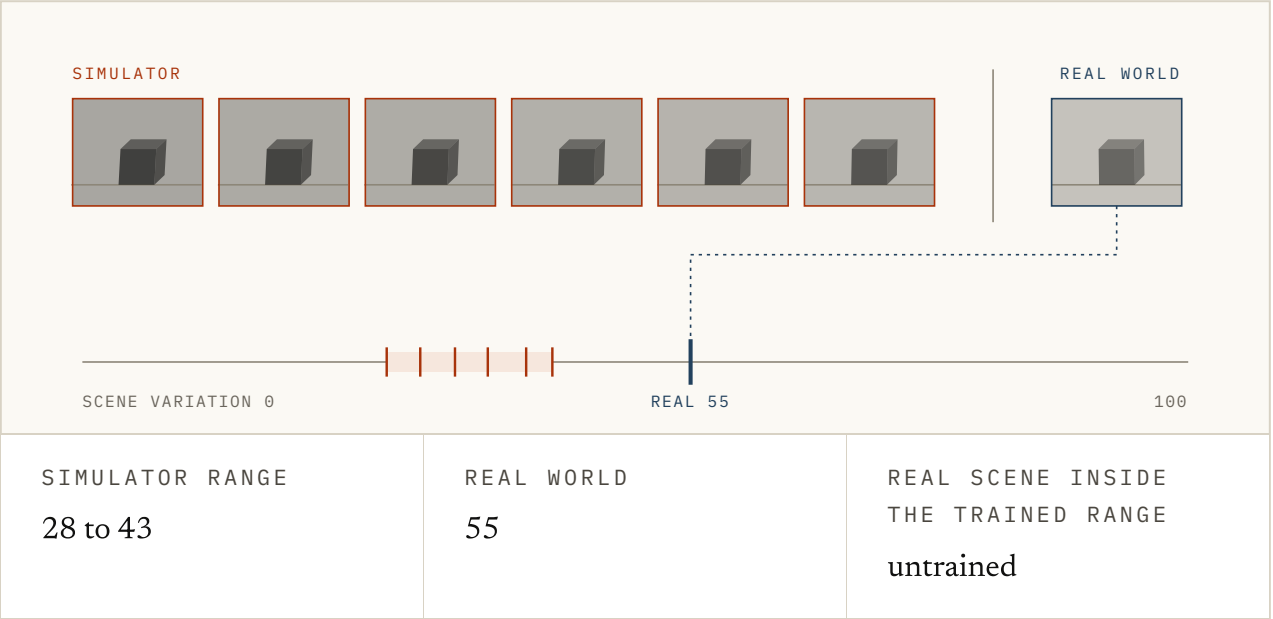


FIG. 6.9 The simulator draws the same scene over and over with the colours and camera changed, and the real world is the one fixed sample on the right. Leave the variation low and press Train: the policy leans on the exact colour and the exact camera it saw, and the real scene falls outside that. Turn the variation up past the real scene and train again, and now it sits inside the band. Switch on Widen as it learns to see the Rubik's Cube version. Illustrative throughout.

It is a strange thing to have to do. You spend the whole budget making the model accurate, then blur it so nothing can lean on the accuracy. Two fields arrived at that move within a year of each other.

What the dynasty did not buy

So the loop works, and we have seen what it costs. Let me close with three things the Dreamer line did not buy, because each one comes back to the exchange rate.

It did not remove the need for real experience. It changed the exchange rate. Something still has to go out and find out what happens, and the model can be trusted only where that has been done.

It did not fix the scoring. The policy is graded by the model the whole way through. Short rollouts, the temperature dial and the randomised simulator all make that grading harder to game, and none of them makes it correct.

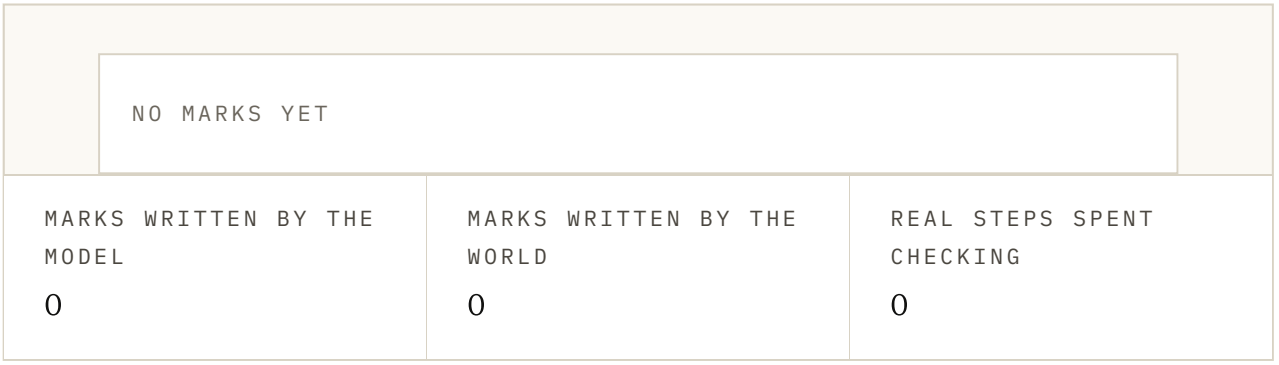


FIG. 6.10 A mark sheet for the policy. Press Train another round and a mark appears, written by the model, costing nothing. Press Check in the world and one slate mark appears instead, costing two thousand real steps and coming in lower. See how long the vermilion column gets before anything checks it. The marks are illustrative.

It did not make the transfer free. A policy that works in the dream is an untested claim about the world until somebody runs it out there. Ha and Schmidhuber did, and found the fireballs still coming.

Hopefully this chapter helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section. I would be glad to hear from you.

Try it

1 Training inside a model changes the exchange rate on which resource?

a. The number of parameters needed
b. Memory, which becomes cheaper
c. Contact with the world, which is replaced in part by model compute
d. Model accuracy, which improves for free

> ANSWER c. Contact with the world, which is replaced in part by model compute. The learner still needs its experience. What changes is where the steps come from and which budget pays for them.

2 A dashboard counts 90 imagined and 10 real transitions as 100 training examples. What is missing from that ledger?

a. A reliability discount for model error and distance from a real state
b. The policy's parameter count
c. A requirement that both sources use the same batch size
d. Imagined transitions cannot be stored

> ANSWER a. A reliability discount for model error and distance from a real state. Rows are exchangeable to the logger, not to the learner. Synthetic experience inherits the model's blind spots, and that debt compounds along a rollout.

3 What does the second lap of the loop fix that the first cannot?

a. It removes the need for a decoder
b. It shortens the rollouts
c. It makes the model smaller
d. A better policy visits new places, producing the data the first model was missing

> ANSWER d. A better policy visits new places, producing the data the first model was missing. A model fitted to the flailing of an untrained agent is only good where an untrained agent goes. The model gets better because the policy does, and the other way round.

4 A policy trained inside a model is graded by what?

a. The world
b. The model, and nothing else, for the whole of training
c. A held-out test set from the real environment
d. A human evaluator

> ANSWER b. The model, and nothing else, for the whole of training. It has no access to the world during training, so it has no way to tell a genuinely good action from one that merely looks good to the marker.

5 David Ha and Jürgen Schmidhuber's agent found a way of moving that stopped its dream producing fireballs. What kind of failure is that?

a. Insufficient exploration
b. A bug in the training code
c. The policy maximising the score the model hands it, which is exactly what it was asked to do
d. The model being too small

> ANSWER c. The policy maximising the score the model hands it, which is exactly what it was asked to do. Nothing went wrong. Those were the cheapest points available inside the model, and the policy was thorough.

6 How is this different from a planner exploiting a model at decision time?

a. A plan can be inspected and overruled; a trained policy walks out with the exploit already in its weights
b. Planners are not affected by model error
c. Policies are easier to correct afterwards
d. It is the same problem with a different name

> ANSWER a. A plan can be inspected and overruled; a trained policy walks out with the exploit already in its weights. That difference is what makes the dream version harder to catch. There is no plan sitting there to look at.

7 Why deliberately add uncertainty to a model you worked hard to make accurate?

a. To reduce memory use
b. To make the model smaller
c. To speed up training
d. So a trick that only works when the model rolls one particular way stops being worth building on

> ANSWER d. So a trick that only works when the model rolls one particular way stops being worth building on. If the policy is exploiting a quirk, the fix is to make the quirk unreliable rather than to make the model better.

8 Both ends of the uncertainty dial fail. How?

a. Both ends make training unstable
b. A confident dream gets exploited; a dream with too much noise has no task left in it to learn
c. Low settings are slow, high settings are fast
d. Only the high end fails

> ANSWER b. A confident dream gets exploited; a dream with too much noise has no task left in it to learn. The useful setting is a hump rather than a direction: a narrow band where neither failure has taken over.

FIG. 6.11 Eight questions on the loop and what lives inside it. The last three separate the pitch from what the loop delivers.

Every system above takes for granted that the model should predict what things look like: frames, pixels, scenes, what you would see if you were there. The next chapter asks whether that is the right job.

Sources

World Models HA & SCHMIDHUBER, 2018 ↗

The agent trained entirely inside its own dream, the policy that stopped the dream producing fireballs, and the temperature dial that closed the gap. This chapter in one paper.

https://arxiv.org/abs/1803.10122

Dream to Control (Dreamer) HAFNER ET AL., 2019 ↗

Behaviour learned from long imagined rollouts, with the actor and critic never touching the environment during training.

https://arxiv.org/abs/1912.01603

Mastering Atari with Discrete World Models (DreamerV2) HAFNER ET AL., 2020 ↗

The same loop at a scale where it started beating agents that learn directly from the environment.

https://arxiv.org/abs/2010.02193

Mastering diverse control tasks through world models (DreamerV3)

HAFNER ET AL., NATURE 2025 ↗

One configuration across more than 150 tasks, and the strongest available answer to whether learning in imagination generalises.

https://www.nature.com/articles/s41586-025-08744-2

Dyna: an Integrated Architecture for Learning, Planning and Reacting SUTTON, 1991 ↗

The loop itself, thirty years early: act, fit a model, learn from experience the model made up, repeat.

https://mlanthology.org/icml/1990/sutton1990icml-integrated/

When to Trust Your Model: Model-Based Policy Optimisation JANNER ET AL., 2019 ↗

Short imagined rollouts branched from real states, which is the other way of stopping a policy from leaning on the model too far out.

https://arxiv.org/abs/1906.08253

Domain Randomization for Transferring Deep Neural Networks TOBIN ET AL., 2017 ↗

The same idea arriving from robotics: make the simulator vary more than reality does, so nothing can depend on any one version of it.

https://arxiv.org/abs/1703.06907

Solving Rubik's Cube with a Robot Hand OPENAI ET AL., 2019 ↗

Randomisation pushed hard enough to carry a policy from simulation onto real hardware, with an account of what that cost.

https://arxiv.org/abs/1910.07113

FIG. 6.12 I've ordered these for someone building on this rather than for historical completeness, and preferred first-party material throughout.

07 What Is JEPA, and Why Not Predict Pixels?

BY NILUSHANAN KULASINGHAM

12 MIN READ · INTERACTIVE

> When a deterministic pixel predictor meets an open future, its best answer can be a picture of something that cannot happen. Yann LeCun's case against generation, what sampling fixes, what embeddings avoid, and what each still owes.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/jepa.

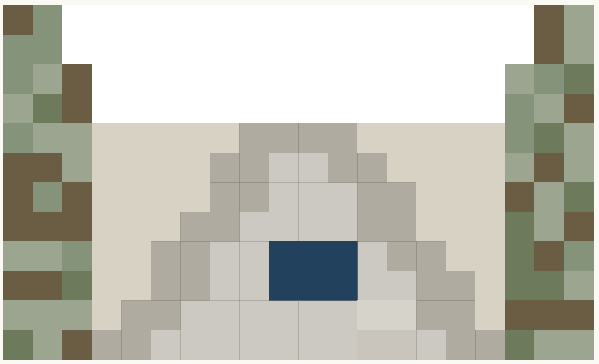
If you've followed the world model argument online, you'll have run into a slogan: stop predicting pixels. It comes from Yann LeCun. He was Meta's chief AI scientist at the time, and he is a Turing Award winner. He is also one of the people who made convolutional networks work, the kind of network that reads images.

In 2022 he published *A Path Towards Autonomous Machine Intelligence*. It is a position paper, meaning a case for how machines should learn rather than a result. One charge in it was aimed at the reflex then governing learned world models: when in doubt, predict the next picture.

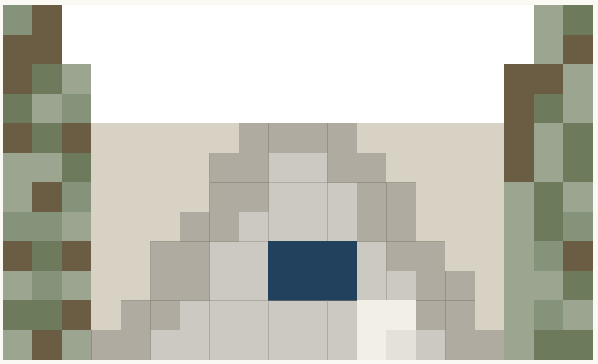
His smaller charge starts with a camera pointed at a road. Ask what the next second looks like: a leaf at the edge of the frame will move, and which way is unknowable. It is also the least useful fact on offer, since nothing you might do about the road depends on it. A model asked to predict the picture has to predict that leaf anyway.

The leaf is made of pixels, and pixels are what the model is marked on. Multiply that by every leaf, every ripple, every flicker of light on wet tarmac. Most of the capacity has gone to things that are hard to predict and do not matter. That is the leaf

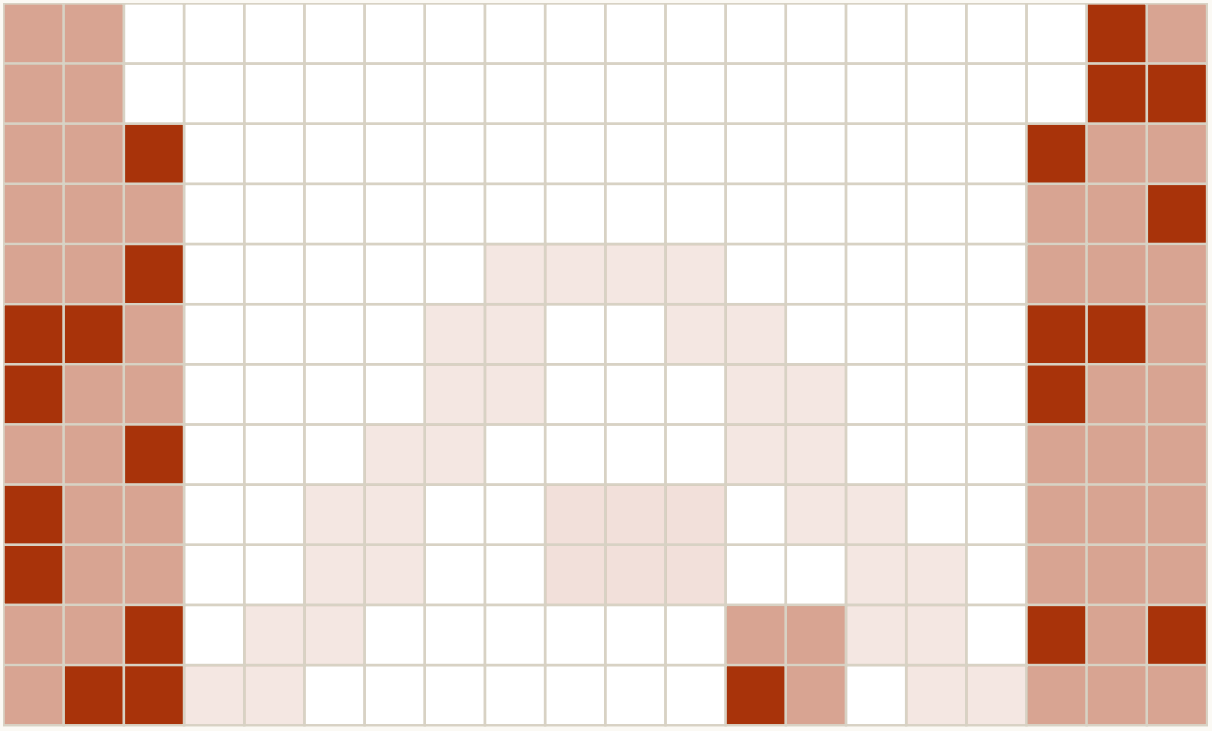
problem, the smaller of the two charges.



WHAT THE MODEL PREDICTED



WHAT HAPPENED NEXT



WHERE THE ERROR LANDED
each cell tinted by its squared error

PIXELS A DRIVER CARES ABOUT
the car, the road edge

10% OF THE ERROR
15% OF THE FRAME

PIXELS NO DECISION DEPENDS ON
leaves, puddle, light

90% OF THE ERROR
30% OF THE FRAME

Most of the error, and so most of the push on the weights, is on pixels no decision depends on.

