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.

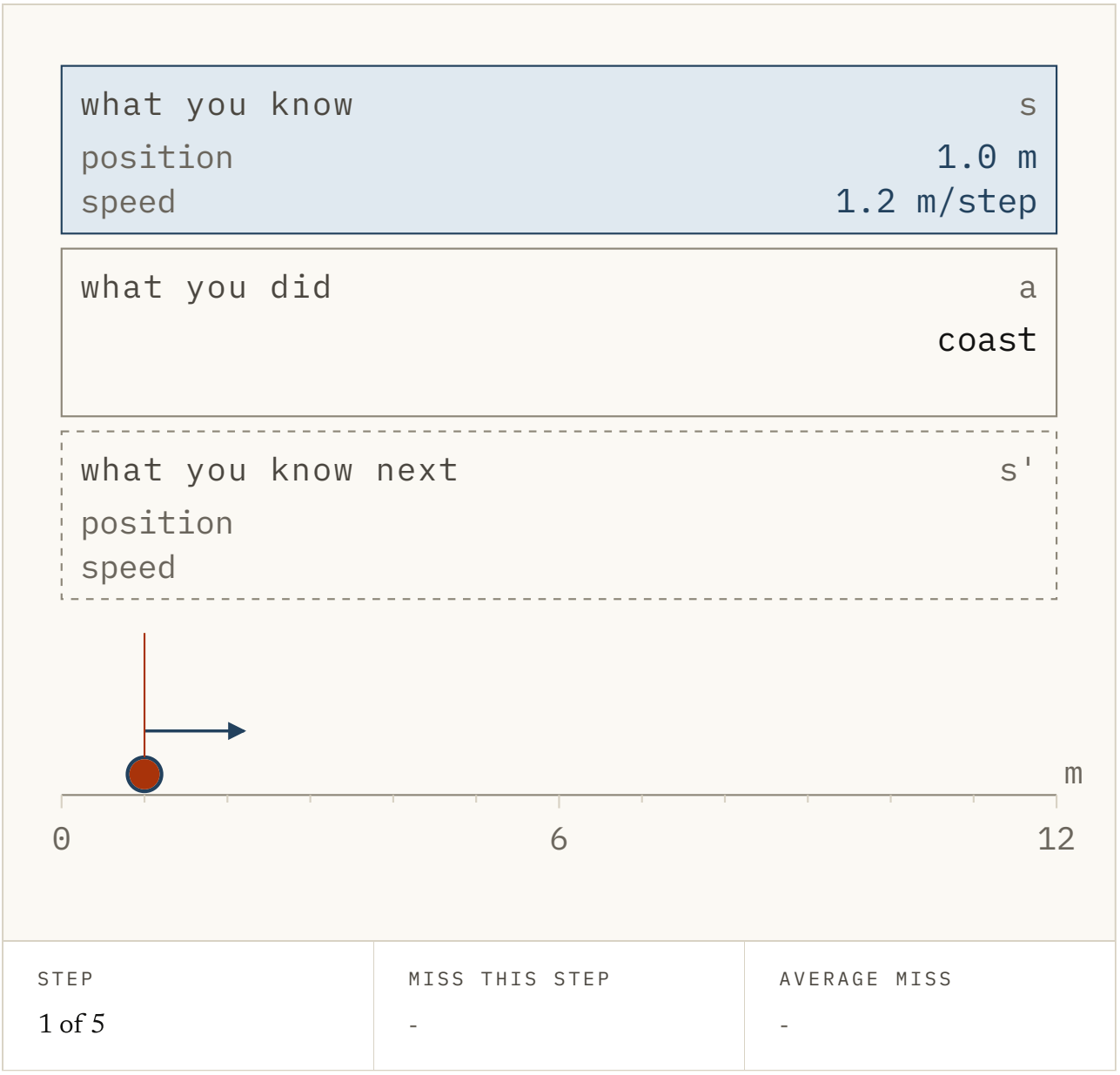
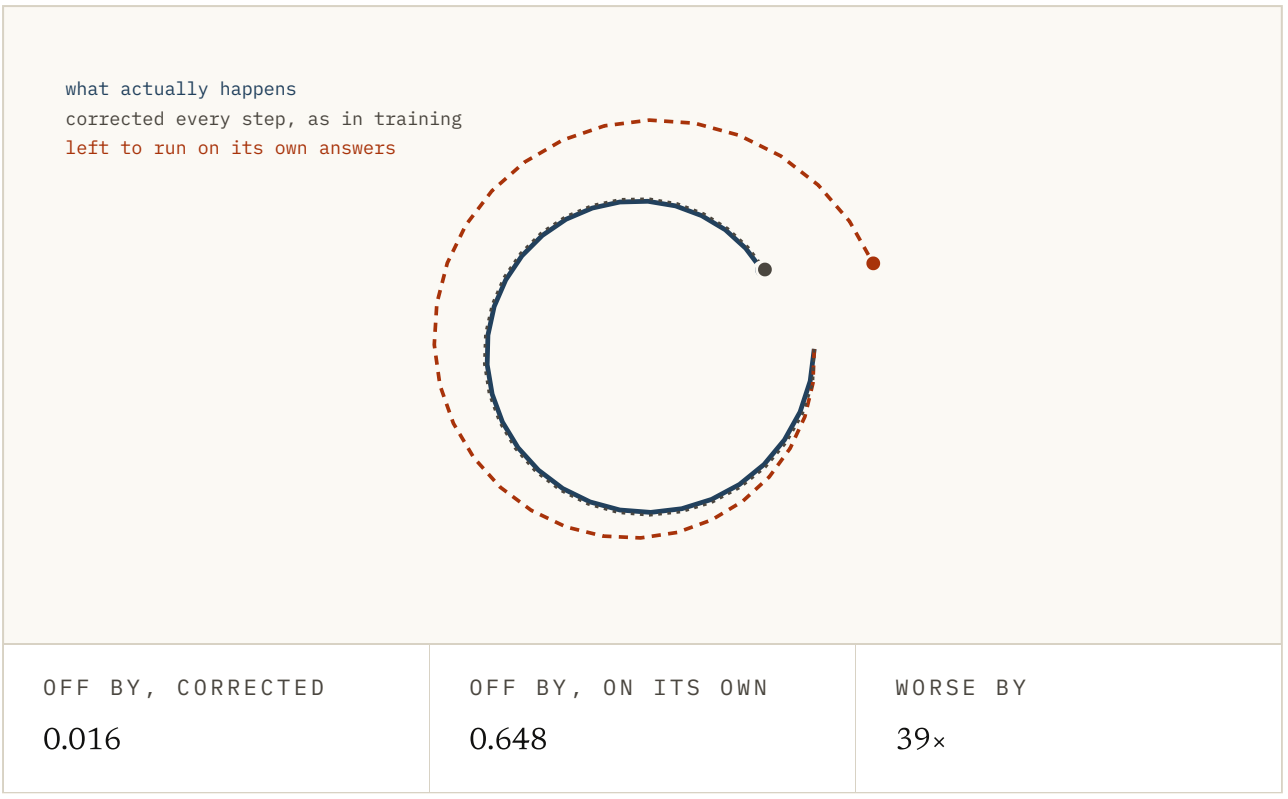


