02 How Do World Models Work?

BY NILUSHANAN KULASINGHAM

25 MIN READ · INTERACTIVE

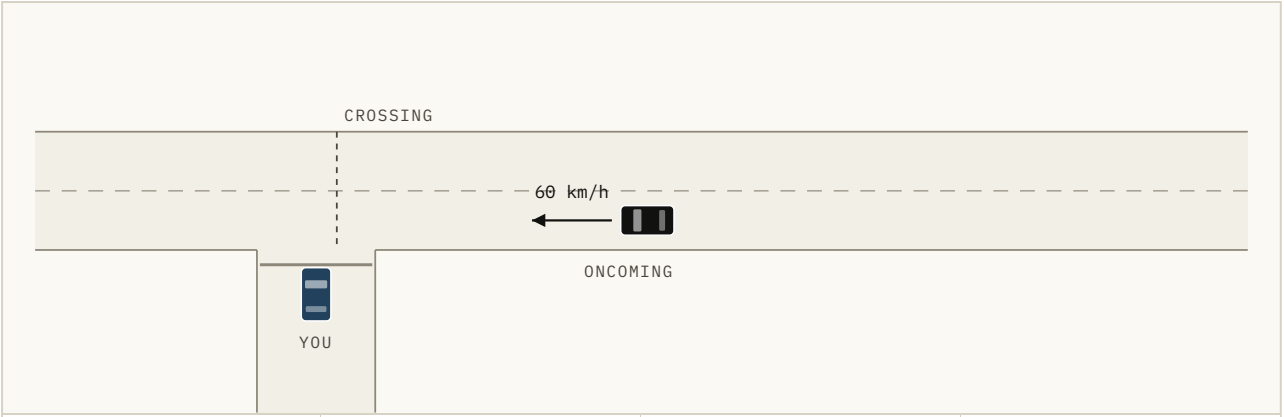
> A model you can run forward lets you try an action before paying for it. That is the oldest idea in the field and still the best one. The trouble is that a search good enough to find the best plan is also good enough to find the places where the model is wrong. Nobody has engineered that away.

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.



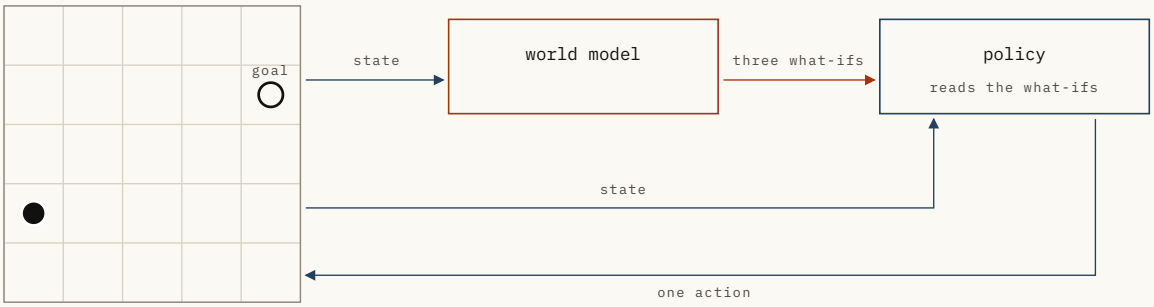
FORECAST ARRIVAL	REAL ARRIVAL	FORECAST WAS OFF BY	OUTCOME
3.6 s	after Go	after Go	not yet

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.



MODEL CALLS

0

WORLD STEPS

0

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.

$$\hat{p}(z_{t+1},\ r_t \mid z_t,\ a_t)$$

given where you are now and an action you might take
, what to expect for where you end up
and what the step was worth.

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.