SHADES, ERRORS AND SHARES ARE ILLUSTRATIVE

SECOND	SHARE OF THE ERROR ON THE CAR AND ROAD EDGE	SHARE ON LEAVES AND WATER
1	10%	90%

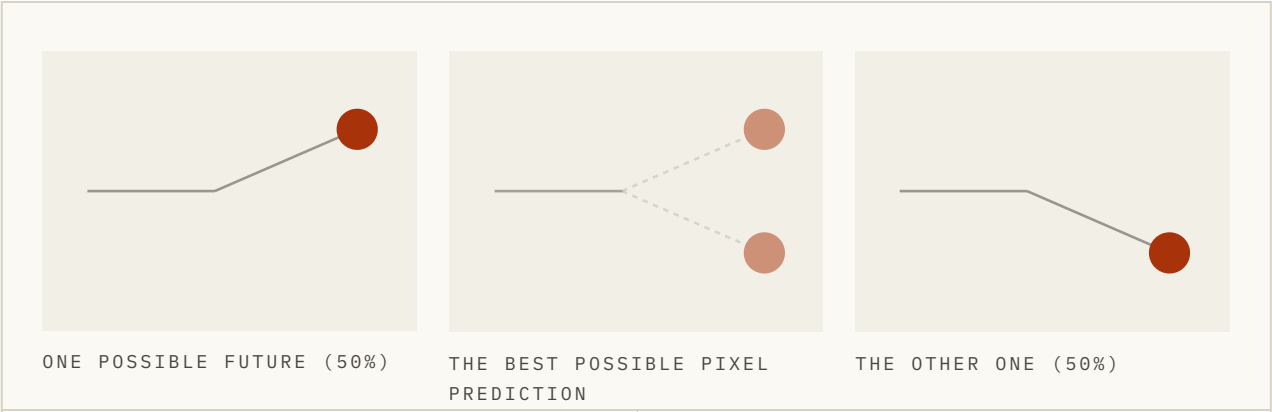
FIG. 7.1 A road scene, scored the way a pixel loss scores it. Press Next second and watch where the error lands: the car ahead moves the way the model expected, and the leaves and the puddle flicker the way nobody could expect. Now drag the foliage slider up and read the ledger on the right. The share of the error on things a driver would care about keeps shrinking, and every one of those pixels pulls on the weights just as hard. The numbers are illustrative.

JEPA is a family of models from Meta. It predicts a description of what it cannot see, instead of the pixels. An embedding, in everything that follows, is a short list of numbers standing in for a picture.

An average of two futures is neither

The larger charge is a car coming to a junction. It will turn left or right. You do not know which, and no amount of staring at the current frame will tell you.

Now let's ask a deterministic model, one that gives a single answer every time, for one next picture. Score it by squared pixel error, the gap at each pixel squared and added up. The best answer in the maths is the conditional average of the two turns. That is the left and right futures blended at their odds, rather than either one on its own.



IS THE MIDDLE PANEL A THING THAT CAN HAPPEN?

No. It is both futures at once, and neither of them is that.

DOES IT SCORE BETTER THAN EITHER REAL FUTURE?

Yes. Squared error over a set of outcomes is minimised by their average, so the optimum is a picture of nothing.

FIG. 7.2 The middle panel is the two futures drawn at their odds, and that average is what squared error rewards. Drag the chance to either end and you should see the best answer settle on one turn. Leave it near the middle and the best prediction becomes a picture of something that cannot happen.

Have a look at the middle panel. It scores better than either real future. A model that predicted the left turn would be punished half the time. The ghost in the middle is punished a little all the time, and under squared error a little all the time wins.

So that training signal asks for the mean rather than for a plausible future. When the future is open, the mean is a smear that never occurs.

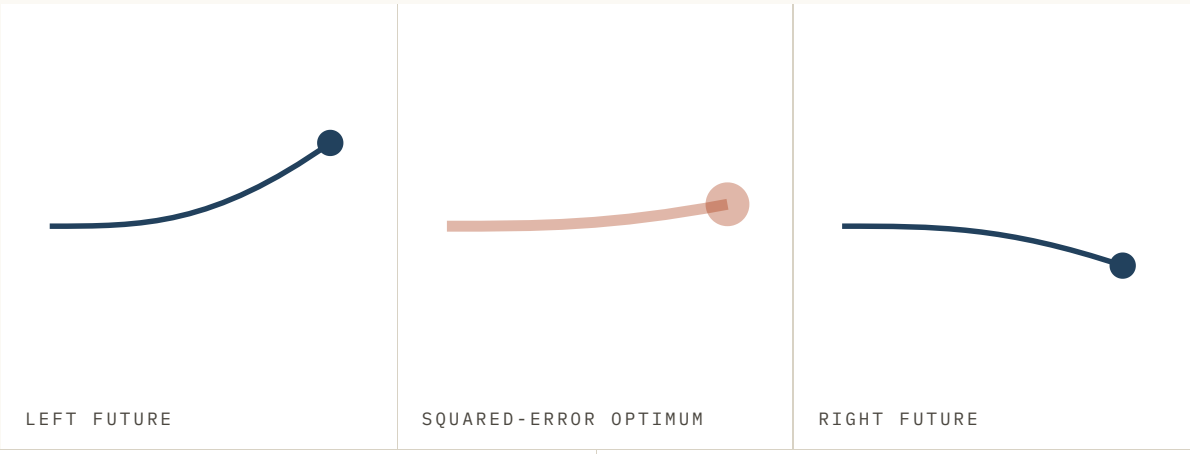
> A deterministic predictor trained with a mean-seeking pixel loss goes vague just when something interesting is about to happen. Modern generators avoid that failure by drawing samples rather than aiming at a mean.

The ghost had an answer five years early

One complication, though. The pixel camp had dealt with the ghost by 2017, five years before the position paper. SV2P, short for *Stochastic Variational Video Prediction*, is Mohammad Babaeizadeh and colleagues' video model from that year,

and Emily Denton and Rob Fergus built one like it. Both learn a spread of futures and draw one at random.

The trick is a hidden variable, a few numbers standing in for whatever the frame cannot tell you, such as which way the car turns. Draw the variable first, then the frame that goes with it. Each draw can be sharp even when the model has no idea which turn will happen.



INDIVIDUAL PREDICTION
Blurred: it lands between outcomes.

UNCERTAINTY
Collapsed into one average image.

FIG. 7.3 Switch from the mean to samples of the same two futures, and the in-between frame disappears. Press Draw a few times. The uncertainty survives as variation across draws. What the model still does not say is which draw the world will choose, so a planner has to value risk across the whole spread.

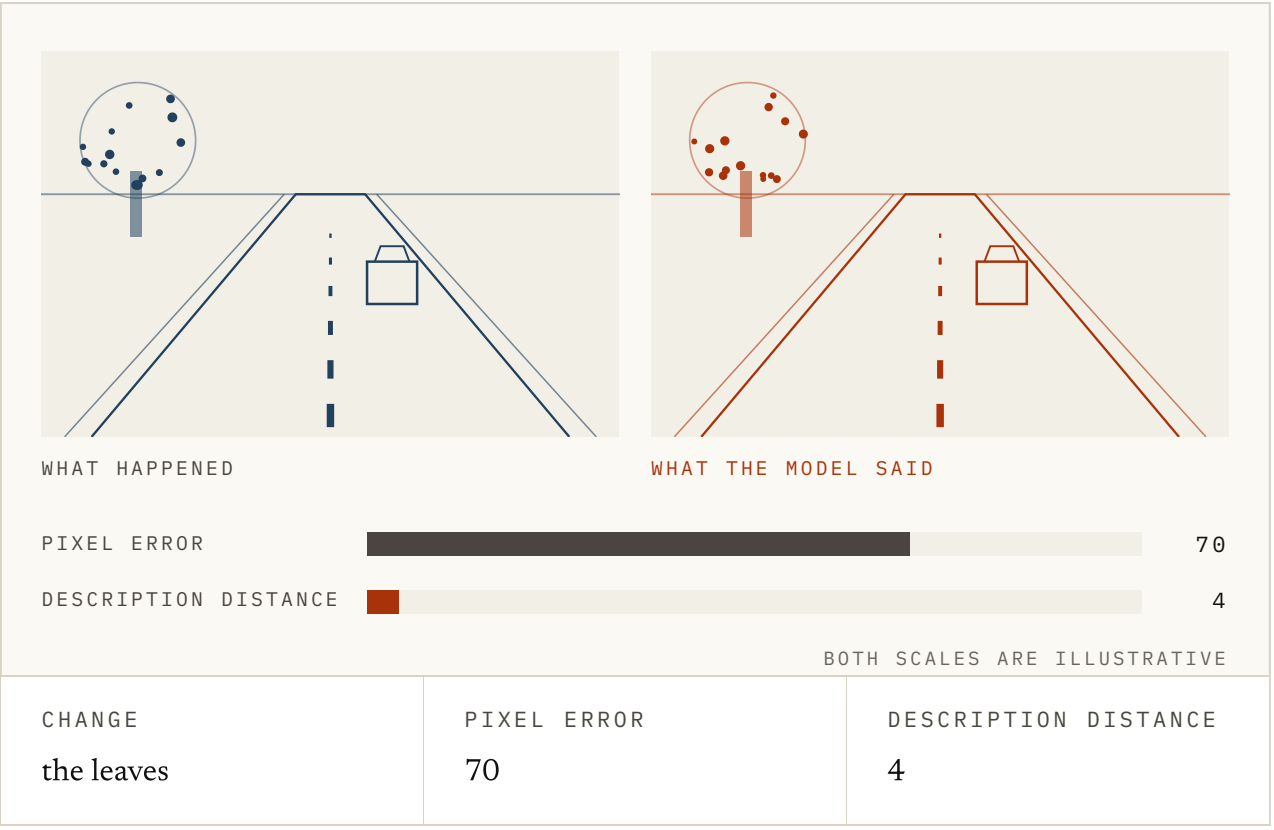
That kills the blur objection, though not the rest of the case.

Sampling fixes the form of the output. It does not make useless texture cheap, make rare futures well estimated, or tell a decision maker which possible future matters. Stochastic generation keeps the pixel target, and embedding prediction changes it.

Predict the description instead

LeCun's proposal is to stop predicting the picture. Encode what you can see into a compact description (an *embedding*, a short list of numbers that stands in for something bigger). Similar things land near each other in that list. Encode the part you are predicting into one too.

Then train the model to predict the second description from the first. Nothing is ever decoded back into pixels. I-JEPA is the simplest example: a 2023 image model from Mahmoud Assran and colleagues in LeCun's group at Meta. Hide part of the picture and predict the embedding of the missing piece. What changes with the target is what counts as a bad prediction.



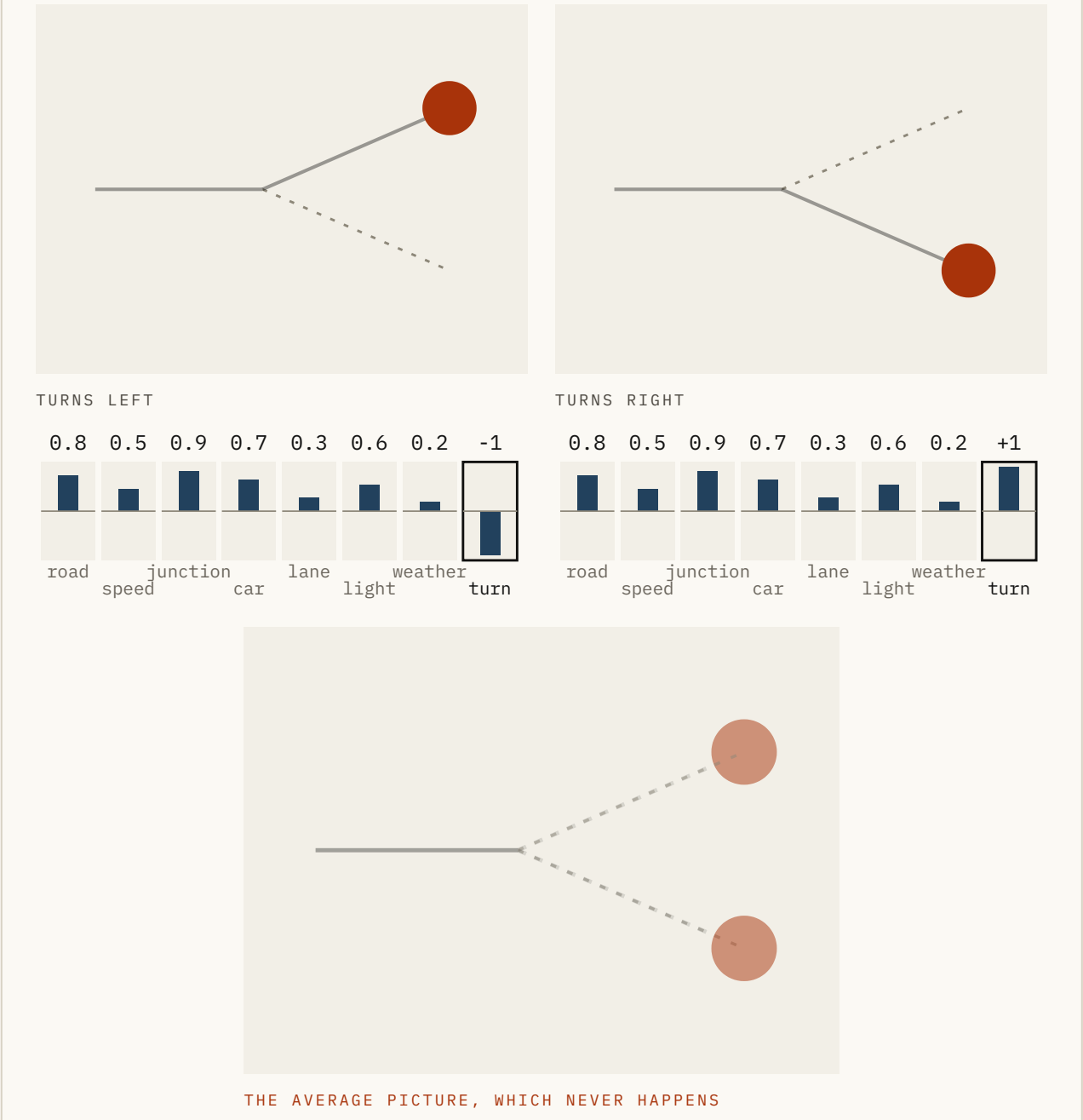
CHANGE	PIXEL ERROR	DESCRIPTION DISTANCE
the leaves	70	4

FIG. 7.4 Two road frames, and one change between them. Pick a change and drag it up, then read the two scales. Move every leaf, or shift the whole frame sideways, and the pixel scale charges a fortune for something no decision turns on. Move the car ahead into your lane and a few hundred pixels change what you would do, and the pixel scale barely notices. Both scales are illustrative.

V-JEPA 2, from Meta AI in 2025, does the same for video. It is trained first on plain video, with no actions. Then it is trained again on a robot's actions, for control.

Two things follow. The leaf problem goes away, because the description never had to record which way the leaf went. If a detail does not survive the encoding, nothing downstream is graded on it.

The ghost shrinks as well, because you are no longer averaging pictures. The descriptions of a car turning left and one turning right can share most of their entries. The part still open is either kept explicit or dropped as irrelevant. One caution: a space of descriptions can still hold two separate answers, and changing the representation does not get rid of uncertainty.



ENTRIES THE TWO FUTURES SHARE	ENTRIES STILL OPEN	DROPPED AS IRRELEVANT
7 of 8	1 of 8, kept explicit	none

FIG. 7.5 The same junction as Figure 7.2, this time described rather than drawn. Each future is a short list of numbers, and most of the entries agree: road, speed, a car at a junction. Leave the switch on pixels and the prediction is the ghost again. Flip it to description and watch what happens: the shared entries are predicted outright, and the one entry that differs is the only place the two futures still disagree. The lists are illustrative.

> This is the JEPA family, and the name describes the architecture. It is joint embedding because both sides get embedded, and predictive because the training signal is one embedding predicting another.

The bill for that

Predicting pixels has one large virtue, and the JEPA side gives it up. The target is fixed, and the model cannot make the next frame easier to predict. Predict an embedding and that stops being true.

The target now comes from the encoder, the network that turns pictures into descriptions, and that network is also being trained. It can get a perfect score while learning nothing, by making every description the same. The prediction is then always right, the loss goes to zero, and the descriptions say nothing about anything. That failure is called collapse.

HOW CLOSE THE PREDICTION IS	HOW MUCH THE REPRESENTATION STILL SAYS	WHAT YOU HAVE GOT
0.00000	0.000	A perfect score and an empty representation. Everything encodes to the same point.

FIG. 7.6 A tiny encoder, two numbers in and two out, trained on four pairs by gradient descent. Leave the safeguard off and drag the step forward: you should see the loss and the spread of the representation reach zero together. Now turn the safeguard on and drag again. The loss never gets close to zero, and that is the run you want.

Most of the difficulty in these methods is preventing that, and the fixes form a lineage. In 2020 the received wisdom for learning descriptions without labels was contrastive, meaning you learn by comparing pairs. Pull two views of the same image together. Push everything else apart, so the descriptions cannot all pile up in one place.

BYOL, short for *Bootstrap Your Own Latent*, is a 2020 method from Jean-Bastien Grill and colleagues at DeepMind. It dropped the pushing apart. Instead it keeps a second copy of the encoder that updates slowly (usually called a *target encoder*, it trails the one being trained), and it predicts that copy's output. The target cannot run away from the predictor.

By the standard account it should have collapsed. It did not, and that convinced people collapse could be avoided without pushing anything apart.

VICReg, a 2021 method from Adrien Bardes, Jean Ponce and LeCun, went the explicit route instead. The name spells out its three terms: variance, invariance, covariance. It penalises descriptions whose parts have collapsed or copied each other. That is Figure 7.6's safeguard done properly.