FIG. 5.1 This is the one job, and for a moment you are the model. You are handed the state and the action, so drag your answer to where the ball will be after one step and press Check. Do that a few times and watch the average miss build up underneath, because that average is the number a dynamics paper 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.

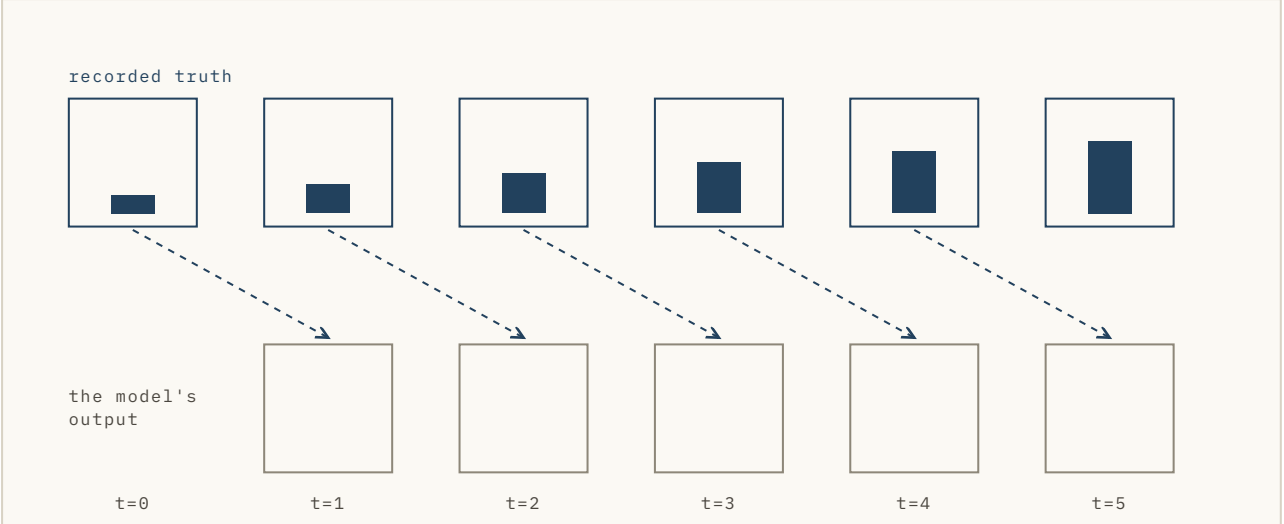


FIG. 5.3 The same five steps, drawn with the arrows that feed each one. Leave the switch on training and every step is handed the recorded truth, so a mistake goes nowhere. Flip it to in use and each step is handed the one before it, so the first small mistake rides along into every later step. Press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