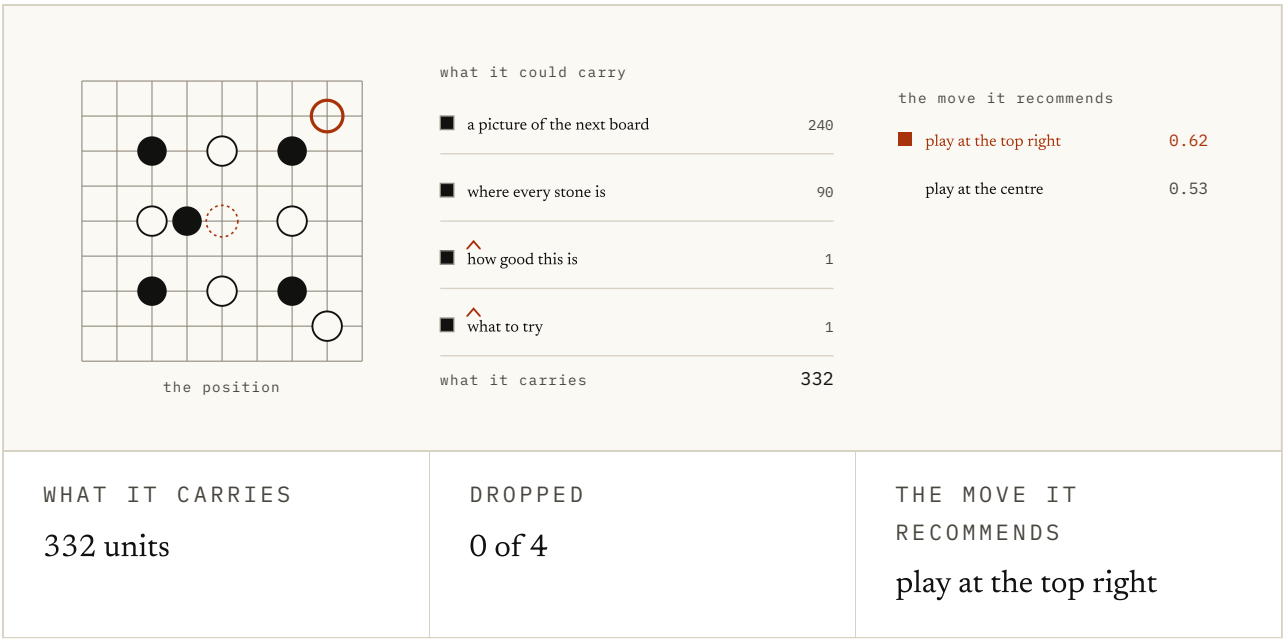


FIG. 2.4 One position, and four things a model could carry. Press to drop the picture of the next board, then drop the stones, and watch the move it recommends stay where it was. Drop either of the two numbers instead and there is nothing left to recommend with. Both survivors wear the hat. The position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

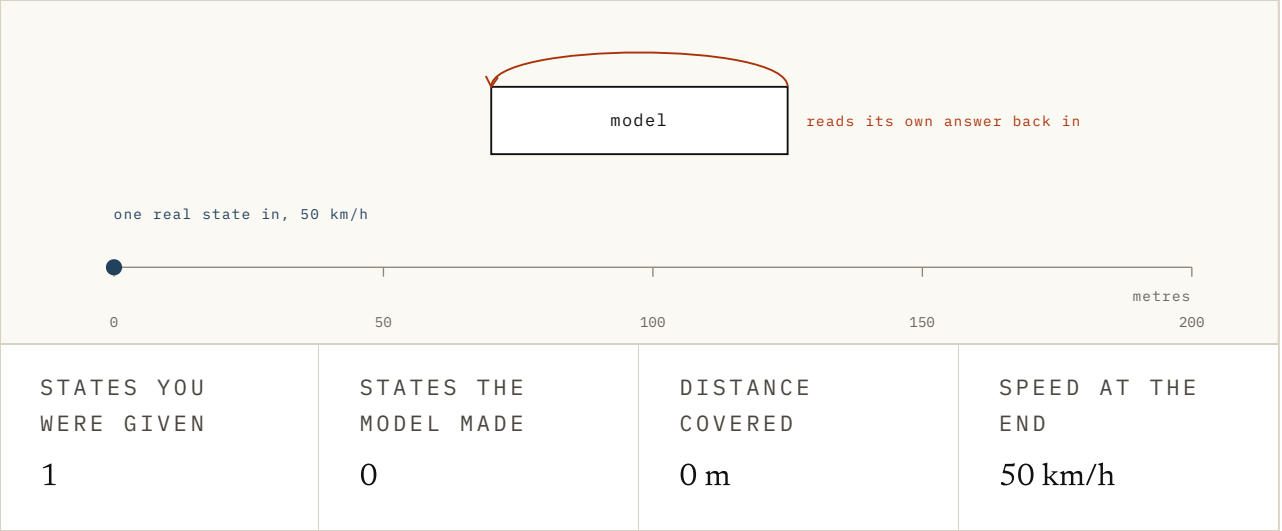


FIG. 2.5 A run of actions nobody has taken. Click a step to change what you do there, drag the horizon to set how long the run is, and press Run: the model answers the first step, then reads its own answer back in for the second. Press Keep to pin a run and write another one beside it. Nothing in here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