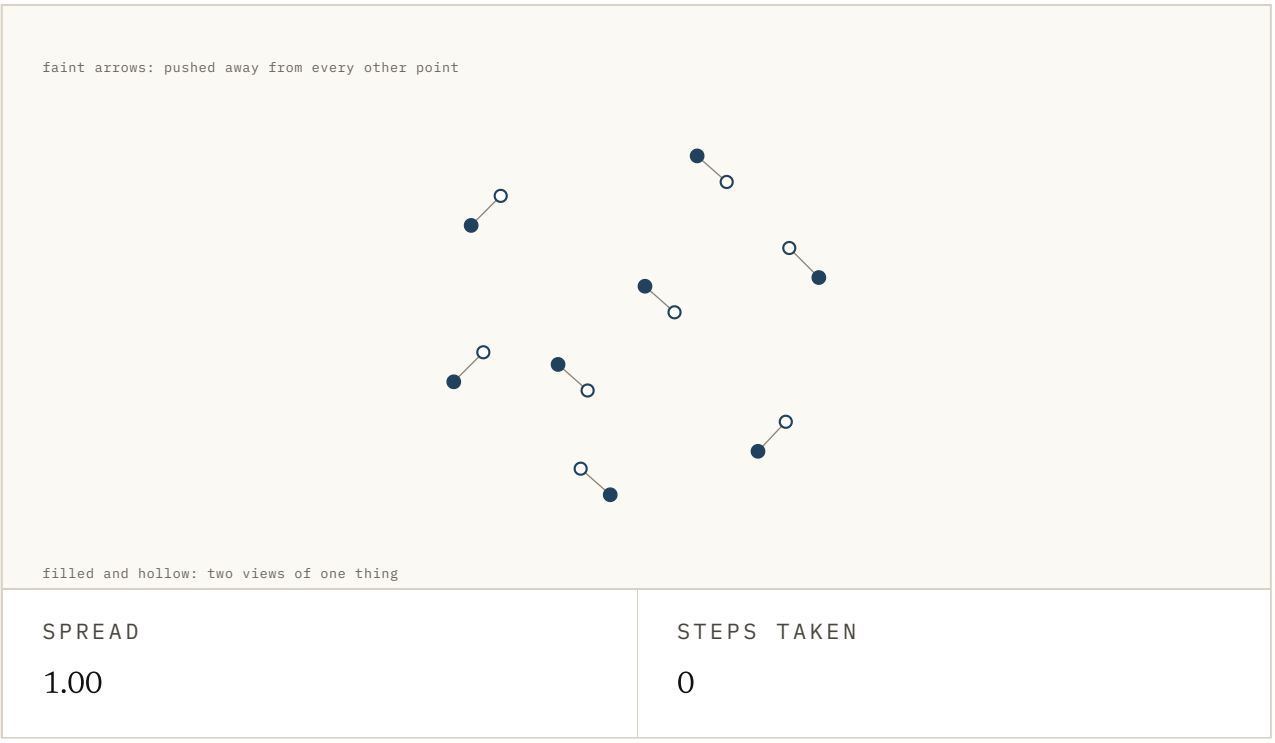
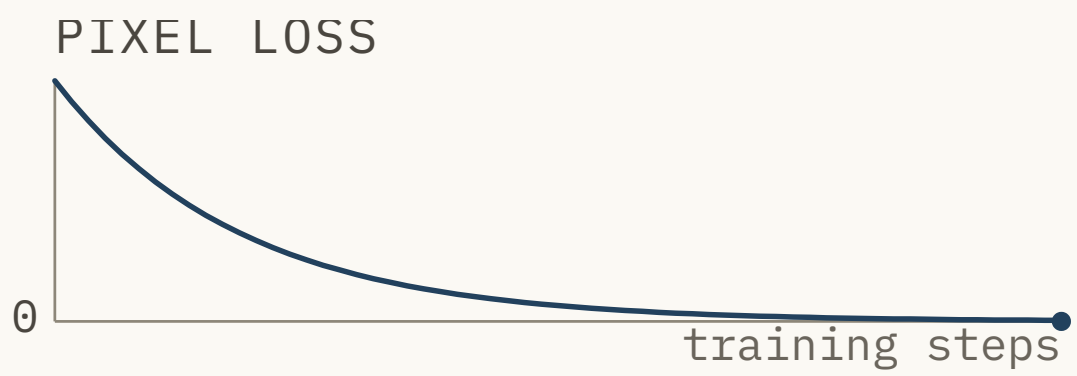
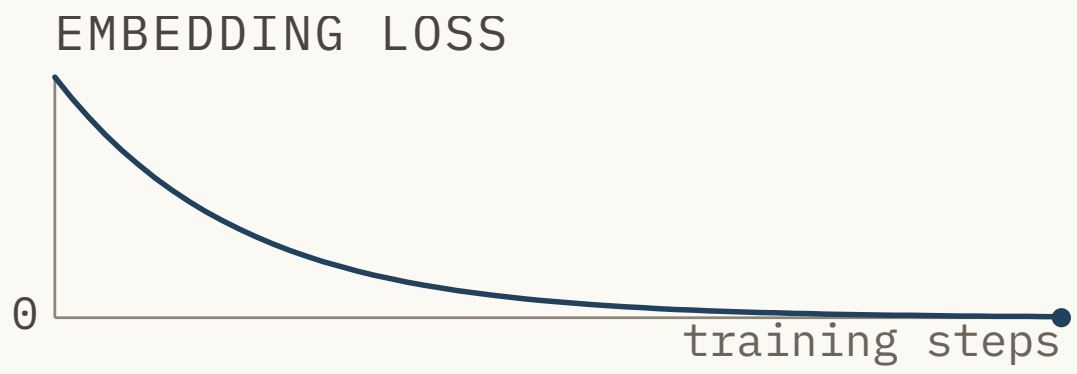


FIG. 7.7 Three ways of stopping the pile-up, each run on the same little cloud of descriptions. Pick one and press Step a few times. Contrastive pushes every pair apart. BYOL pushes nothing apart, and the cloud still holds its shape because the target it chases trails behind. VICReg leaves the points alone and penalises the cloud itself when its spread drops or its two axes copy each other. The cloud is illustrative; watch the spread readout.

Randall Balestriero and colleagues wrote a 2023 survey, *A Cookbook of Self-Supervised Learning*. It is frank that much of the field is machinery for stopping the trivial solution from winning.

The loss is also no longer a number you can read. A pixel loss of zero means the model predicted the frame. An embedding loss of zero might mean it understands everything, or nothing at all. The number will not tell you which.





CURVES ARE ILLUSTRATIVE

PIXEL LOSS AT THE END

WHAT THAT ZERO TELLS YOU

0	not yet revealed
EMBEDDING LOSS AT THE END 0	WHAT THAT ZERO TELLS YOU not yet revealed

FIG. 7.8 Two training runs, both ending with a loss of zero. Press Reveal on the pixel run and there is nothing to reveal: zero means the frame was predicted. Press Reveal on the embedding run and the same zero splits two ways, one where the descriptions still tell left from right and one where they have all become the same. Only the probe underneath can tell you which you got. The curves are illustrative.

So Quentin Garrido and colleagues tested what the embeddings could do. Their 2024 paper is *Learning and Leveraging World Models in Visual Representation Learning*. They found that the strength of the predictor decides what the encoder keeps. Give the predictor an easy job and the encoder throws away whatever the transformation changed. Make it work, and the encoder keeps enough to undo the change, which is more than the loss ever asked for.

What this does not settle

None of this makes generation a mistake. The version of the argument that says it does is the overclaim. Systems that predict pixels are the ones making worlds you can walk around in, and they work. Whatever they waste on leaves, they deliver a picture, and no embedding does that.

The awkward counter-example comes from LeCun's own lab. Kaiming He, best known for ResNet, the network design computer vision ran on for years, was at that lab in 2021. His paper that year was *Masked Autoencoders Are Scalable Vision Learners*. It hides most of an image and rebuilds its pixels, the very thing JEPA refuses to do.

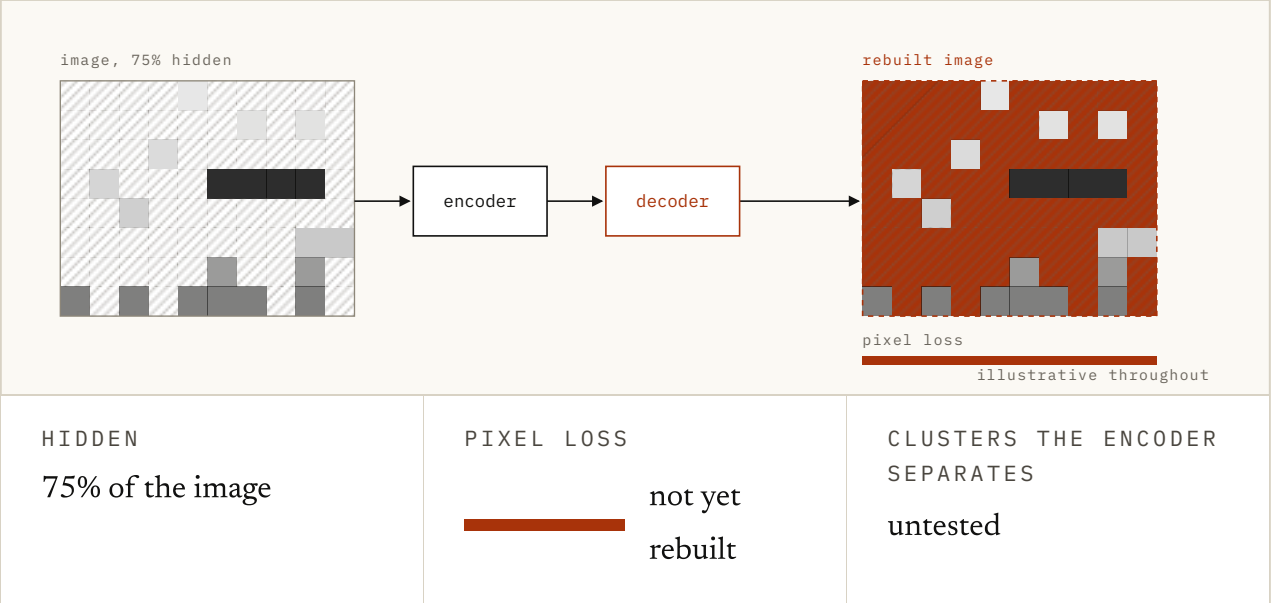


FIG. 7.9 Drag the mask up to three quarters of the image and press Rebuild. The decoder redraws the hidden patches, roughly, which is exactly the job the previous section said was wasteful. Then press Test the encoder. The descriptions it learned still sort the shapes cleanly, and that is the result to keep in mind next time you hear the slogan. Illustrative throughout.

The descriptions it learns are very good anyway. It came out the year before the position paper.

The version of the argument that holds up is narrower than the slogan, but it is right. Predicting appearances spends most of its capacity on things no decision depends on. When the future is open, a deterministic predictor is trained towards a picture of something that will not happen.

If what you want is a compact state to act on, that is a bad way to get one. If what you want is a picture, it is the only way anyone has. Most of the confusion in this corner of the field comes from comparing systems built for different goals.

Hopefully this chapter helped. There is a short quiz below if you want to check the argument stuck, including the part about what it does not settle. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section. I would be glad to hear from you.

Try it

1 A deterministic predictor uses squared pixel error and a car turns left or right with equal probability. What is its optimum?

a. The average of the two, which is a picture of neither
b. Whichever was more common in training
c. The left turn
d. The right turn

> ANSWER a. The average of the two, which is a picture of neither. Squared error over a set of outcomes is minimised by their mean. Committing to one turn is punished half the time; the smear is punished a little all the time, and that wins.

2 A stochastic video model now draws sharp left and right turns instead of their blur. What has it fixed, and what has it not?

a. It removed the need for a pixel target
b. It proved embedding prediction is unnecessary
c. It has learned which turn the real car will choose
d. It fixed the impossible average but still has to represent probabilities and decision risk

> ANSWER d. It fixed the impossible average but still has to represent probabilities and decision risk. Sampling is a real solution to blur: each draw can be plausible. It does not identify which draw reality will select or tell a planner how to value the distribution.

3 A leaf at the edge of the frame moves unpredictably. Why does a pixel-predicting model spend capacity on it?

a. Leaves are important for scene understanding
b. The leaf is made of pixels, and pixels are what the model is marked on
c. It cannot tell leaves from cars
d. The encoder forces it to

> ANSWER b. The leaf is made of pixels, and pixels are what the model is marked on. Nothing tells the objective which pixels matter. Hard-to-predict and irrelevant is the worst combination, and it is most of the frame.

4 What does predicting an embedding rather than a picture do to the leaf problem?

a. Nothing, the leaf is still in the input
b. It makes the leaf easier to predict
c. If a detail does not survive the encoding, nothing downstream is graded on it
d. It moves the problem to the decoder

ANSWER c. If a detail does not survive the encoding, nothing downstream is graded on it. The description was never obliged to record which way the leaf went, so the model is not punished for failing to say.

5 Predicting pixels has one large virtue that predicting embeddings gives up. What is it?

a. The target is fixed: the model cannot make the next frame easier
b. It needs less data
c. It generalises better
d. It is faster to compute

ANSWER a. The target is fixed: the model cannot make the next frame easier. The moment the target is produced by a network that is also being trained, the target can move, and there is a way to make it move somewhere very convenient.

6 What is collapse?

a. The model forgetting earlier training
b. Gradients vanishing in deep layers
c. The loss diverging to infinity
d. Every input encoding to the same description, so the prediction is always right and the representation says nothing

ANSWER d. Every input encoding to the same description, so the prediction is always right and the representation says nothing. It is a perfect score obtained by learning nothing, and it is the cheapest solution available unless something is specifically stopping it.

7 In Figure 7.6, the run with the safeguard has a worse loss. What does that tell you?

a. The safeguard is badly tuned
b. An embedding loss is no longer a number you can read off as quality
c. The model needs more training
d. The safeguard should be removed once training stabilises

ANSWER b. An embedding loss is no longer a number you can read off as quality. A pixel loss of zero means the frame was predicted. An embedding loss of zero might mean everything or nothing, and the number cannot tell you which.

8 Does this argument show that generating pixels is a mistake?

a. No, because collapse makes embeddings unusable
b. Yes, it is strictly worse
c. No. It is a bad way to get a compact state to act on, and the only way anyone has to get an actual picture
d. Yes, except for very short videos

> ANSWER c. No. It is a bad way to get a compact state to act on, and the only way anyone has to get an actual picture. The two goals were never the same goal, and most of the confusion here is people comparing systems built for different ones.

FIG. 7.10 Eight questions on the argument and its bill. The last two are there to stop the argument being read as a knock-down case, which is the mistake I'd most like you to avoid.

Sources

Stochastic Variational Video Prediction (SV2P) BABAEIZADEH ET AL., 2017 ↗

The real counterargument to deterministic blur: learn a distribution and draw distinct plausible futures instead of returning their pixelwise mean.

https://arxiv.org/abs/1710.11252

A Path Towards Autonomous Machine Intelligence LECUN, 2022 ↗

The position paper the whole argument comes from, including why predicting appearances is the wrong job for a system meant to act.

https://openreview.net/forum?id=BZ5a1r-kVsf

I-JEPA ASSRAN ET AL., 2023 ↗

Hide part of an image and predict the embedding of the missing piece rather than redrawing it. The clean statement of the method.

https://arxiv.org/abs/2301.08243

V-JEPA 2 META AI, 2025 ↗

The video version, pre-trained without actions and then post-trained for control, which is where the argument meets a robot.

https://ai.meta.com/blog/v-jepa-2-world-model-benchmarks/

Bootstrap Your Own Latent (BYOL) GRILL ET AL., 2020 ↗

The slowly-updating target copy, and the result that made people believe you could avoid collapse without pushing things apart.

https://arxiv.org/abs/2006.07733

VICReg BARDES, PONCE & LECUN, 2021 ↗

The explicit approach: penalise a representation whose components have collapsed or duplicated each other. Figure 7.6's safeguard, done properly.

https://arxiv.org/abs/2105.04906

A Cookbook of Self-Supervised Learning BALESTRIERO ET AL., 2023 ↗

The survey that says how much of this field is machinery for stopping the trivial solution from winning.

https://arxiv.org/abs/2304.12210

Masked Autoencoders Are Scalable Vision Learners HE ET AL., 2021 ↗

The counter-example worth holding on to: reconstruct the pixels of the masked part, and it works very well anyway.

https://arxiv.org/abs/2111.06377

Learning and Leveraging World Models in Visual Representation Learning

GARRIDO ET AL., 2024 ↗

What the embedding-prediction objective turns out to have learned, tested rather than asserted.

https://arxiv.org/abs/2404.08471

FIG. 7.11 I've ordered these for someone building on this rather than for historical completeness, and preferred first-party material throughout.

08 Are Video Models World Simulators?

BY NILUSHANAN KULASINGHAM

13 MIN READ · INTERACTIVE

> Add an action input to a video model and it is steerable in principle. From Genie to Genie 3: how to tell conditioning from control, what scaling bought, and why fitting physics is not the same as having the rule.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/video-worlds.

If you have watched any generated video lately, you'll have seen somebody walk through a scene with the arrow keys. Until 2024 a video model was something you watched. It chose its own future and you had no say. One extra input changed that.

Take a model that predicts the next frame. Now tell it, along with the frames, which key is being held. From the same opening it can give you one continuation for left and another for right. Have a go with the figure below, and watch who picks each frame.