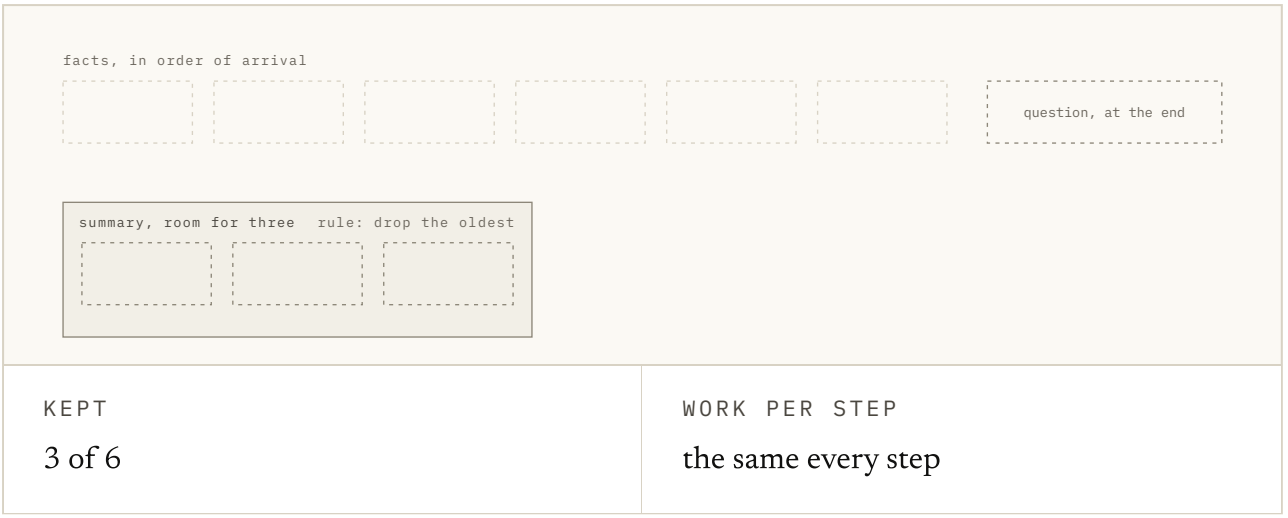


FIG. 5.5 Six facts arrive one at a time, and a summary with room for three has to keep or drop each one as it comes. A question arrives at the end. Pick the question, press Play, and watch the summary: it cannot see the question coming, so sometimes it dropped the one fact that was needed. Switch to 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.