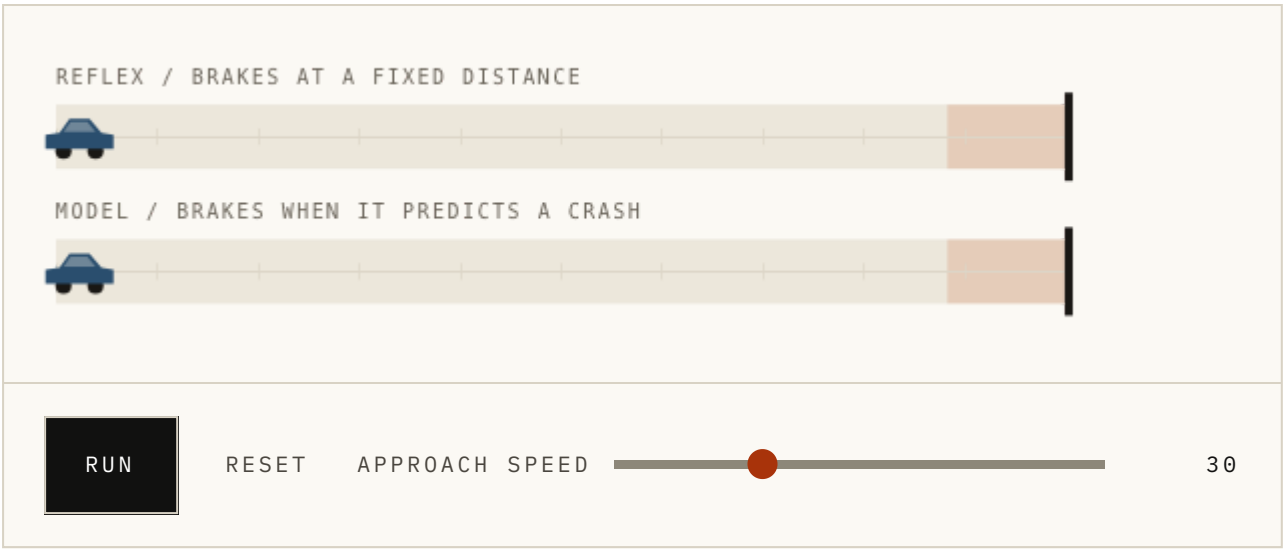


FIG. 2.6 Below about forty you cannot tell the two apart. Drag the speed up and watch which one notices. What is being compared is one fixed rule against one object that recomputes.

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.

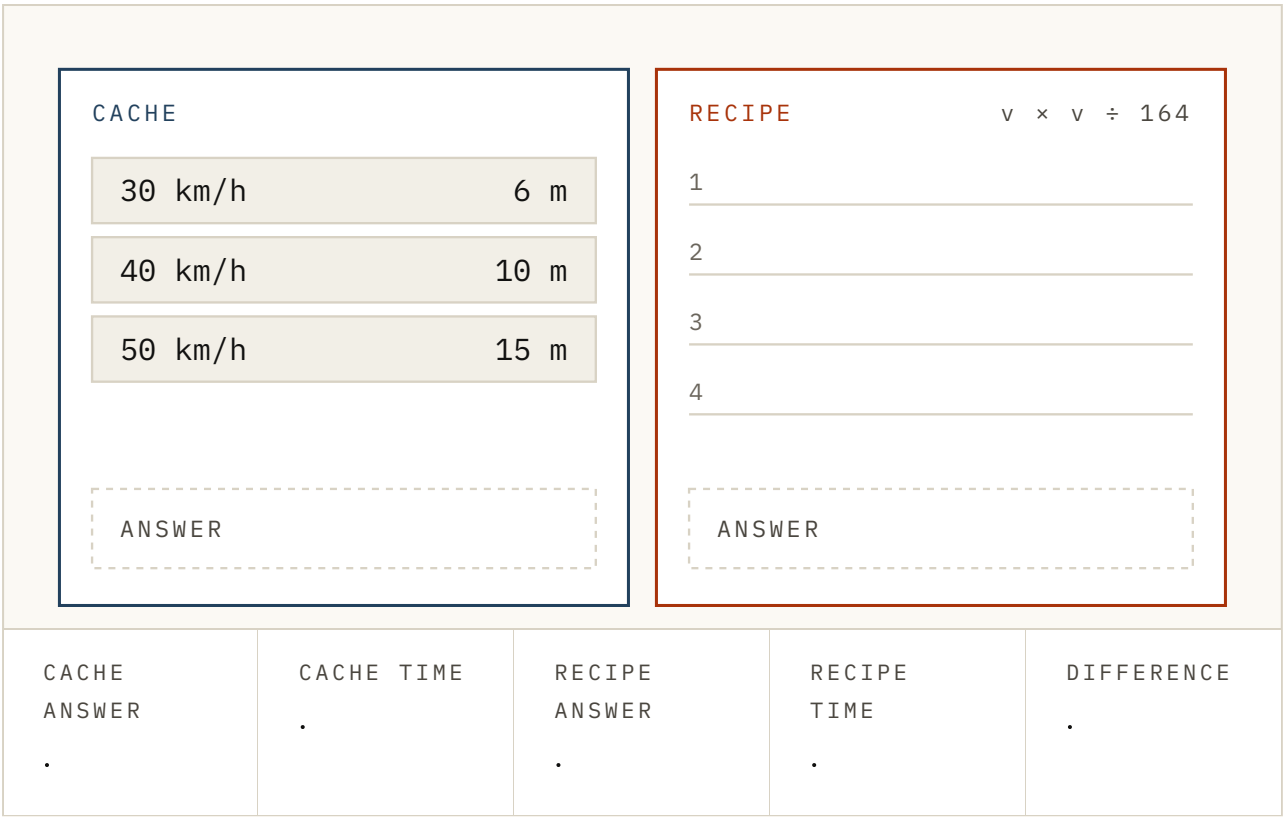


FIG. 2.7 A cache and a recipe, asked the same questions. Press a speed on the left and both answer. Then press one the cache has never seen: it hands back its nearest entry, and the recipe works the answer out from scratch. Watch the two time readouts as well. Illustrative.