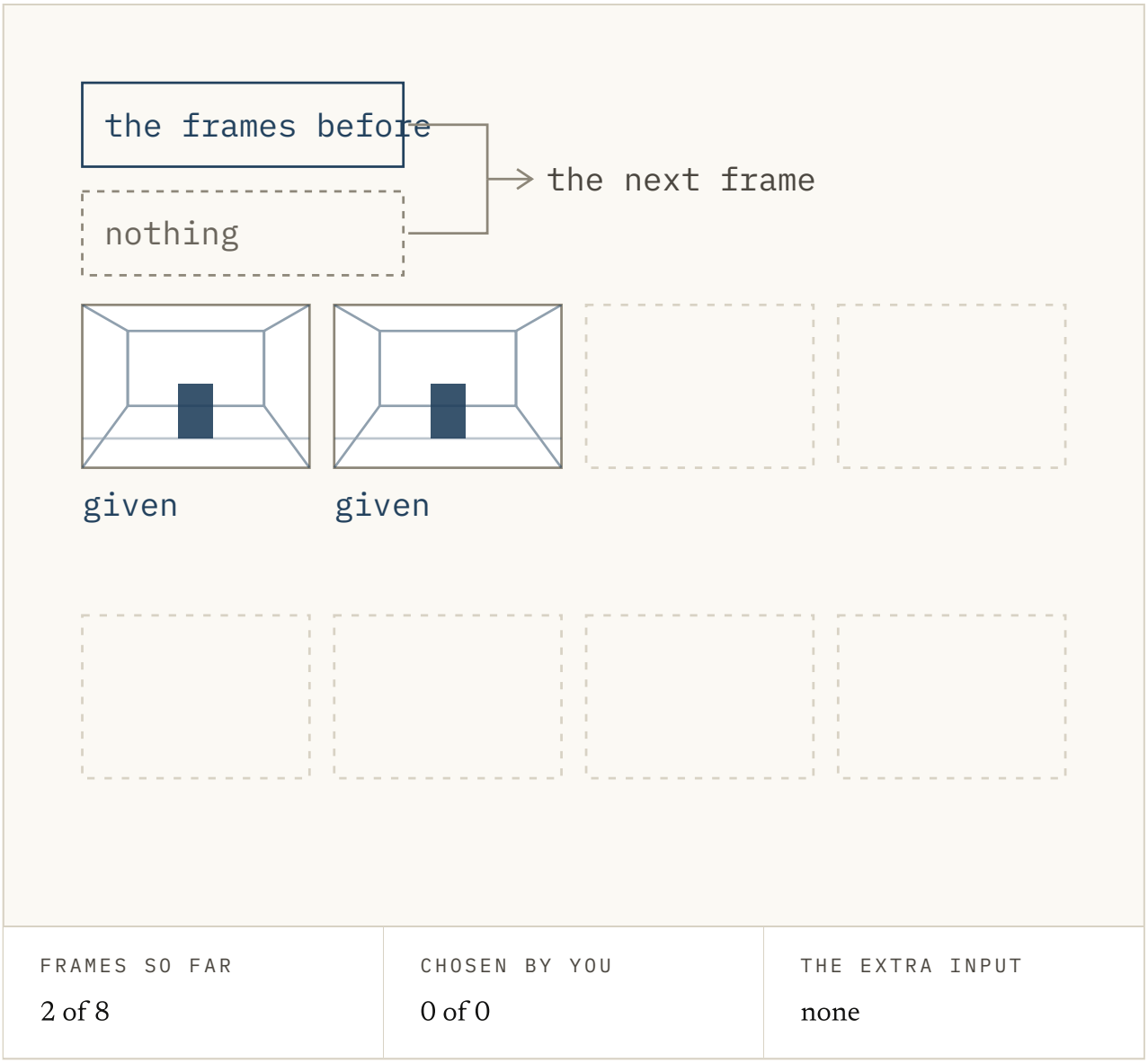


FIG. 8.1 Two ways of getting the next frame. Leave it on watch and press Next frame: the model picks, and starting again gives you the same clip back. Switch to steer and the arrow you press goes in alongside the frames, so the frame you get is the one you asked for. Watch the tally underneath for who chose each frame. The clip is illustrative.

Jake Bruce and colleagues at Google DeepMind made that move with Genie in 2024. Their paper was called *Genie: Generative Interactive Environments*. Genie was trained on footage of 2D platform games, Mario and its descendants, and it turned them into worlds you could steer. Nobody had labelled the keypresses.

Instead, the model had to explain each change between frames with one of a handful of codes. The codes it settled on behaved like a controller. Genie also married two families that had grown up apart: world models built for agents (chapters 5 and 6)

and video models built for audiences.

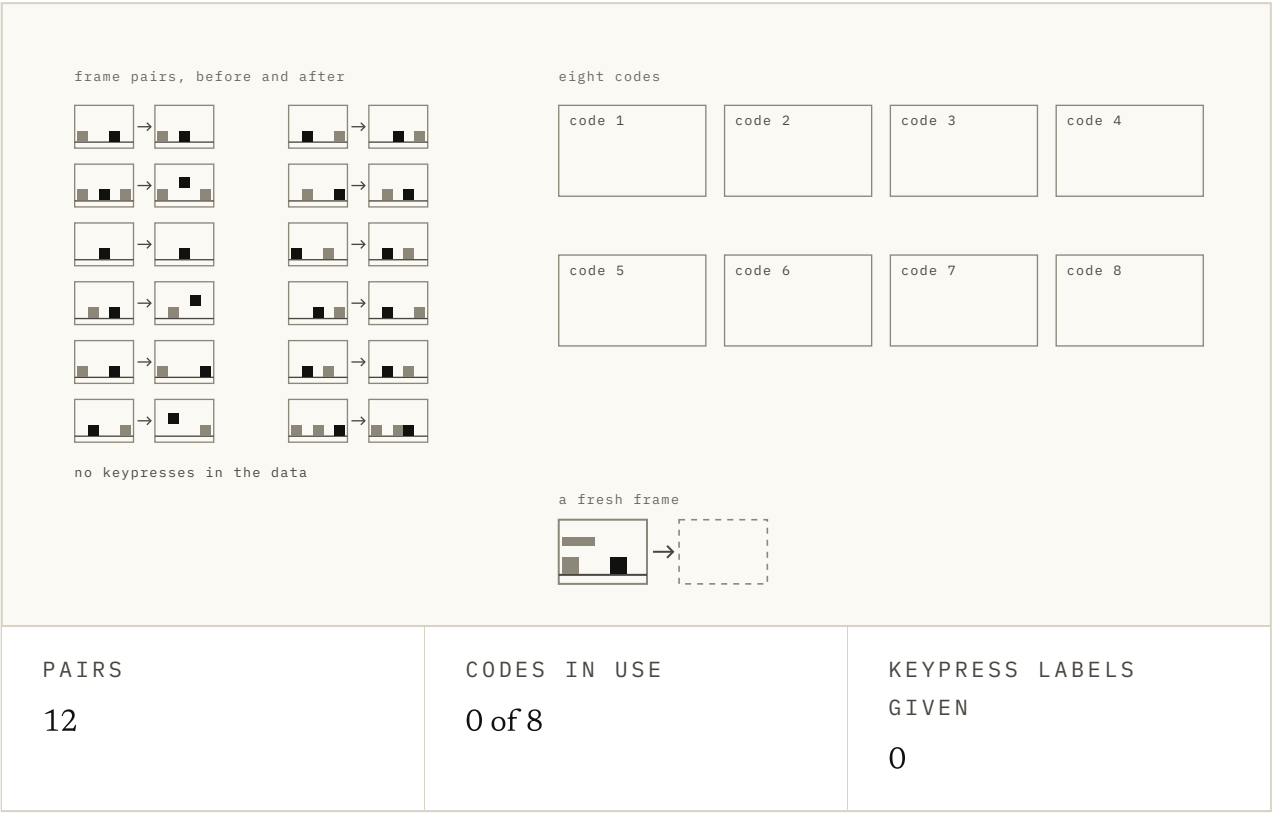


FIG. 8.2 This is that trick in miniature. The model sees pairs of frames and no keypresses, and has to file each change under one of eight codes. Press Sort the changes and watch the piles: one code collects every step left, another every jump, and nobody told it which was which. Then press a code and see what it does to a fresh frame. The frames and the codes are illustrative.

A video model with an action input is somewhere you can be, and that is new. Whether it is a simulator, something that obeys your actions and holds the rules of the world, has to be measured, and the announcements tend to skip that.

An action input is not control

A model can receive an action and still ignore it, or respond only weakly. It can also let the action change how things look without changing the path underneath. Think of a steering wheel that turns without being connected to anything.

So control has to be measured. Hold the start fixed, vary only the command, and check whether the futures split apart as asked. I call it the steering-wheel test, and the figure below is that test as a picture.

ACTION INPUT	LEFT-RIGHT SEPARATION	CONTROL TEST
present in every row	107 px	The command changes the video, but weakly.

FIG. 8.3 Every row gets a different command from the same start. Conditioning is the name for giving the model the action as an input, and action fidelity is how far the model obeys it. Drag fidelity towards zero and you should see the paths fold together while the input stays present.

As you can see, the input can be there the whole time and mean very little. That failure is the one I'd expect. A model trained on footage is rewarded for frames that look right. If the footage mostly goes straight, going straight is a good bet whatever the key says.

Good video hides weak control. If three beautiful continuations all go straight, you have been shown video. A planner that asked for three counterfactuals, three what-ifs from one start, has been handed one of them three times.

From watching to walking in

Scaling did more than most people expected, including most of the people building it. Let's take the two systems that mark the change.

GameNGen came from Dani Valevski and colleagues at Google Research in 2024. Their paper was called *Diffusion Models Are Real-Time Game Engines*. A diffusion model makes a picture by starting from noise and cleaning it up step by step. GameNGen used one to run DOOM, the first-person shooter from the early nineties.

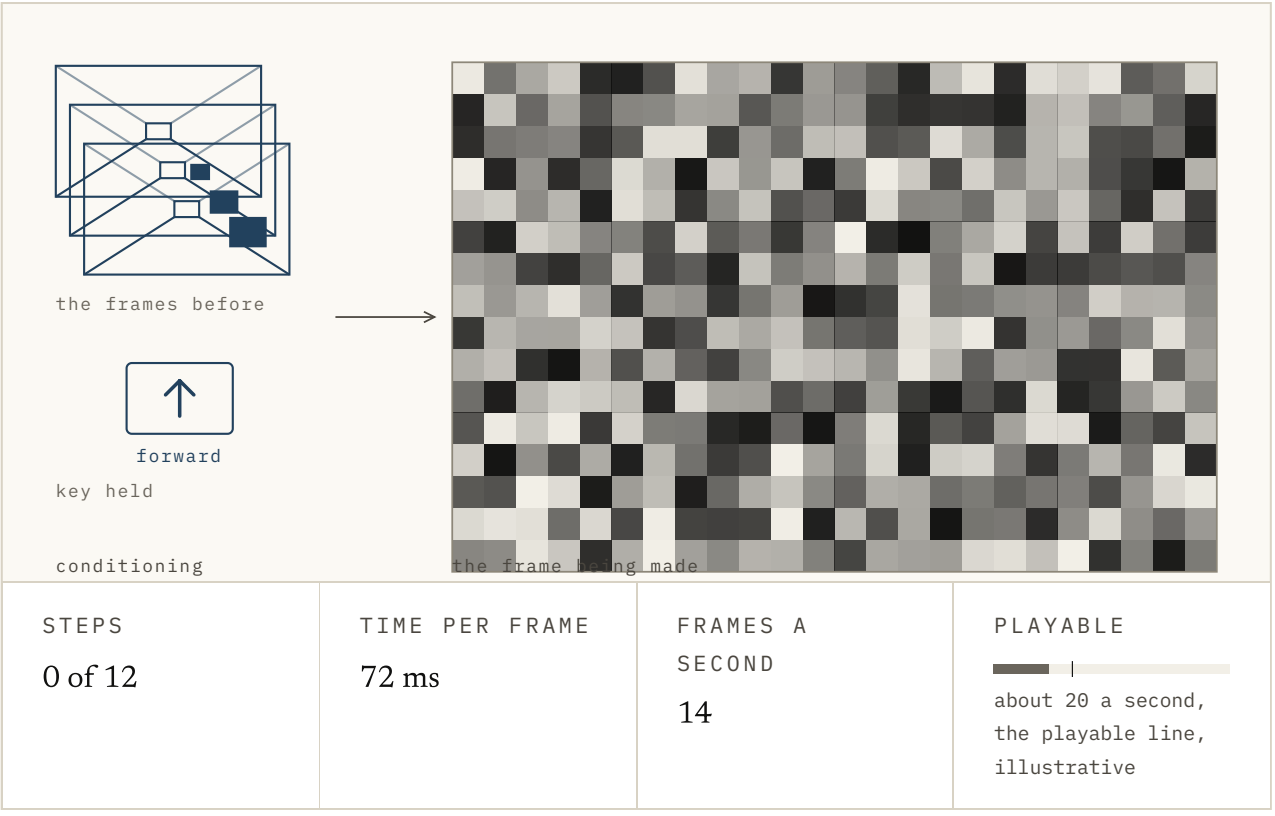


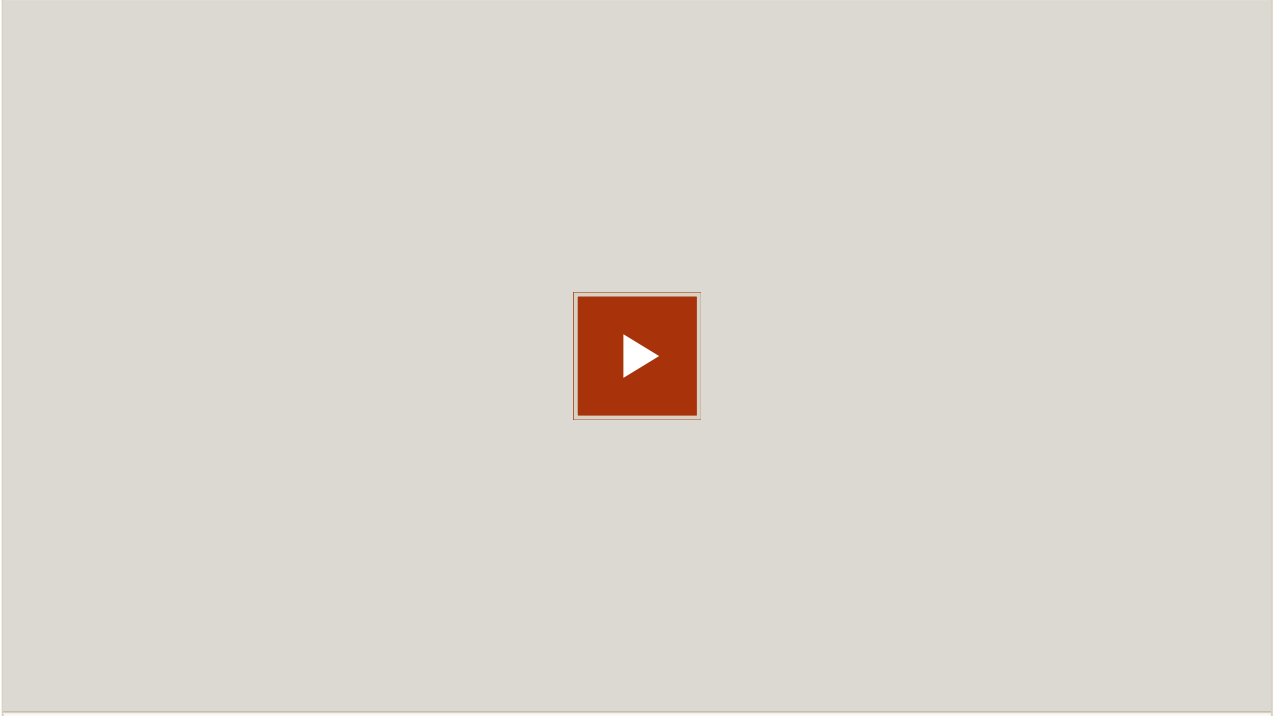
FIG. 8.4 This is how a diffusion model makes one frame of a game. The frames before it and the key being held sit on the left as the conditioning. Press Clean up and watch the noise turn into a frame one step at a time. Then drag the number of steps down and read the frames a second: fewer steps is faster and rougher, and somewhere along that slider watching turns into playing. The picture and the timings are illustrative.

It made each frame from the earlier frames and the player's inputs, fast enough to play. People shown short clips struggled to tell it from the real game, its authors reported.

Genie 3, from Google DeepMind a year later, is where the numbers stopped looking like a demo. It is reported to make interactive video at 720p and 24 frames a second.

> Those numbers are DeepMind's report of its own system. Nobody outside the lab has repeated them. I'd say that is normal here rather than a slight.

It is said to hold a scene together for several minutes and to take instructions part-way through: change the weather, add a flock of birds. Walk away from something, walk back, and it is roughly as you left it. Have a look at the clip below and watch for exactly that.



GOOGLE DEEPMIND Genie 3: Creating dynamic worlds that you can navigate in real-time

FIG. 8.5 DeepMind's own clip of Genie 3. Watch for the moments the camera turns away and comes back, and ask yourself what you were shown. The scene is roughly as it was, and nothing in the interface lets you open up whatever held it there. Everything in the clip is the lab's report, including the numbers I quoted above.

The public interface offers no stored 3D layout of the scene, and no list of its objects, that you could inspect. Yet DeepMind says Genie 3 can look back over its own path and keep visual information for about a minute. Whatever state supports that stays hidden inside the generator.

In under two years the field went from steering a blurry platformer to walking around a photoreal scene that remembered where you had been. Scaling did not give you anything you could open up and read.

Emergent physics, and where the footage runs out

You will see the claim made about all of them. OpenAI's 2024 report, *Video generation models as world simulators*, put emergent physics into common use. It was the write-up for Sora, OpenAI's own video generator. The title is the part that spread, and the claim in it bundles an observation with a conclusion.

The observation is real. Train on enough video and the outputs start to respect regularities nobody put in. Dropped objects speed up as they fall, water pools rather than piling up, and hidden objects come back into view when the thing in front moves. Nobody supplied an equation; those regularities are in the training footage, and predicting it well means copying them.