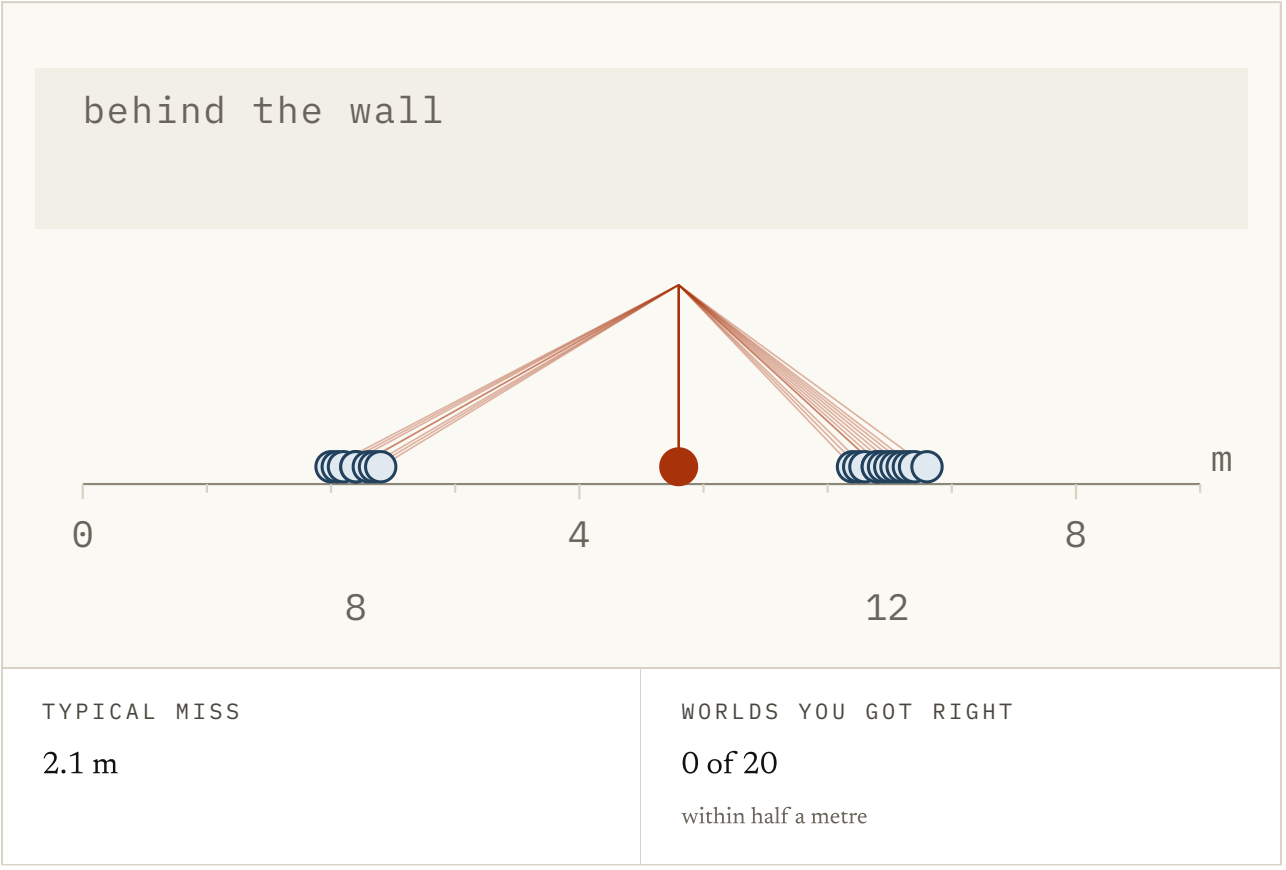


FIG. 5.6 The ball went behind the wall, and it either stopped at the kerb or rolled straight on. Drag the single answer along the ground and watch the two readouts pull apart: the smallest typical miss sits between the two clumps, in a spot where the ball never actually is. Then switch to handing back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

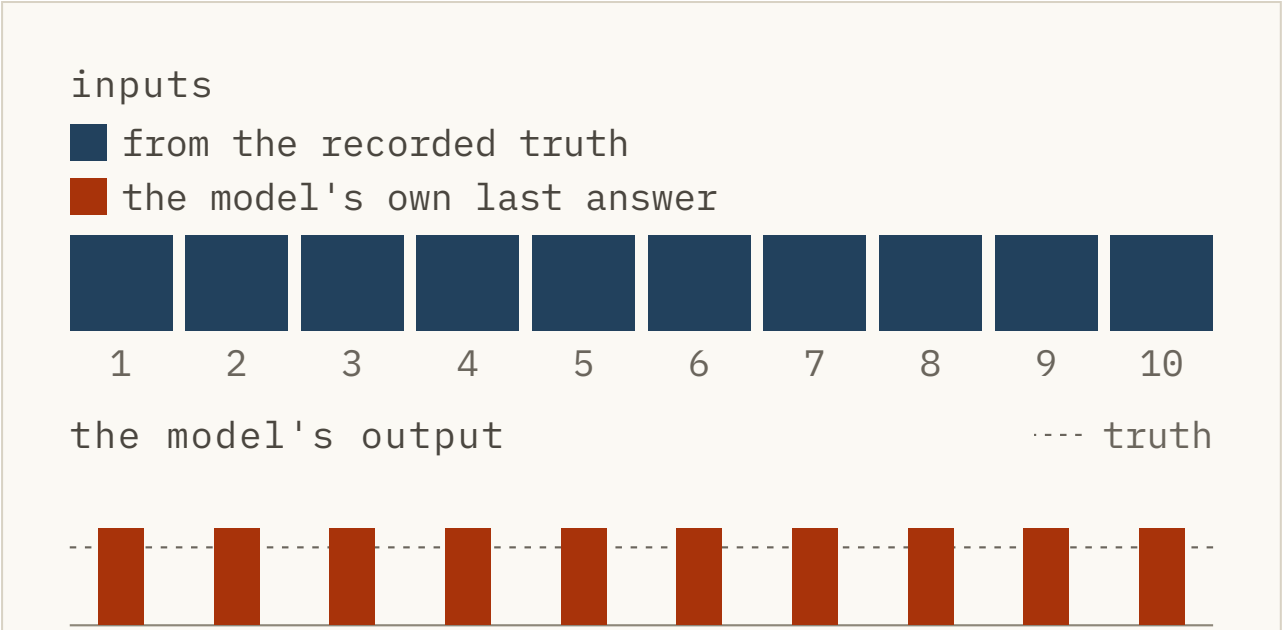
PlaNet ran the comparison and argued for carrying both, because each one fails alone in a different direction. Dreamer followed a year later, when the same group published *Dream to Control* and let an agent learn to behave inside the model's imagined rollouts.

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.



INPUTS FROM THE TRUTH	INPUTS FROM THE MODEL	WHAT THE MODEL HAS PRACTISED
10 of 10	0 of 10	never recovering

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*.