> 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.

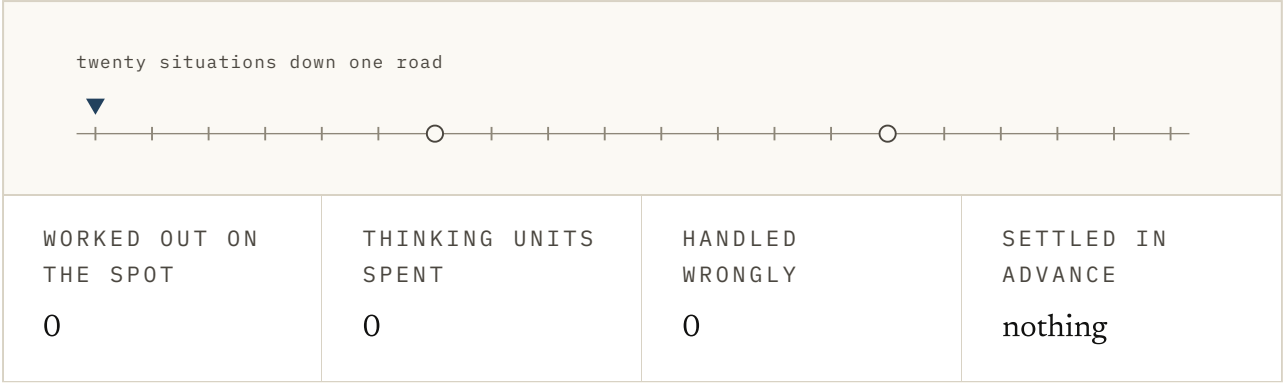


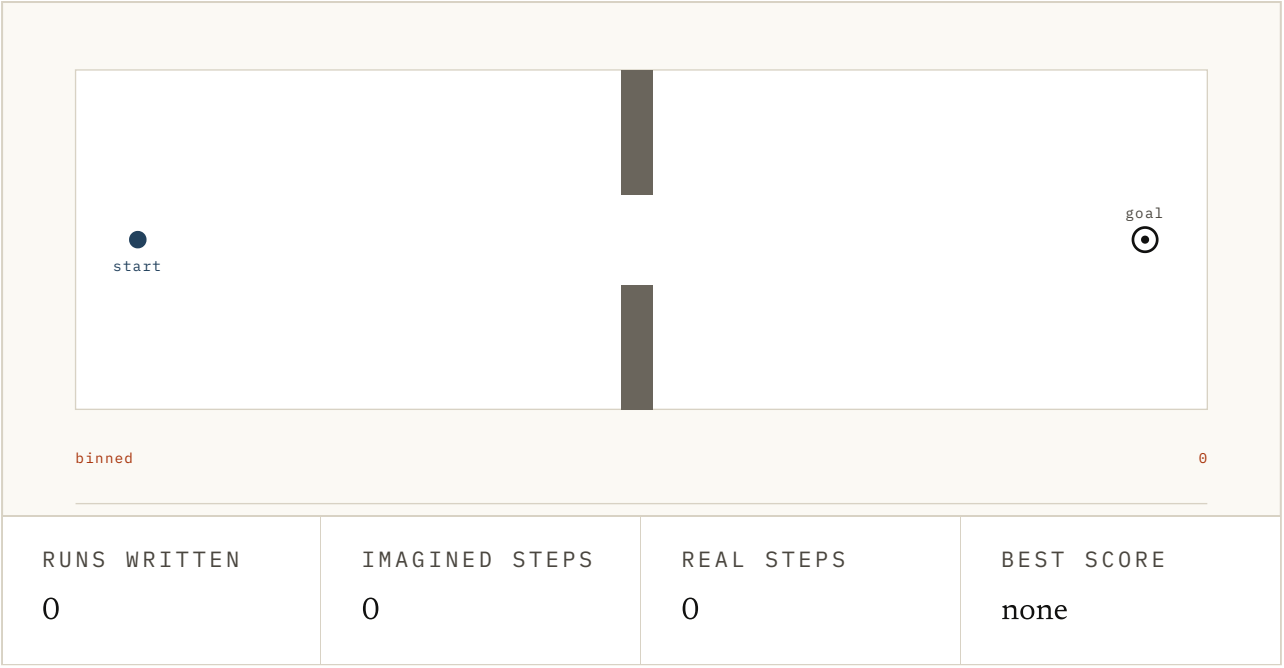
FIG. 2.8 Twenty situations down one road, and one control: how much of the answer was settled before the road was seen. Press Drive to step through, and watch which decisions come back at once and which stop to work something out. Settle the lot and two of the twenty get an answer that was prepared for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