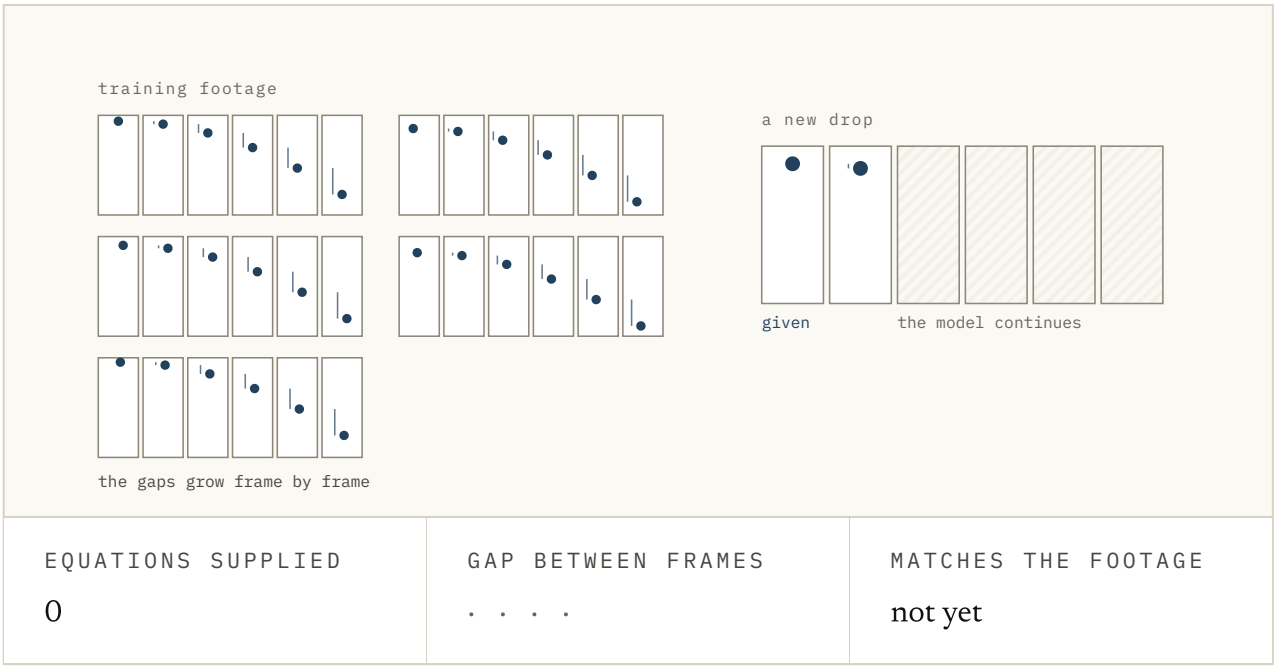


FIG. 8.6 On the left are the training clips, each one a thing being dropped, and each one speeds up on the way down. On the right, press Continue and the model finishes a drop it has never seen. It speeds up too, and nowhere in the figure is there an equation. Now switch the footage to the made-up kind, where things fall at a steady speed, and press Continue again. The regularity follows the footage. Illustrative.

The conclusion is where the trouble starts. "It learned physics" says the model picked up the rule, and that is testable. The test has never been whether the outputs look right on the footage it was trained on.

Bingyi Kang and colleagues ran one in 2024, in a paper called *How Far is Video Generation from World Model: A Physical Law Perspective*. They trained video models on simple simulated scenes of balls moving and colliding, then asked for speeds the footage never showed. Figure 8.7 runs that test twice: once on speeds the footage covered, and once on speeds past them.

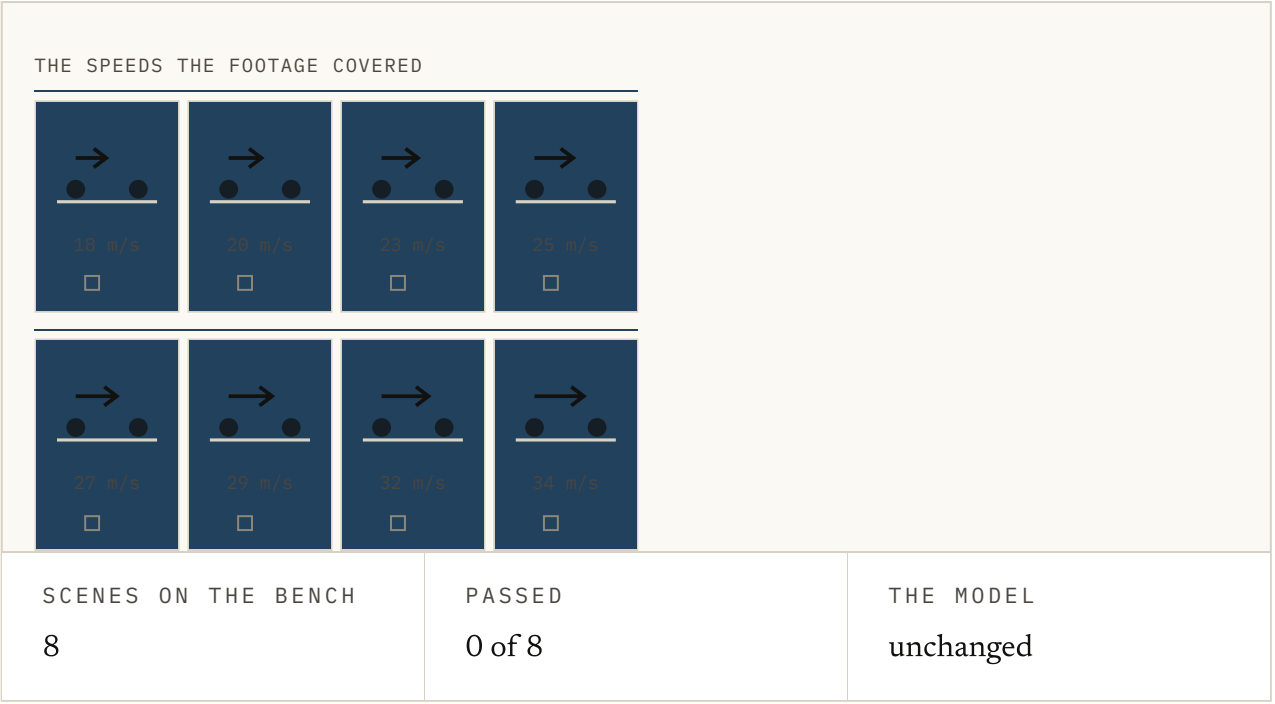


FIG. 8.7 Here is that test as a scorecard. Each tile is one scene, ordered by speed, and the shaded ones are the speeds the footage covered. Press Score it, then switch the bench to scenes from past where the footage ends and score it again. Same model, same procedure, and the only thing that changed is where the scenes came from. The speeds and the errors are illustrative.

Everything inside the band fits with having learned physics. It fits just as well with having learned the answers to the questions it was asked, and Kang's group concluded their models were doing the second: matching each new scene to the training scenes it most resembled and copying what happened there. Nobody tests outside the band, because that is where the footage runs out. I call that the band problem.

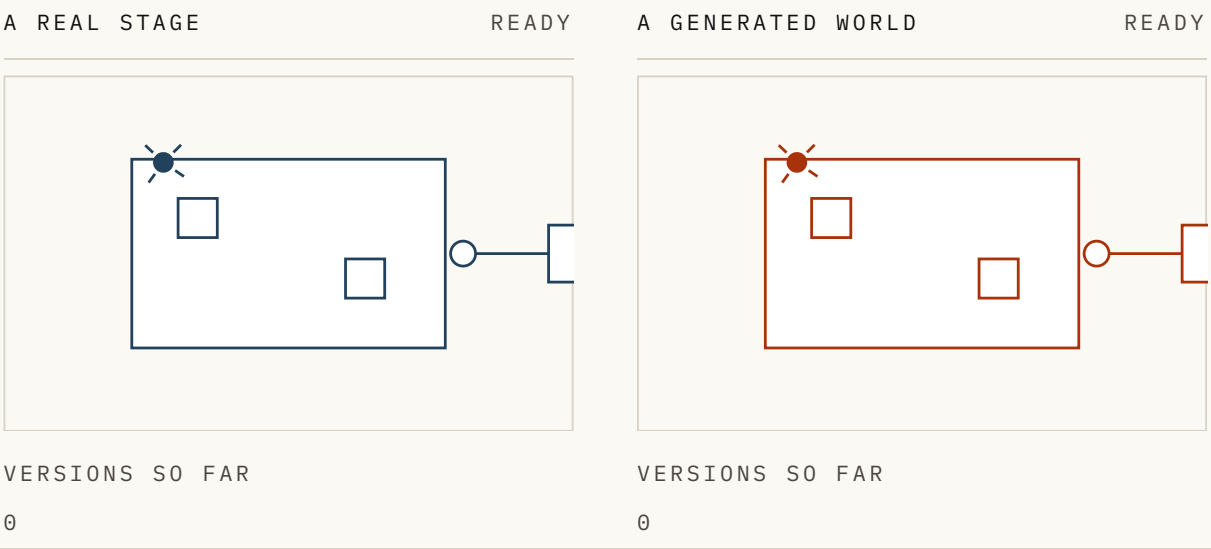
So the defensible claim is narrower. These systems copy a great many physical regularities across the situations they were trained on. Whether that is the same as having the rule, no demo you have seen can settle, because every one of them was inside the band.

> The band problem is not special to video. Any fitted function has it. It feels different here only because a wrong answer that looks like a photograph convinces people in a way a wrong number never could.

What they are for

None of that argues against using them. Answering "it has not learned physics" with "so it is a toy" is the mirror-image error, and I see it nearly as often.

For training other systems a generated world is useful. It avoids contact with the real environment and costs GPU time instead. It resets at once, and it can produce a thousand versions of a scene that would take a week to stage. Press the buttons in the figure below a few times and you'll see what I mean.



STAGE · VERSIONS	STAGE · TIME SPENT	STAGE · REAL CONTACTS	STAGE · STATUS
0	0 days	0	ready
GENERATED · VERSIONS	**GENERATED · TIME SPENT**	**GENERATED · REAL CONTACTS**	**GENERATED · STATUS**
0	0 frames of GPU time	0	ready

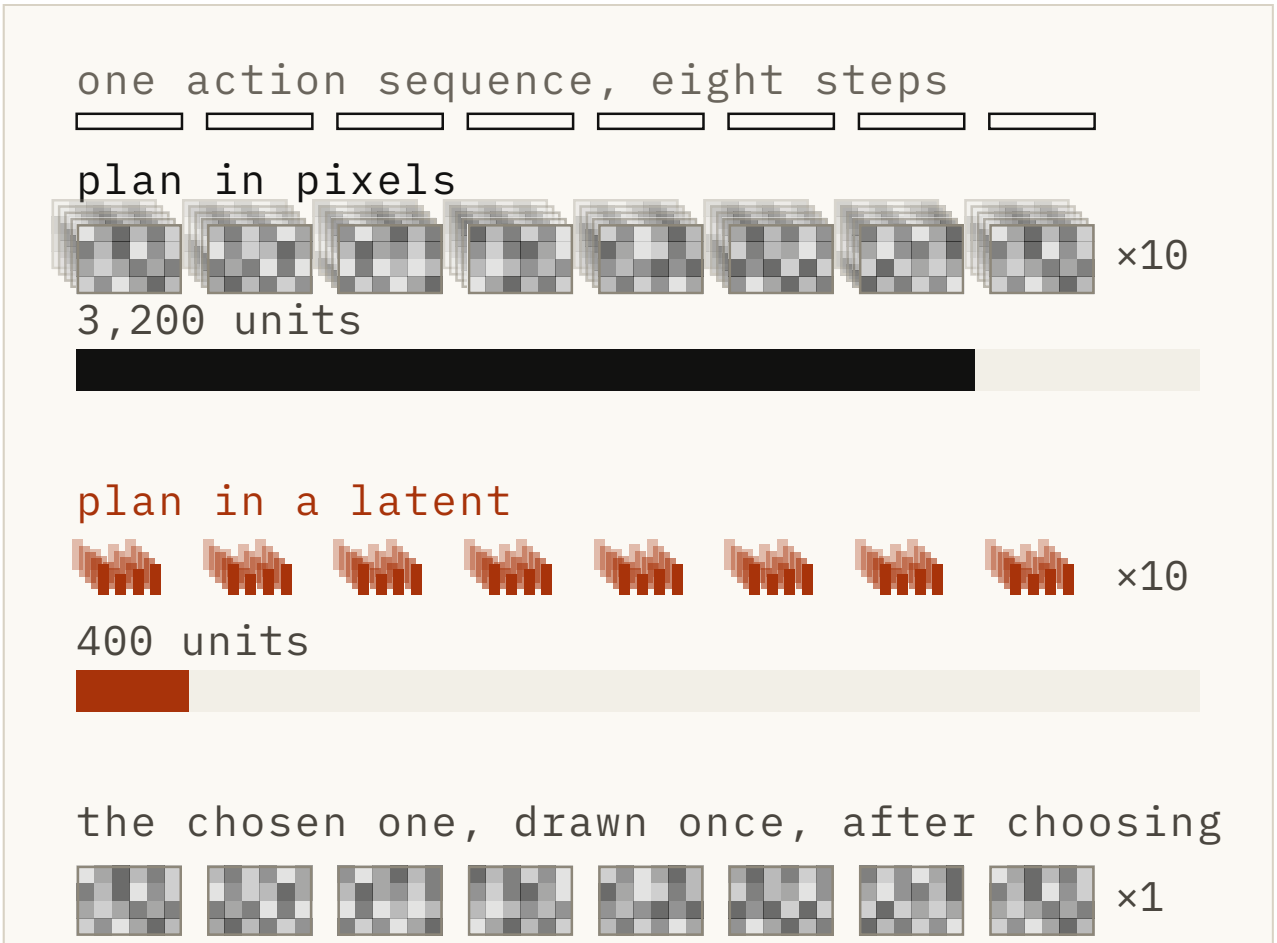
FIG. 8.8 One scene, two ways of getting more of it. On the left is a real stage: press Another and a day goes by and one new version arrives. On the right is a generated world: press Another and a version appears in the time it takes to draw a frame, with the lighting, the clutter and the camera moved. Press Reset on both and watch which one is ready. The counts and timings are illustrative.

NVIDIA, whose chips train most of these models, built Cosmos for this. Cosmos is a set of video generators. The 2025 paper is called *Cosmos World Foundation Model Platform for Physical AI*. It makes physically-aware video as training data for robots and vehicles, where real data is dearest.

NVIDIA sells Cosmos beside its explicit, hand-built simulators as one platform, and has not declared either the replacement for the other. I'd take that as a hint about where things stand.

For being a world you can move around in, they are already good. No other approach lets you walk into the picture. I think that is worth more than the sceptics admit and less than the launches imply.

For being the model an agent plans inside, raw video generators are an awkward shape. A planner may need to score a hundred action sequences, and rendering each one is expensive. Systems can instead distil a learned latent, a short internal code that stands in for the frames, and plan in that.



SEQUENCES	FRAMES RENDERED, PIXELS	CODES SCORED, LATENT
10	10 × 8 = 80	10 × 8 = 80
FRAMES DRAWN, LATENT	**COST, PIXELS**	**COST, LATENT**
8	3,200 units	400 units

FIG. 8.9 A planner has to score many action sequences before it picks one. Drag the number of sequences up and watch the two bills. Planning in pixels renders every frame of every sequence, and its bar runs off the chart. Planning in a latent, a short code standing in for each frame, scores the same sequences for a fraction of that, and only the chosen one is ever drawn. The costs are illustrative.

Where the seams are

There are three seams where a demo can mislead you.

Consistency is learned, not enforced. Nothing stops the room from changing, and usually it does not. But usually is a statistical property of a trained system, and a guarantee is a property of an engine.

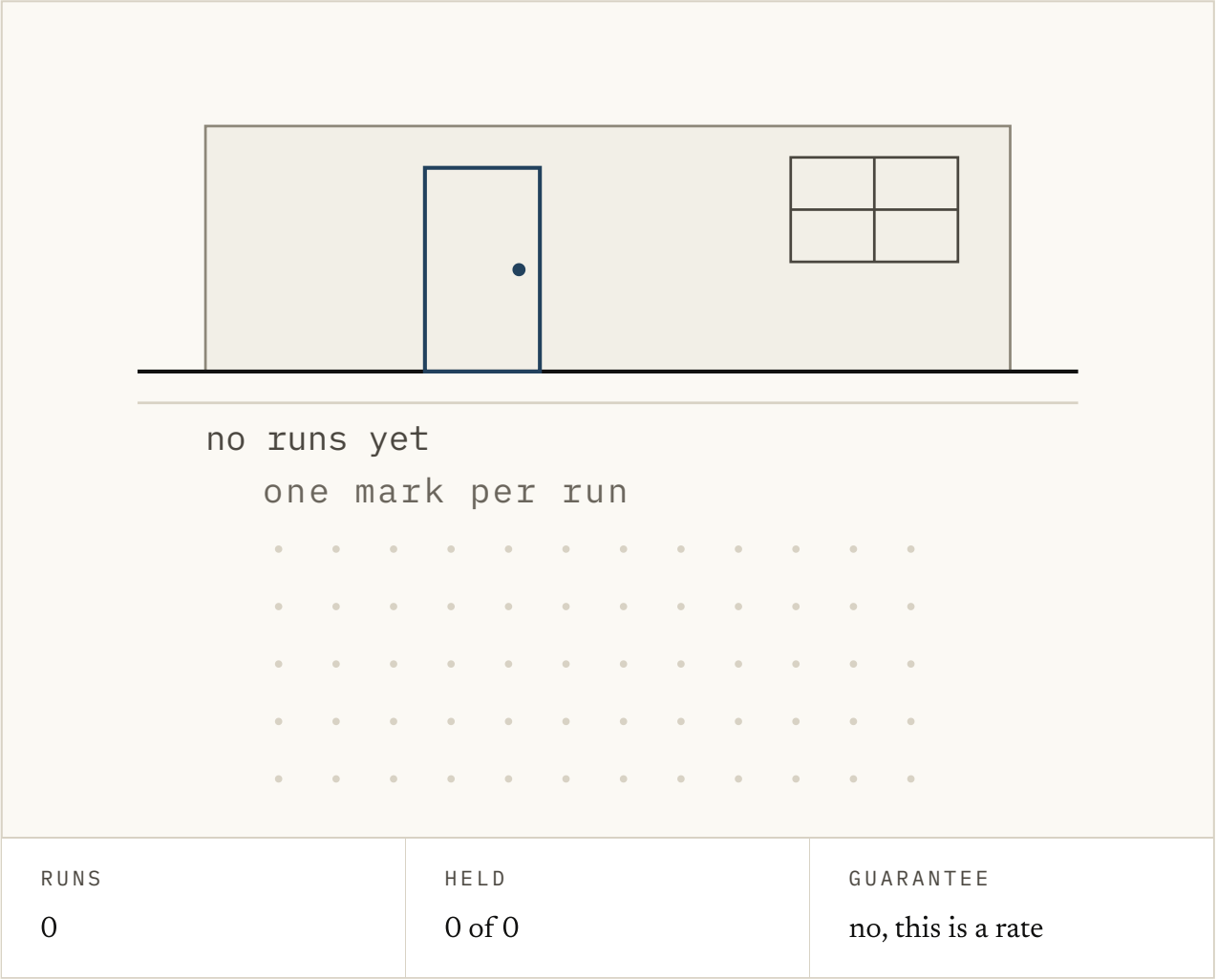


FIG. 8.10 One room, one door, and a walk away and back every time you press. Press Again a few times and the door comes back where it was, which is what usually looks like. Keep pressing and one run comes back with the door somewhere else. Then flip the switch to an engine holding the scene and there is nothing left to fail. The rate is illustrative.

The interesting moments are the vague ones. A model trained on pixels is pushed towards the average of the possible futures wherever the future is open. We saw in chapter 7 what that average looks like: a picture of something that cannot happen.

The future is most open when something is about to happen. So the steering-wheel test is hardest to pass exactly where passing it matters. Step along the clip in the figure below and you'll see why.

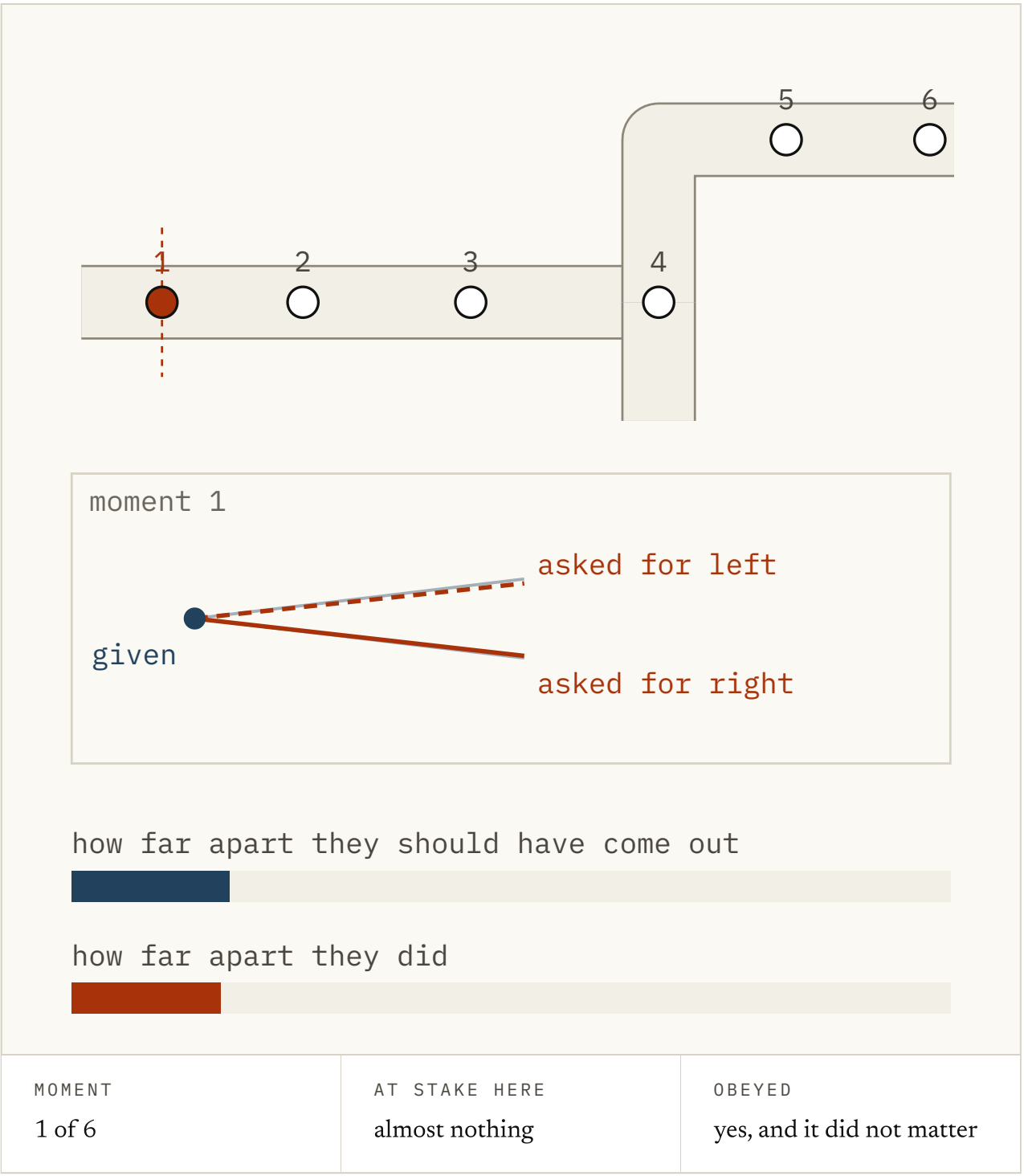


FIG. 8.11 One clip with a junction in it, and the steering-wheel test run at every moment along the way. Step through the moments and read the two bars: how far apart the two commands should have come out, and how far apart they did. In the corridor it obeys, and nothing was at stake. At the junction the two commands come out on top of each other. The moments and the numbers are illustrative.

The scores do not compose. Benchmarks measure visual quality, geometry, action fidelity and persistence one at a time. There is no accepted rule for combining them into one ranking.

So I will not tell you whether Genie 3 beats GameNGen. A claim about improvement belongs to one particular test. Click through the measures below and watch the winner change.

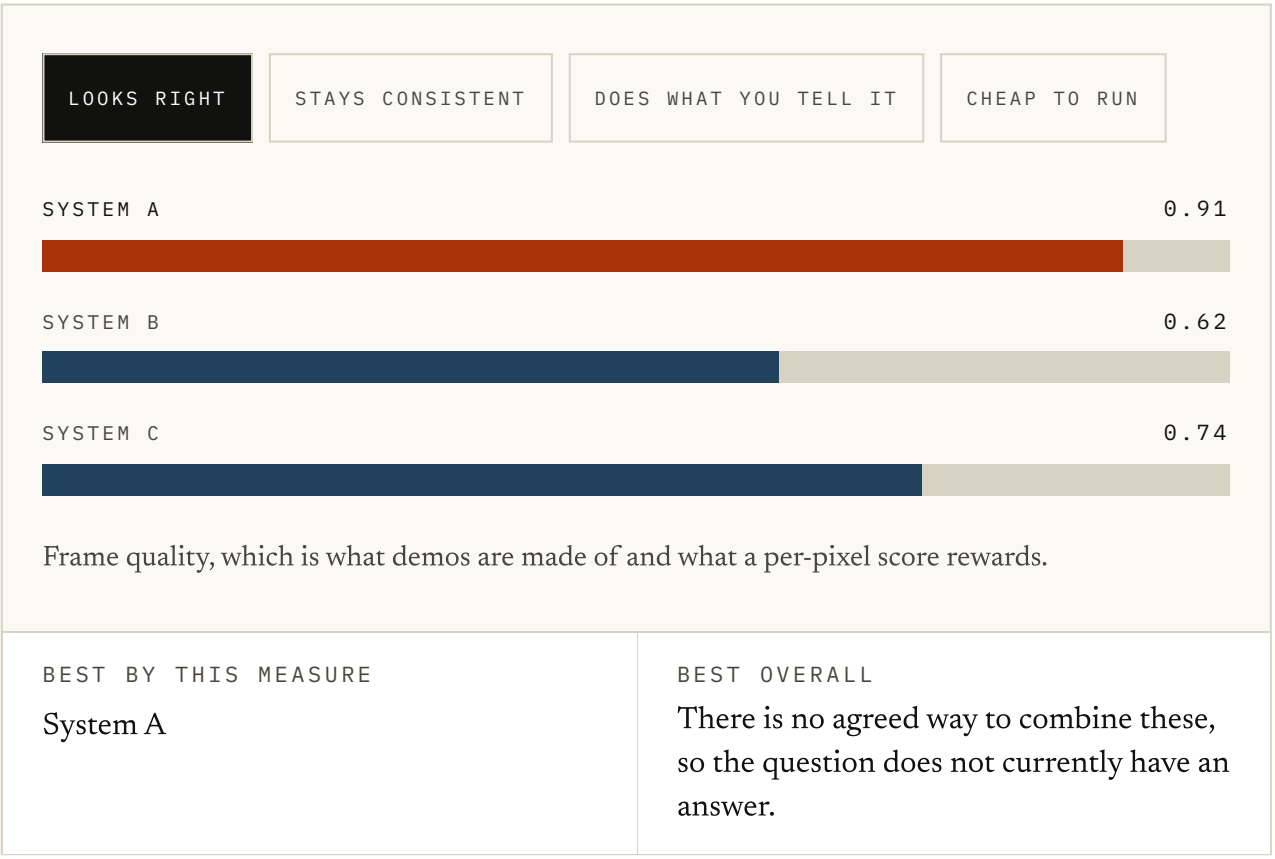


FIG. 8.12 The scores are invented and the systems are not real, but the disagreement is. Click through the four measures and watch the winner change. There is no accepted way to combine them into one ordering, so "better" only ever means better at one of these.

So, are video models world simulators? My answer is that they are somewhere you can be, and that is new and real. Whether one obeys you, and whether it holds the rules rather than the footage, are two separate measurements, and the demo you saw did not make either of them. Ask for the steering-wheel test, and ask what was outside the band.

Hopefully this chapter helped. There is a short quiz below if you want to check the ideas stuck. If I have got something wrong, or you know a better source than the ones I used, the About page has a corrections section and I would be glad to hear from you.

Try it

1 What change makes a video model capable of being steered, without proving that it will obey?

a. Higher resolution
b. A longer context window
c. More parameters
d. Conditioning each generated frame on an action as well as on the frames before it

> ANSWER d. Conditioning each generated frame on an action as well as on the frames before it. Action conditioning supplies the interface. Controllability still has to be tested by varying the action from the same start and measuring whether the requested futures separate.

2 A generated room stays as you left it when you turn back. What can you conclude about its memory?

a. Some internal state supports persistence, but the behaviour does not reveal whether it is a scene graph, context or another mechanism
b. It must use a separate object database
c. It must contain exportable geometry
d. It has no memory of any kind

> ANSWER a. Some internal state supports persistence, but the behaviour does not reveal whether it is a scene graph, context or another mechanism. Behaviour establishes persistence, not implementation. Genie 3 is reported to refer back to its trajectory, while exposing no persistent geometry or object store to inspect.

3 Dropped objects in generated video accelerate downwards, and nobody supplied gravity. What does that establish?

a. Nothing at all
b. The model has learned the law of gravity
c. That the regularity is in the training footage, and reproducing it is required to predict the footage well
d. The model contains a physics engine

> ANSWER c. That the regularity is in the training footage, and reproducing it is required to predict the footage well. The observation is real. The question is whether reproducing a regularity across the range you trained on amounts to having the rule, and the observation alone does not settle it.

4 In Figure 8.7, the model is within a few per cent across the band it was trained on. Why is that not evidence it has the rule?

a. The measurement is unreliable

b. Matching inside the band is equally consistent with having learned the answers to the questions it was asked
c. A few per cent is too large an error
d. The band was too narrow

> ANSWER b. Matching inside the band is equally consistent with having learned the answers to the questions it was asked. Having the rule and having a fit through the sampled region only come apart outside the band, and outside is where nobody tests because there is no footage to compare against.

5 Why is a generated video world the wrong shape for an agent to plan inside?

a. It cannot be conditioned on actions
b. It has no reward signal
c. It is not accurate enough
d. Planning means running the model many times over, and a frame is the opposite of a compact state

> ANSWER d. Planning means running the model many times over, and a frame is the opposite of a compact state. A planner needs to try many action sequences and score them cheaply. These models are expensive to run and produce frames rather than something small to reason over.

6 What are generated worlds unambiguously good for right now?

a. Producing training data and being a place you can move around in
b. Long-horizon planning
c. Compressing video
d. Replacing physics engines

> ANSWER a. Producing training data and being a place you can move around in. They avoid real-world contact, reset instantly and produce broad variation, while still costing compute. No other approach in this course gives you an actual picture to walk into.

7 Why does a wrong answer from one of these systems feel more convincing than a wrong number?

a. It usually is not wrong
b. The errors are smaller
c. It arrives as something that looks like a photograph
d. The models are better calibrated

> ANSWER c. It arrives as something that looks like a photograph. The extrapolation problem here is the ordinary problem with any fitted function. What is different is how persuasive the output is when it fails.

8 A system improves its geometry score but loses action fidelity. Is it better overall?

a. Yes, geometry is the only objective measure
b. Not without a declared use case or rule for combining the dimensions

c. No, action fidelity always dominates
d. Yes, any benchmark gain establishes overall progress

> ANSWER b. Not without a declared use case or rule for combining the dimensions. Benchmarks now measure several useful dimensions. The unresolved part is how to combine them when systems and applications value different contracts.

FIG. 8.13 Eight questions on the move and the claim. If a demo has just impressed you, go straight to the middle three.

Sources

Genie: Generative Interactive Environments BRUCE ET AL., 2024 ↗

Latent actions learned from unlabelled video, which is the move that turns a video model into somewhere you can be.

https://arxiv.org/abs/2402.15391

Genie 3 GOOGLE DEEPMIND, 2025 ↗

The reported numbers this chapter quotes: 720p, 24 frames a second, minutes of coherence. A lab report rather than a result anyone outside has repeated.

https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/

Diffusion Models Are Real-Time Game Engines (GameNGen) VALEVSKI ET AL., 2024 ↗

DOOM generated frame by frame from previous frames and inputs, fast enough to play. The clearest demonstration that this is playable rather than merely watchable.

https://arxiv.org/abs/2408.14837

How Far is Video Generation from World Model: A Physical Law Perspective KANG ET AL., 2024 ↗

The measured version of Figure 8.7. Video models match physical laws within the distribution they were trained on, and do not extrapolate outside it.

https://arxiv.org/abs/2411.02385

Video generation models as world simulators OPENAI, 2024 ↗

The report that made the emergent-physics claim a mainstream one. Worth reading for exactly what it does and does not assert.

https://openai.com/index/video-generation-models-as-world-simulators/

Cosmos World Foundation Model Platform for Physical AI NVIDIA, 2025 ↗

Generated video built as training data for robots and vehicles, which is the use these systems are unambiguously good for.

https://arxiv.org/abs/2501.03575

Cosmos NVIDIA ↗

The product framing, and a useful boundary case: generative video beside explicit simulation, sold as one platform.

https://www.nvidia.com/en-us/ai/cosmos/

FIG. 8.14 I've ordered these for someone building on this rather than for historical completeness. Kang and the OpenAI report sit together as claim and measurement.

09 What Is Still Broken in World Models?

BY NILUSHANAN KULASINGHAM

11 MIN READ · INTERACTIVE

> Scenes do not fail all at once, and different systems fail under different contracts. The closing argument: what to test, which benchmark claims compose, and which do not.

The figures in this chapter are interactive. Press and drag them online at worldmodels101.com/chapters/whats-broken.

Every system in this course ends on something that does not work yet. I think the failures belong together, because the confusion around the phrase was built out of them. Five kept turning up: persistence, action, horizon, target and verification. Each was found by a different community, and each was measured late or not at all.

Chapter 1 met the five machines the phrase can mean: the renderer, the simulator, the dynamics model, the representation and the implicit model.

PERSISTENCE

Does the same object remain itself after it leaves view and returns?

ACTION

Do different commands reliably cause the corresponding different futures?

HORIZON

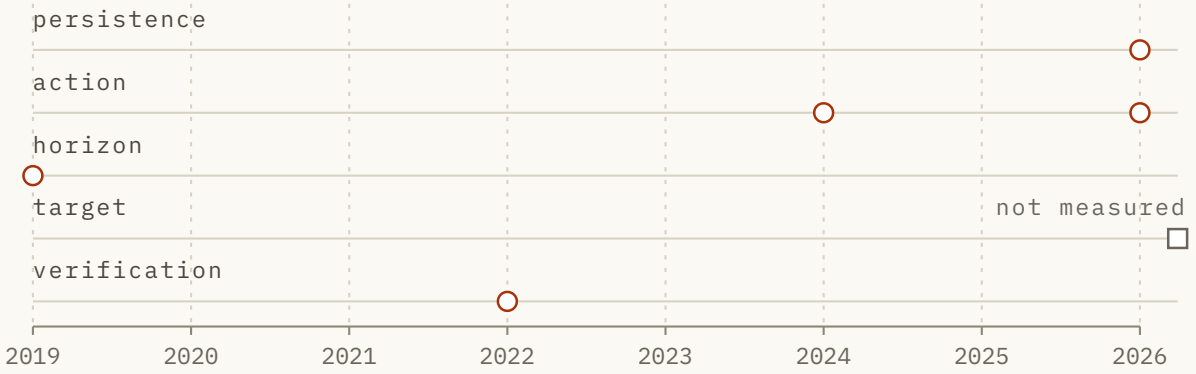
How far can it run on its own outputs before the answer stops supporting a decision?

TARGET

What information is the training target allowed to discard or collapse?

VERIFICATION

What observation could prove the claimed internal model is not there?



Five failures, five communities. Read the dates from left to right.

WHO RAN INTO IT FIRST

Click a card above.

FIRST MEASURED

Click a card above.

FIG. 9.1 The five failures, laid out. Click one to read the question it asks, which community ran into it first, and when somebody first measured it. Then read the dates along the bottom from left to right. The order is the odd part.

Click through the five above before you read on, and notice which one each system hits first.

Rollouts do not fail all at once

Let's start with the rollout, because that is where most of the failures show up. A rollout is the model's predictions run forward step after step, each one built on the last. The usual report is one average error over the whole run. That melts several failures into one number and hides the order they happened in.

Identity can go while texture stays sharp. Action fidelity, whether the action you gave it is the action it took, can go while both still look fine. Which goes first depends on the system and the task. So the useful report is one horizon per property: how many steps each one held up for.