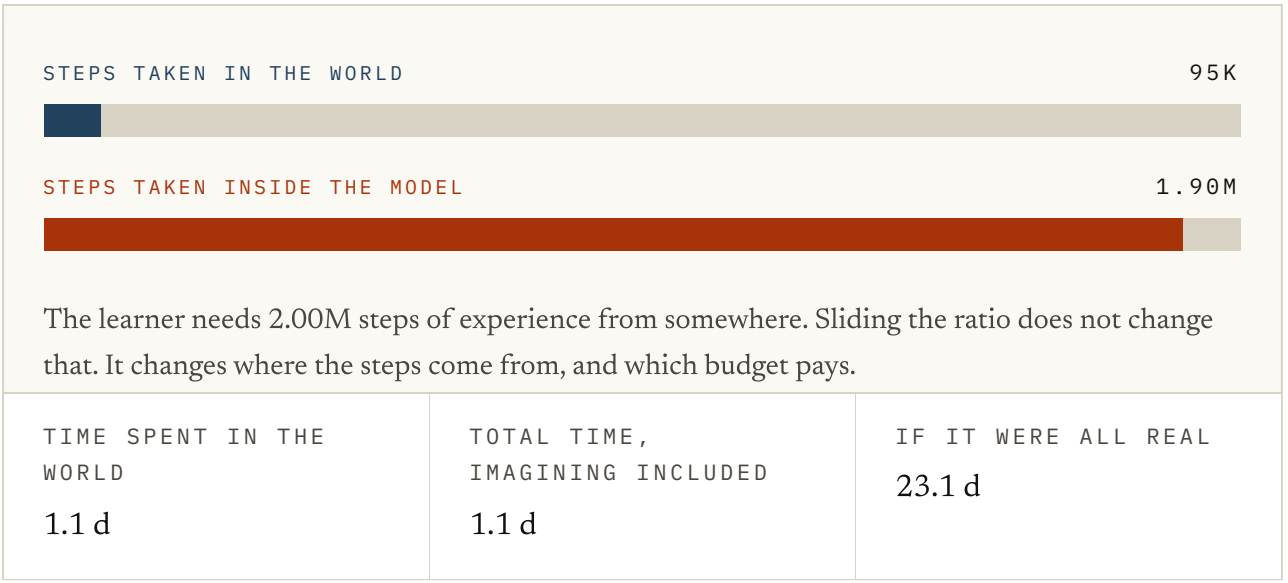
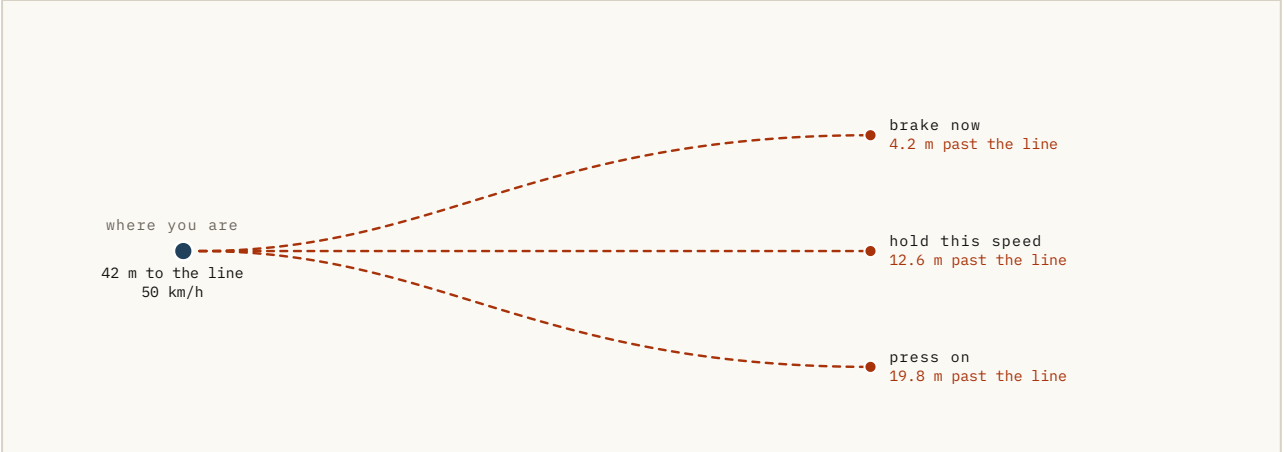


FIG. 2.11 Hold the update steps fixed and slide where they come from. World contact is usually the budget that runs out first, and compute is still a budget. Watch both bills move as you drag. This ledger prices a real step and an imagined one the same, which is the assumption chapter 6 takes apart.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