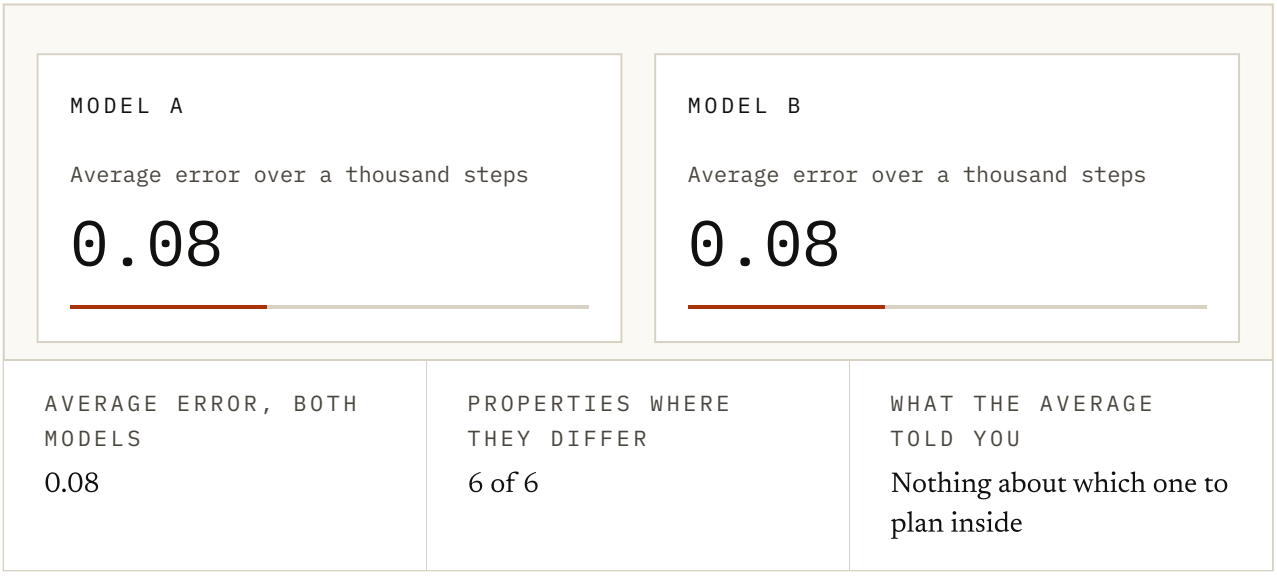


FIG. 9.2 Two models with the same average error over a thousand steps. Leave the report on one number and there is nothing to choose between them. Switch it to one horizon per property and read the two rulers: the same average was hiding two different machines. Click a property to see how far apart they are on it. The profiles are illustrative.

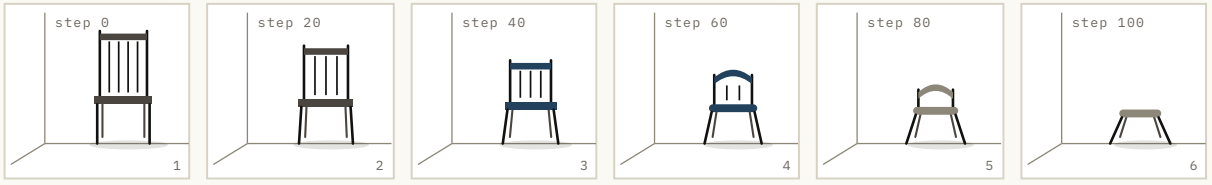
In both profiles, colour and texture outlast identity and physics, and the loss explains why. A per-frame loss pays the model for getting most of the pixels about right, and most pixels are surfaces and light. Nothing in it is watching whether the chair you walked past is the same chair.

The result is what I call the single-frame alibi. You get a rollout a hundred steps past the point of being useful, and any one frame of it still looks completely fine. The mistake only shows when you look along the sequence.

Three failures that outlived every fix

Things stop being themselves. Object permanence is the expectation that a thing is still there, and still itself, when you look back. It is the failure everyone notices, and the one chapter 1's turn-around test was built to catch and could not.

Take a renderer, a model whose output is pictures. It has no object store, so identity lives in the predictor's hidden state, the numbers it carries between steps. When the chair becomes another chair, it does so a little at a time and without an error message. A simulator stores the object explicitly, but then needs the structure right first.



SELECTED FRAME	THIS FRAME ON ITS OWN	SAME CHAIR AS FRAME 1?
Frame 6, step 100	looks fine	no
HORIZON	**FRAMES THAT LOOK FINE ON THEIR OWN**	**FRAMES SHOWING THE SAME CHAIR AS FRAME 1**
100 steps	6 of 6	1 of 6

FIG. 9.3 Drag the horizon and look at any one frame on its own: sharp edges, plausible light, nothing to complain about. Now look along the strip. Leave the switch on hidden state and the chair turns into a different chair a little at a time, with no frame you can point to as the mistake. Flip it to object store and every frame shows the same chair, because a stored object is being read back. The frames are illustrative.

What if is not really available. A model can tell you what is likely to happen next. What would have happened if you had done something else is a different question, and it is the one a decision needs. The model's answer to it comes from the model, not from the world.

Bingyi Kang and colleagues measured the gap in 2024, in *How Far is Video Generation from World Model*. They tested video generators on physical laws inside and outside their training data. Inside, the generators obeyed the laws. Outside, they broke them, and nothing in the output said which case you were in.

> I think this one gets under-rated because the model always answers. A system that said "I have never seen anything like that" would be more useful and less impressive, and nobody is rewarded for building it.

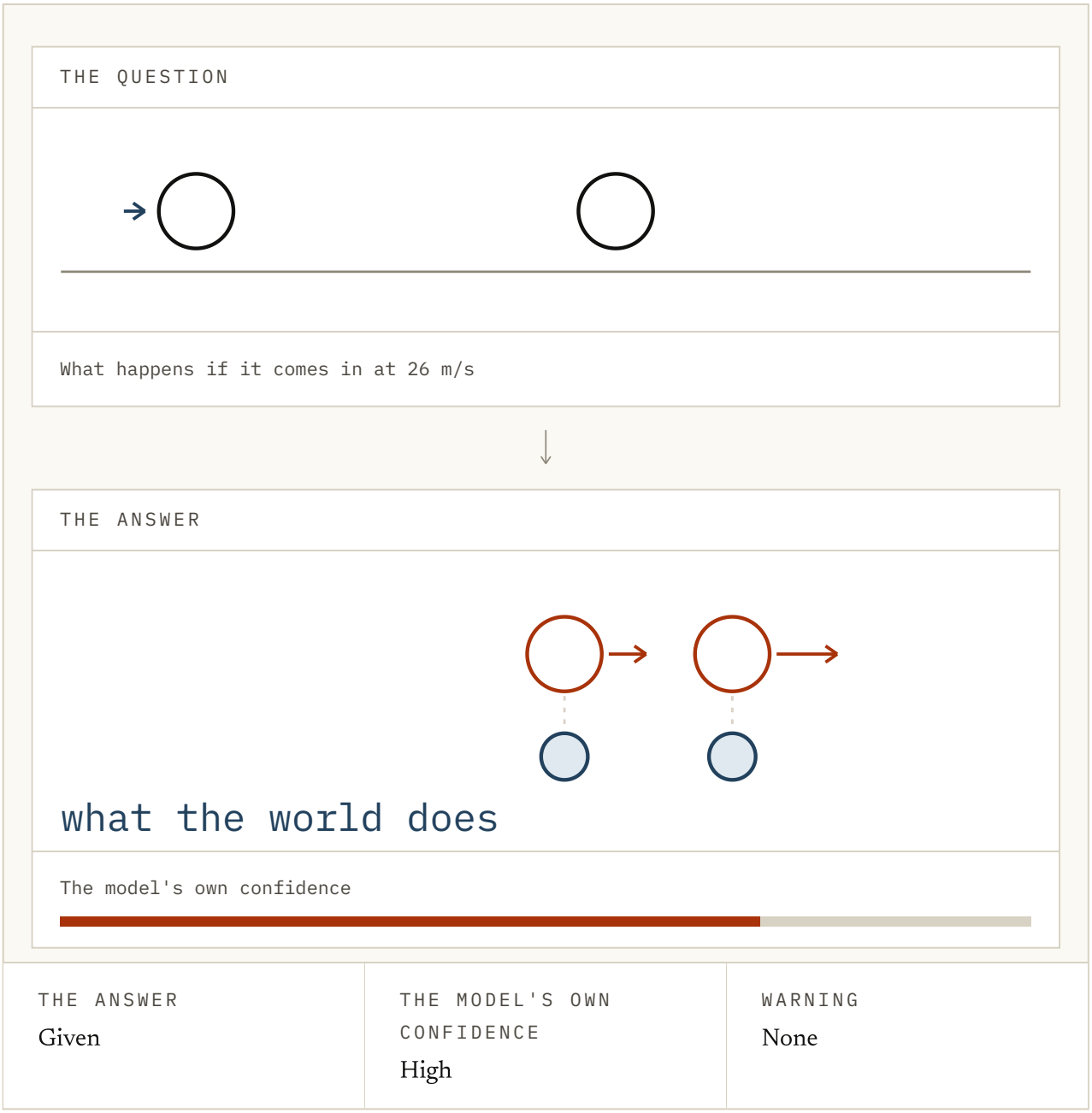
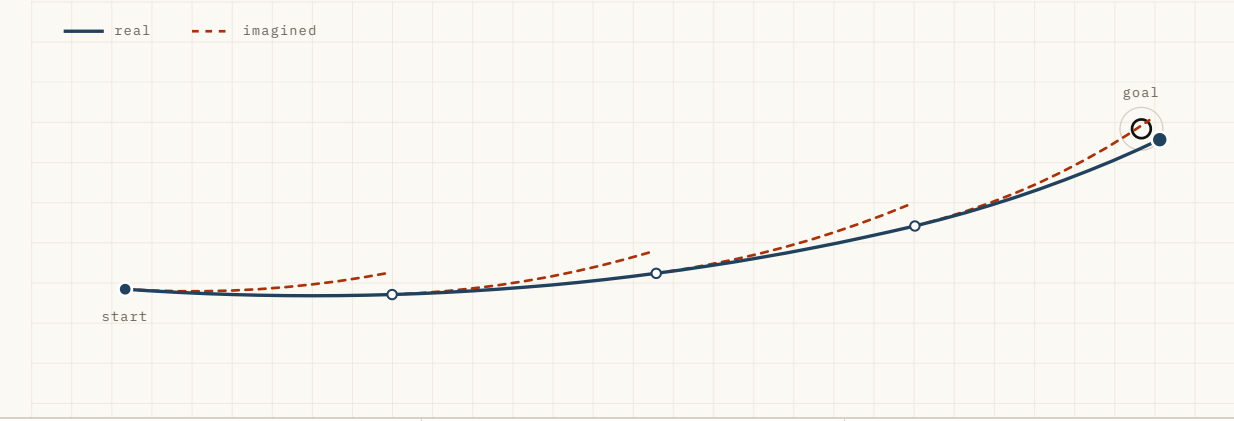


FIG. 9.4 Three questions put to the same generator: a speed the footage had, one just past it, and one nothing like it. Step through them and watch the answer card. It arrives every time, drawn the same way, with the same confidence, and only the slate outcome beside it says when it was wrong. Then switch on the flag a useful system would raise and see which questions it catches. The speeds and the outcomes are illustrative.

Long horizons are still the binding constraint. Most tricks in model-based control are ways of not needing the model to be right for very long. Horizon was the first of the five to be measured. It was measured by the people who had to act on the model's predictions, in reinforcement learning.

Tingwu Wang and colleagues did it in 2019, in *Benchmarking Model-Based Reinforcement Learning*. They ran the methods that learn a model of their environment and plan inside it side by side. The planning horizon, meaning how many steps ahead the model was trusted, decided where each won and lost.

Michael Janner and colleagues answered the same year, under a title that is still the question: *When to Trust Your Model*. Their answer was only briefly, and from a real state. Short rollouts and replanning every few steps work because they stop relying on the model before it drifts. The horizon over which it holds still decides what you can build.



STEPS TRUSTED	REPLANS	DISTANCE FROM GOAL AT THE END
10	3	reached

FIG. 9.5 Janner's answer, made playable. The slate path is what really happens and the vermilion path is what the model imagines. Drag the steps trusted out to thirty and press Run: the imagined path leaves the real one, so the plan is for a world that is not there. Pull it back to three and run again. Now the planner snaps back to the real state every few steps, and the errors never get the chance to pile up. The numbers are illustrative.

The league table nobody can publish

Measurement arrived in the wrong order, and I want to walk you through why. Reinforcement learning measured horizon first, because an agent that trusts a bad model loses, and the loss shows up in the score. The video people came next and

measured what they could, which was the frame.

VBench came in 2023, from Ziqi Huang and colleagues. It scored generated video on separate dimensions, such as frame quality and how smoothly things moved. It used enough of them that the ordering stopped being obvious.

Benchmarks that test worlds you act inside, rather than watch, came last. PlayWorld, released in August 2026, tests 171 scenarios. They cover geometry, interaction fidelity (does an action do what it should) and what happens to objects as they leave and re-enter view. That is persistence and action, the two failures a renderer hits first, tested seven years after horizon.

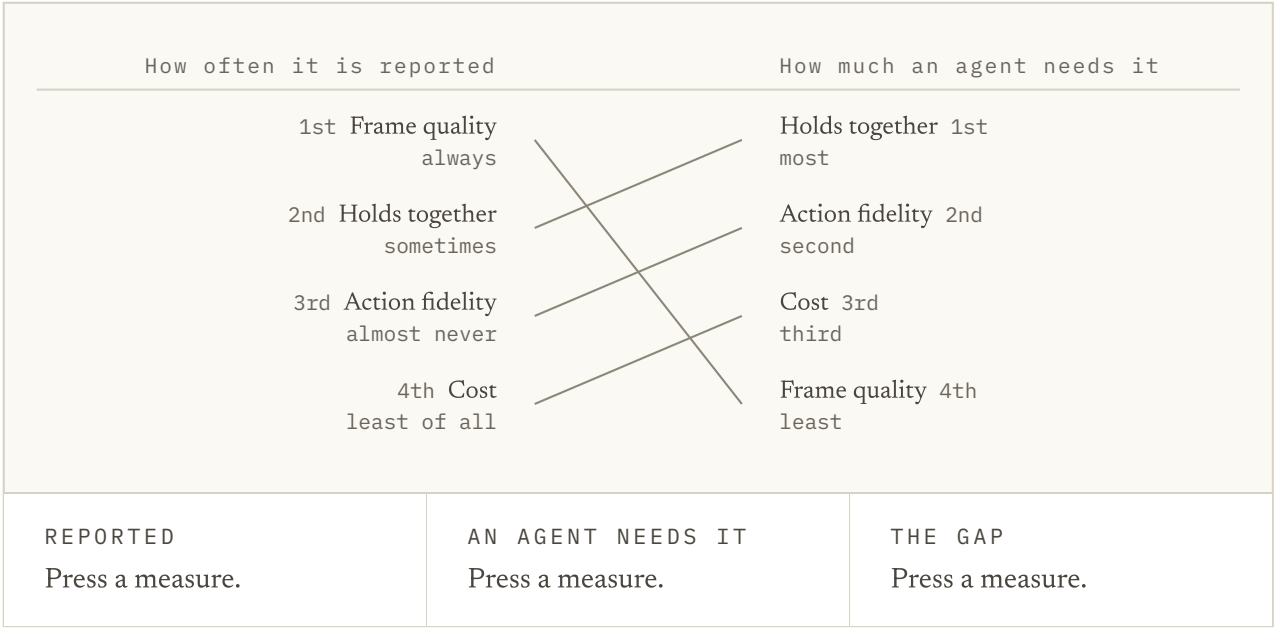


FIG. 9.6 The four measures, ordered twice. On the left, how often they get reported; on the right, how much an agent needs them. Press a measure to draw its line across. Nearly every line crosses, and that is the whole of what I mean by the wrong order. The orderings are my judgement rather than a survey.

Frame quality is measurable, and it is what demos are made of. Whether the world holds together over a thousand steps is what an agent needs, and no demo is long enough to show it. Action fidelity separates a world model from a video generator, and it is almost never reported. Cost is reported least of all.

That order is close to the reverse of how much each matters, because the easy measures were taken first. Downstream task success looks like the way round this. But it scores the model and the policy together, and a good policy hides a bad model.

Progress is real. Genie 3, from Google DeepMind in 2025, is said to stay coherent for several minutes. That is reported by the lab that built it, on a measure it chose. "Better" means nothing until the measure is named, and the system contract with it, meaning which of the five machines is claimed.

Match the failure test to the system

The phrase covers five machines, and for most of this history they sat the same exam. That is where the confusion came from. A renderer depends on persistence and action fidelity, and the single-frame alibi is its usual failure. A simulator is the easy one to check, because it exports structure that another program can test.