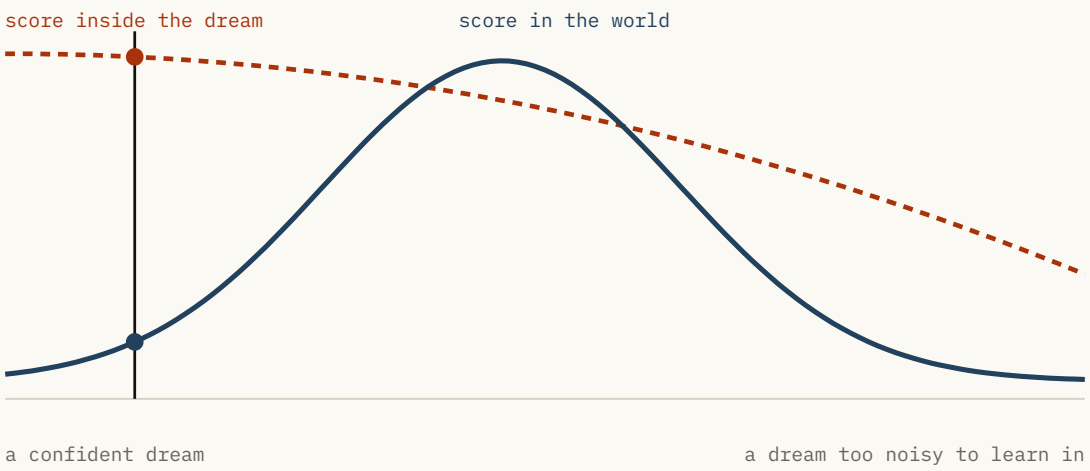
CHECKED	NEVER CHECKED	THE ONE YOU TOOK WAS OFF BY	SITUATIONS SEEN
0	3	not yet	1

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.

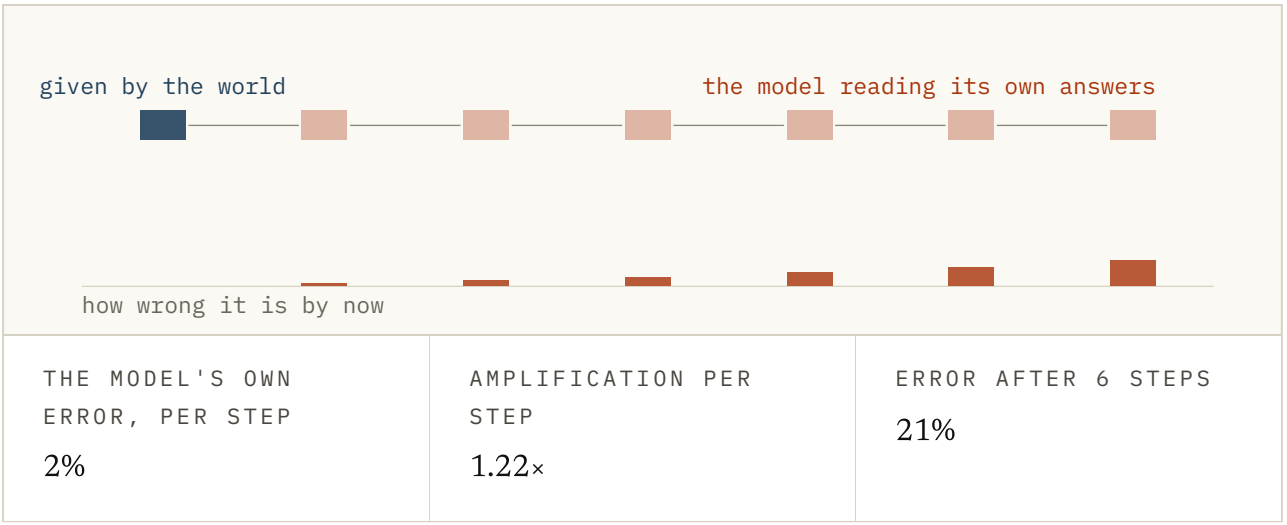
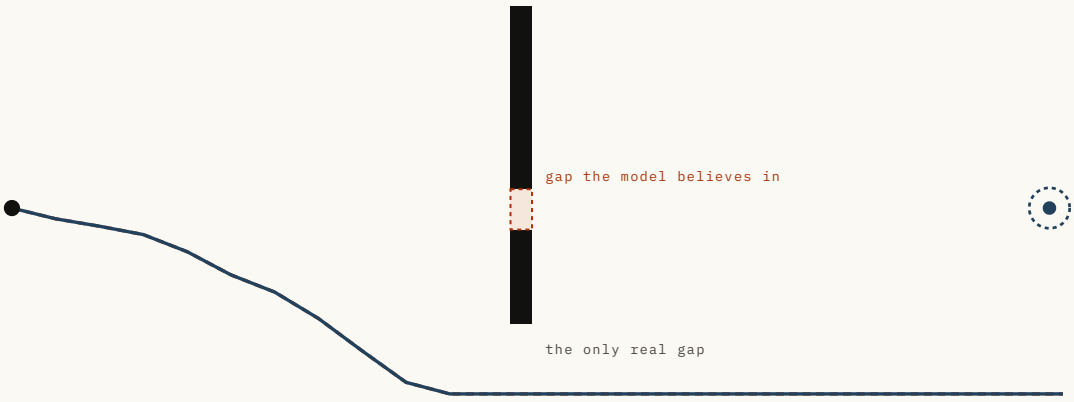


FIG. 2.14 Two things happen at every step. The model adds two per cent of its own error, and whatever was already wrong gets stretched by the dynamics it is pushed through. Pull the amplification down to 1.00 and the second effect switches off: the error just adds, and the bars go straight. Leave it 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.



IN THE MODEL	IN REALITY
Reached the goal	Reached the goal

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.

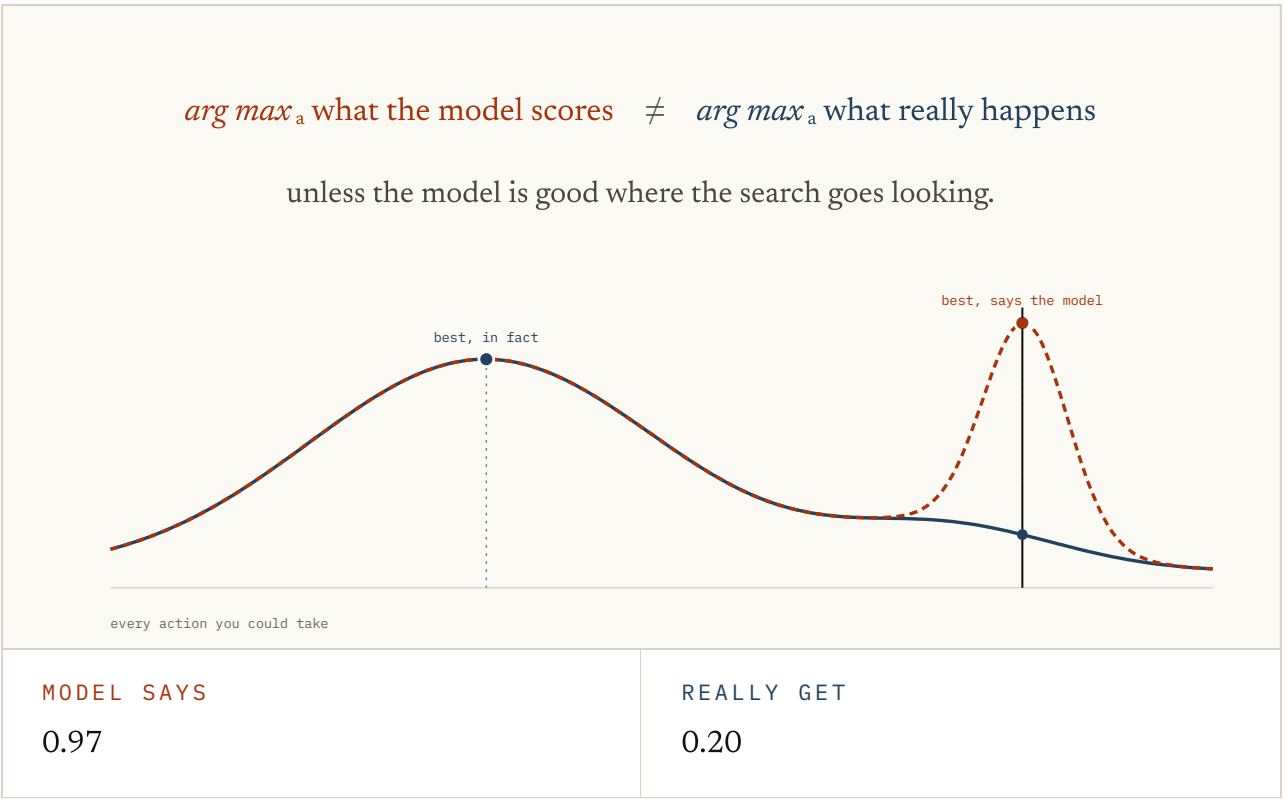


FIG. 2.16 Two ways of scoring the same actions. They agree nearly everywhere, which is what a low average error looks like. Drag across to the spike and read both numbers: the model's favourite action scores 0.97 in imagination and 0.20 in the world.

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 3	YEARS APART 1	JOBS FOR ONE PHRASE 3	SELECTED 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.

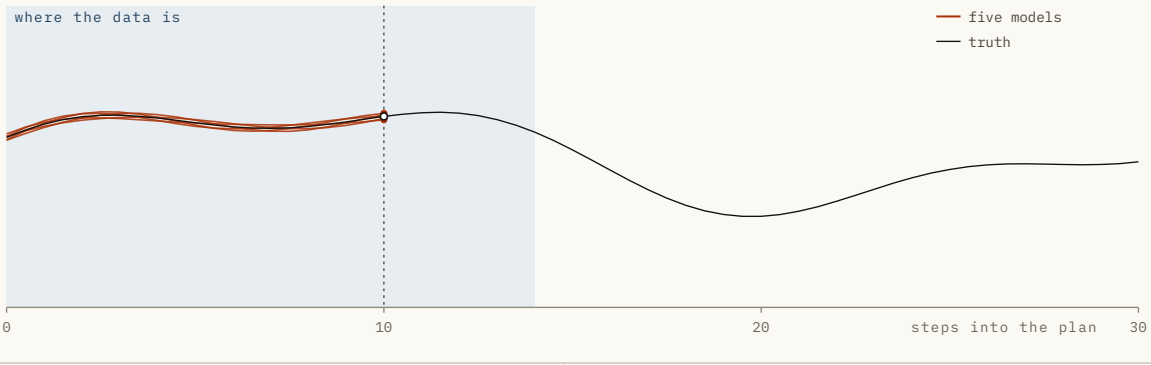
In 2018 Ha and Schmidhuber wrote the paper that made the label fashionable. Its title was simply *World Models*. It went out as an interactive web page, demos running in the browser, and built the 1990 object from newer parts: one network that sees, one that predicts, one that chooses. They called the 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