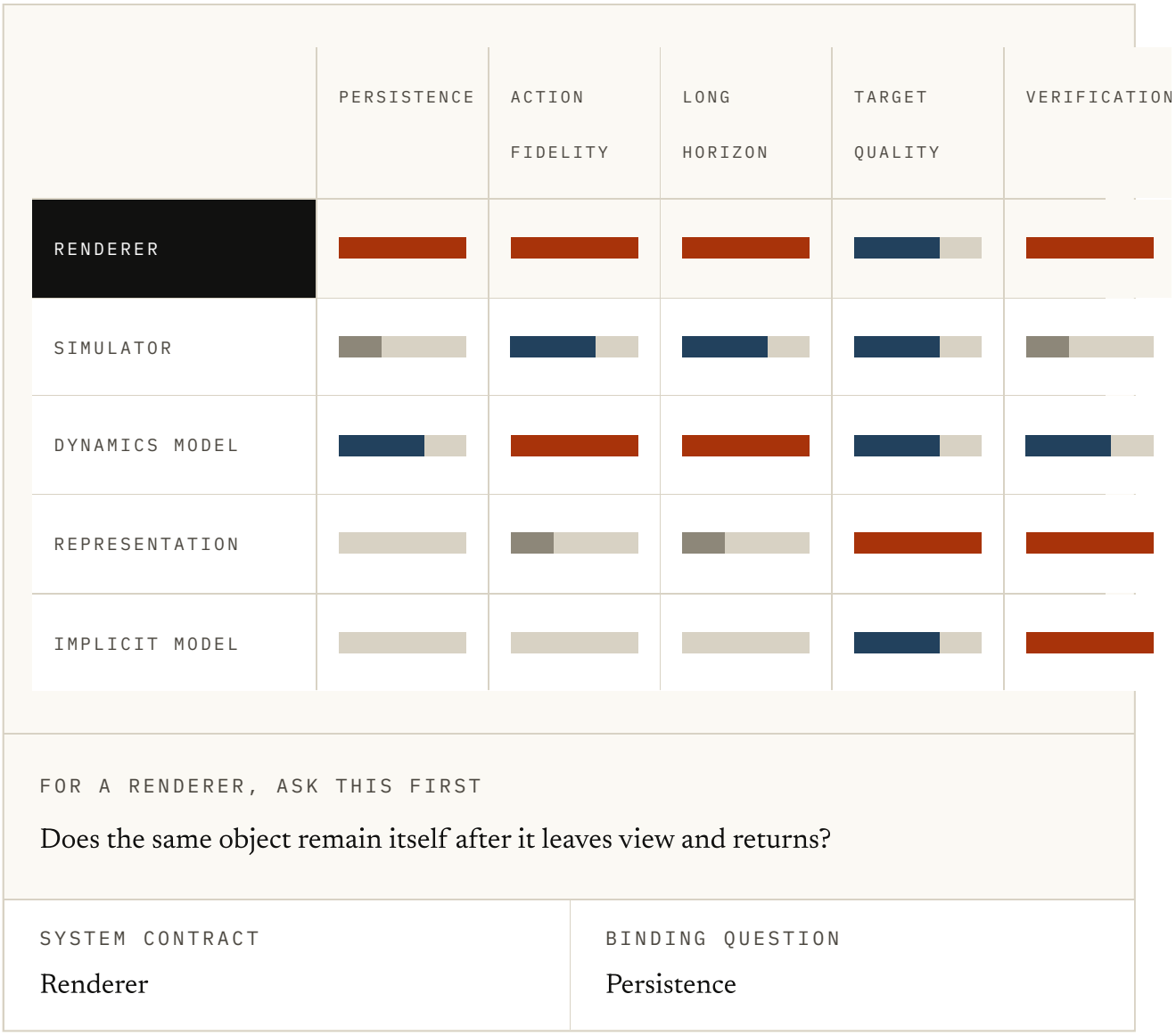


FIG. 9.7 Five system contracts crossed with the five failure questions. The levels are editorial judgements, mine, rather than benchmark scores. Select a row to see the question that most often binds that kind of system, and read the renderer and simulator rows against the paragraph above. The next three paragraphs walk the other three rows. A single "world model quality" axis would flatten all five.

A compact dynamics model predicts a small state rather than pixels. It depends on rollout horizon, and on whether planning search can exploit it. A representation, an embedding learned for something else to use, can look stable while throwing away the variable the task needed. I call that the target problem: predicting the wrong thing well.

An implicit model is a claim that a model formed inside a network trained for some other job. Checking it needs a probe, a small second network trained to read one fact out of the first's internals. Kenneth Li and colleagues showed what that takes in 2022, in *Emergent World Representations*.

A network trained only to predict the next move in Othello (a board game played with two-sided discs) turned out, under probes, to hold a picture of the board. The probe could have found nothing.

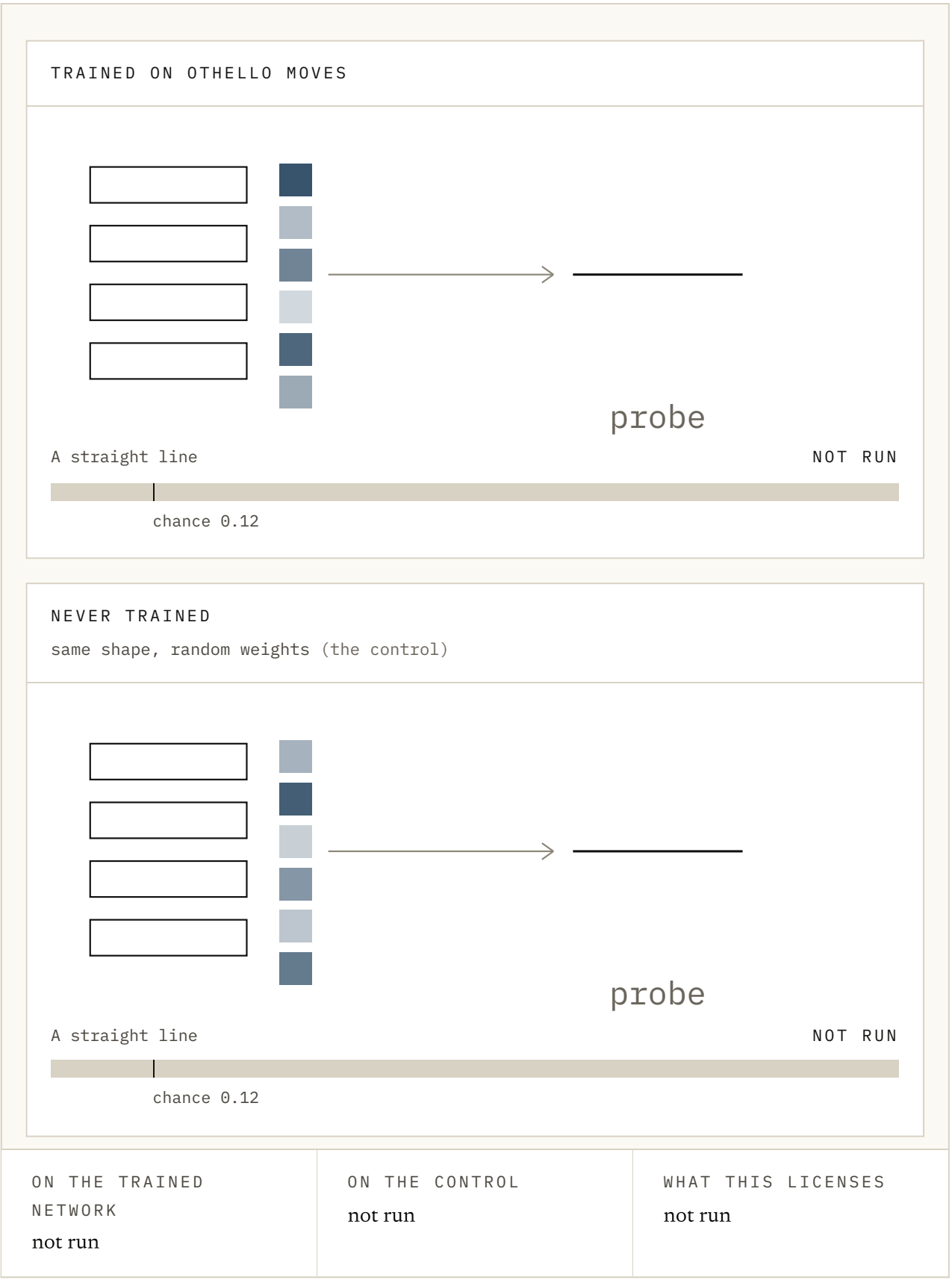


FIG. 9.8 The same probe run against two networks: the one trained on Othello moves, and one of the same shape that never trained. Press Run the probe and compare the two bars against the chance line. Then let the probe be a big network instead of a straight line, run it again, and it reads a board out of the untrained one too. That is why the probe has to be small. The accuracies are illustrative.

Four questions still handle most papers, so let me leave you with them. The first is **what does it output?** A picture, a structure, a compact state or an embedding, and the other three questions depend on the answer.

The second is **can you roll it forward under actions nobody took?** If not, it will not support a decision, however good it looks.

The third is **over what horizon does it hold up, and who measured that?** The number matters more than the architecture, and it is usually missing.

The fourth is **which measure is the claim using, and what does it ignore?** There is no overall score, so anyone implying one has picked a measure and not told you which.

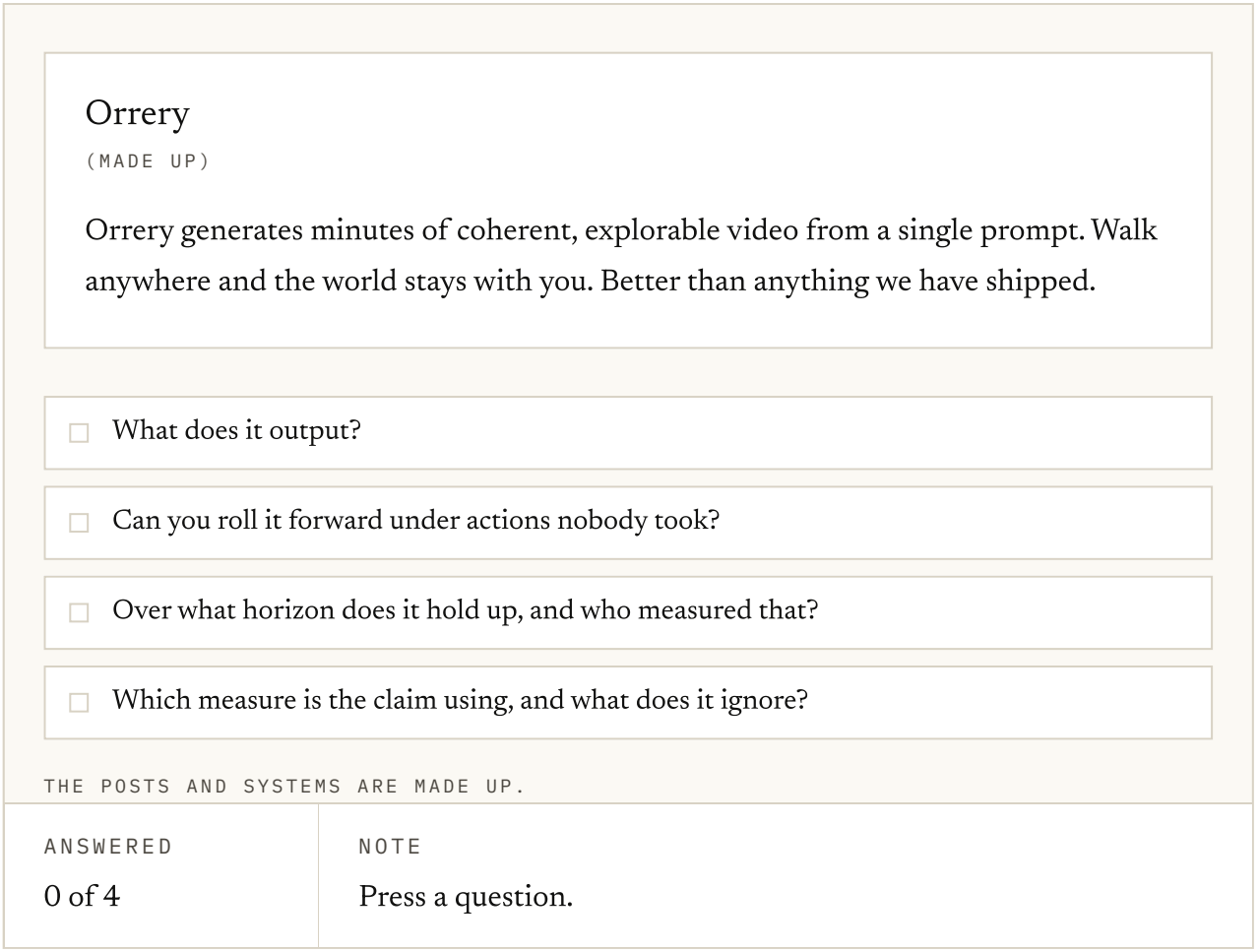


FIG. 9.9 The four questions, put to three made-up launch posts. Pick a post, then press each question in turn. Where the post answers it, the words that answer it light up. Where it does not, you get "not stated", and the counter at the bottom keeps score. Try all three posts, and you should find that the most exciting one answers the fewest. The posts are invented, and so are the systems in them.

Yann LeCun, then Meta's chief AI scientist and a Turing Award winner, wrote the fullest statement of the target in 2022. The paper is *A Path Towards Autonomous Machine Intelligence*, and it sets out what a world model would have to do to deserve the name. Read against the four questions, it is a list of what is still missing, and naming an architecture answers none of them.

That brings us to the end of the course. The phrase still means five different things, and the people using it still rarely say which. What has changed is on your side: you can now tell which of the five is being offered, what it would take for the claim to be true, and what to ask when somebody is not saying.

Thank you for reading this far. Hopefully these nine chapters helped, and there is a short quiz below if you want to check the ideas stuck. If I have got something wrong anywhere in the course, or you know a better source than the ones I used, the About page has a corrections section. I would be glad to hear from you.

Try it

1 Why is one average error insufficient as a long-rollout report?

a. Different properties can fail at different horizons, and an average hides which contract was lost
b. The error is measured in the wrong units
c. Rollouts are too short to average over
d. Averages are always misleading

> ANSWER a. Different properties can fail at different horizons, and an average hides which contract was lost. There is no universal failure order. The point is to report separate horizons for identity, physics, control and appearance instead of allowing one strong dimension to conceal another.

2 A rollout still looks completely fine in any single frame. What does that tell you?

a. The horizon has not been reached
b. It is still usable
c. Very little. Most of the pixels are surfaces and light, and nothing in a per-frame loss is watching whether objects stayed themselves
d. The model has learned physics

> ANSWER c. Very little. Most of the pixels are surfaces and light, and nothing in a per-frame loss is watching whether objects stayed themselves. This is how the subject fools you, and it is why demos are weaker evidence than they feel.

3 Why is object permanence fragile in a renderer with no exposed object store?

a. Occlusion is computationally expensive
b. Training data lacks occluded objects
c. The models are too small
d. Identity is carried implicitly by predictor state rather than enforced by a persistent record

> ANSWER d. Identity is carried implicitly by predictor state rather than enforced by a persistent record. A structured simulator may store objects explicitly. In the renderer case, identity can drift gradually because persistence is an inferred behaviour rather than an inspectable record.

4 You ask a model what would have happened if you had acted differently. What have you got?

a. A fact about the world
b. An answer that is true inside the model, with nothing in it indicating whether the model had data there
c. A proof
d. Nothing, models cannot answer that

> ANSWER b. An answer that is true inside the model, with nothing in it indicating whether the model had data there. Where the model has data the two are close. Where it does not, they are not, and the model answers with equal confidence either way.

5 Short rollouts, replanning and starting from real states all work. What do they have in common?

a. They are all ways of not needing the model to be right for very long
b. They only apply to robotics
c. They remove the need for a policy
d. They make the model more accurate

> ANSWER a. They are all ways of not needing the model to be right for very long. Read uncharitably, the whole toolkit is an admission. The horizon over which a model holds up is the number that decides what you can build.

6 Why does downstream task success not settle which model is better?

a. It only works in simulation
b. Tasks are too easy
c. It scores the model and the policy together, so a good policy hides a bad model
d. It cannot be measured reliably

> ANSWER c. It scores the model and the policy together, so a good policy hides a bad model. It sounds careful, and it conflates the two things you were trying to tell apart.

7 A benchmark winner is best on geometry but mediocre on action fidelity. What is the first thing to ask?

a. What hardware it runs on
b. How long it trained for
c. How large is it
d. Which measure, and what does that measure ignore

> ANSWER d. Which measure, and what does that measure ignore. Benchmarks such as PlayWorld make useful dimensions explicit. They do not remove the need to say which contract matters for the intended use or how competing dimensions were combined.

8 Across the whole course, which single question does the most work on an unfamiliar system?

a. How many parameters does it have
b. What does it output, and can you roll it forward under actions nobody took
c. Is it open source
d. Does it use a transformer

> ANSWER b. What does it output, and can you roll it forward under actions nobody took. The first half decides what it can be used for. The second decides whether it can support a decision. Almost everything else follows, and neither depends on knowing the architecture.

FIG. 9.10 Eight questions on what is still broken. If you have read the whole course, the last one should be easy and slightly annoying.

Sources

PlayWorld: A Benchmark for Interactive World Models ZHANG ET AL., 2026 ↗

A current benchmark spanning 171 scenarios and separating geometry, interaction fidelity, and out-of-sight evolution rather than pretending world quality is one number.

https://arxiv.org/abs/2608.13552

How Far is Video Generation from World Model: A Physical Law Perspective

KANG ET AL., 2024 ↗

Physical laws matched inside the training distribution and not extrapolated outside it. The clearest measured statement of the gap this chapter is about.

https://arxiv.org/abs/2411.02385

VBench: Comprehensive Benchmark Suite for Video Generative Models

HUANG ET AL., 2023 ↗

A serious attempt at the measurement problem, and useful for seeing how many separate dimensions it takes before the ordering stops being obvious.

https://arxiv.org/abs/2311.17982

Benchmarking Model-Based Reinforcement Learning WANG ET AL., 2019 ↗

Where model-based methods win and lose, and the discovery that the planning horizon is the number that decides it.

https://arxiv.org/abs/1907.02057

When to Trust Your Model: Model-Based Policy Optimisation JANNER ET AL., 2019 ↗

Short rollouts as an answer to a model you cannot trust for long. The title is this chapter's question.

https://arxiv.org/abs/1906.08253

Emergent World Representations LI ET AL., 2022 ↗

The strongest available evidence that something structured forms inside a predictor, and a good example of what it takes to show it rather than assert it.

https://arxiv.org/abs/2210.13382

Genie 3 GOOGLE DEEPMIND, 2025 ↗

Reported coherence over minutes, from the lab that built it. Included as an example of exactly the kind of claim this chapter is asking you to read carefully.

https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/

A Path Towards Autonomous Machine Intelligence LECUN, 2022 ↗

The most complete statement of what a world model would have to do to be worth the name, which doubles as a list of what is still missing.

https://openreview.net/forum?id=BZ5a1r-kVsf

FIG. 9.11 I've ordered these for someone building on this rather than for historical completeness, benchmarks first and LeCun last.