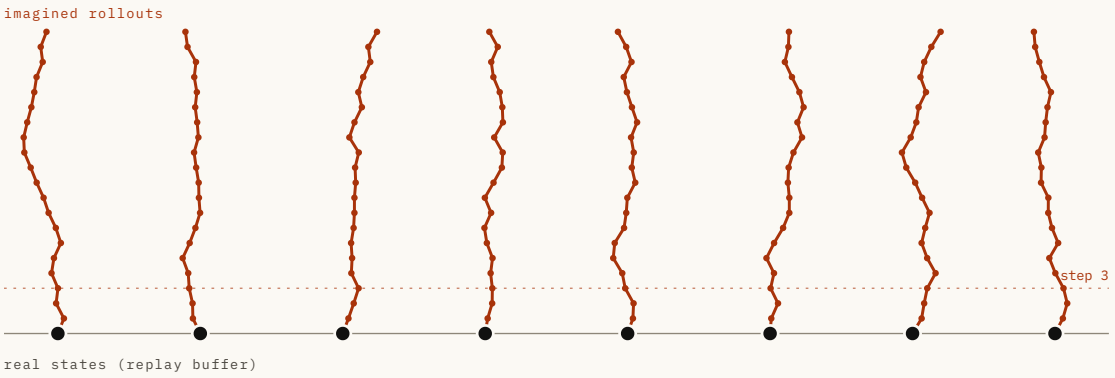
ERROR AT THE MARKER
0.0

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.



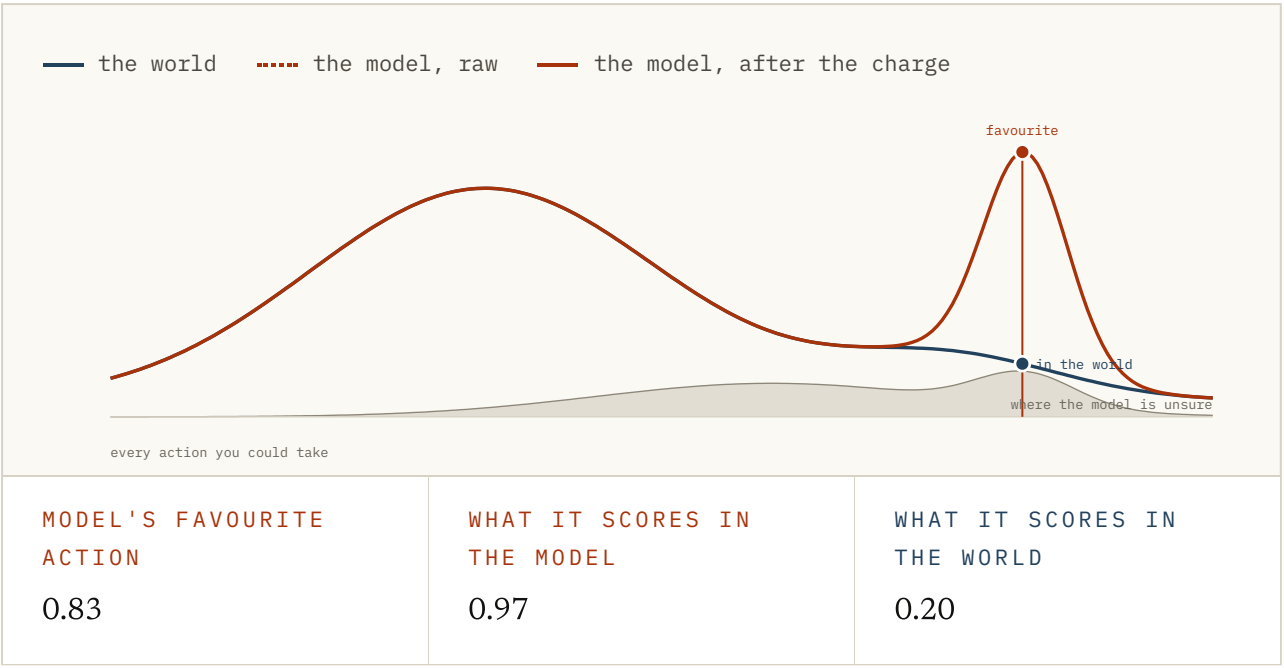
IMAGINED STEPS IN THE BATCH	STEPS NEAR EVIDENCE	FURTHEST STEP FROM A REAL STATE
24	24	3 steps

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.