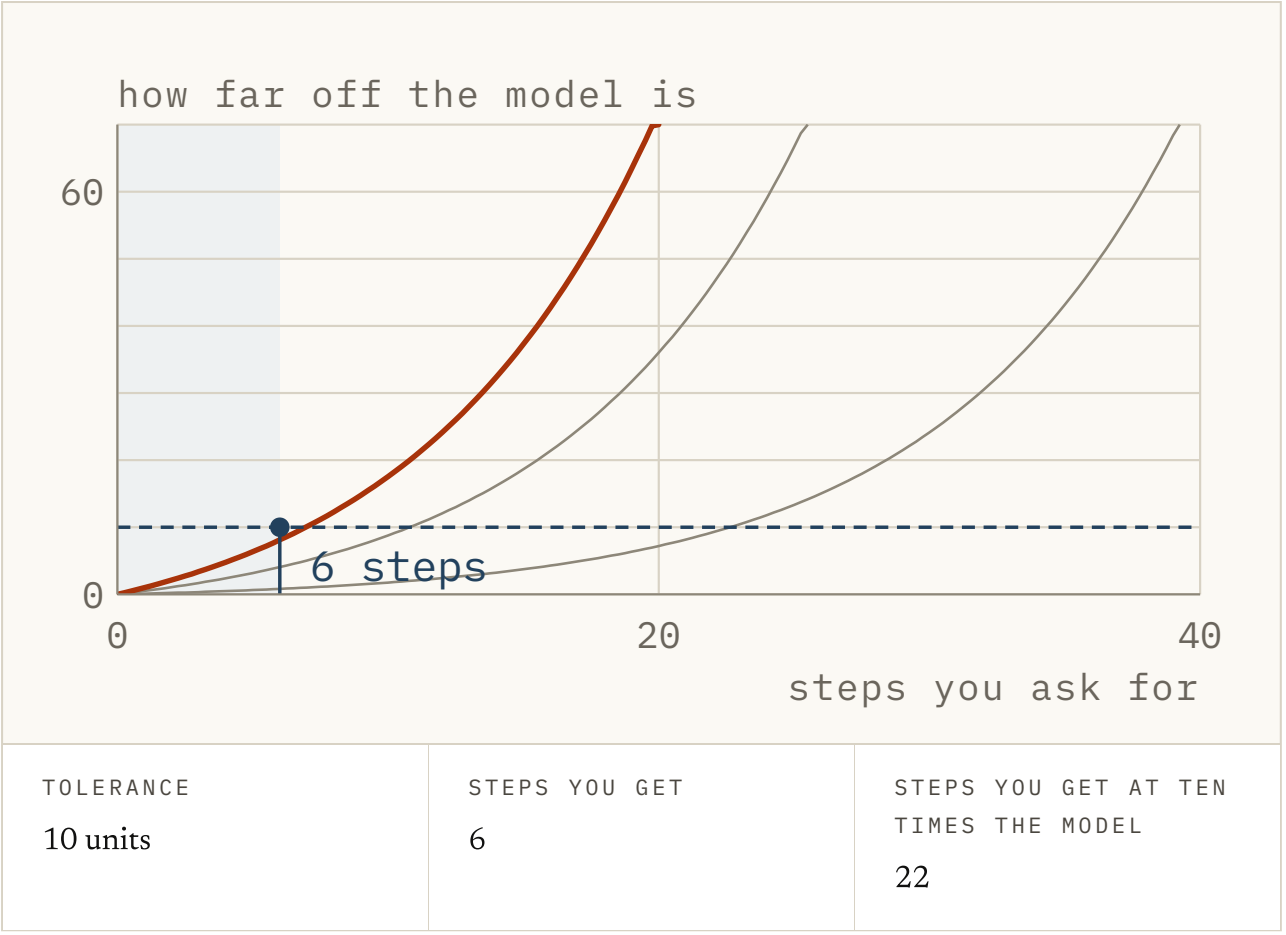


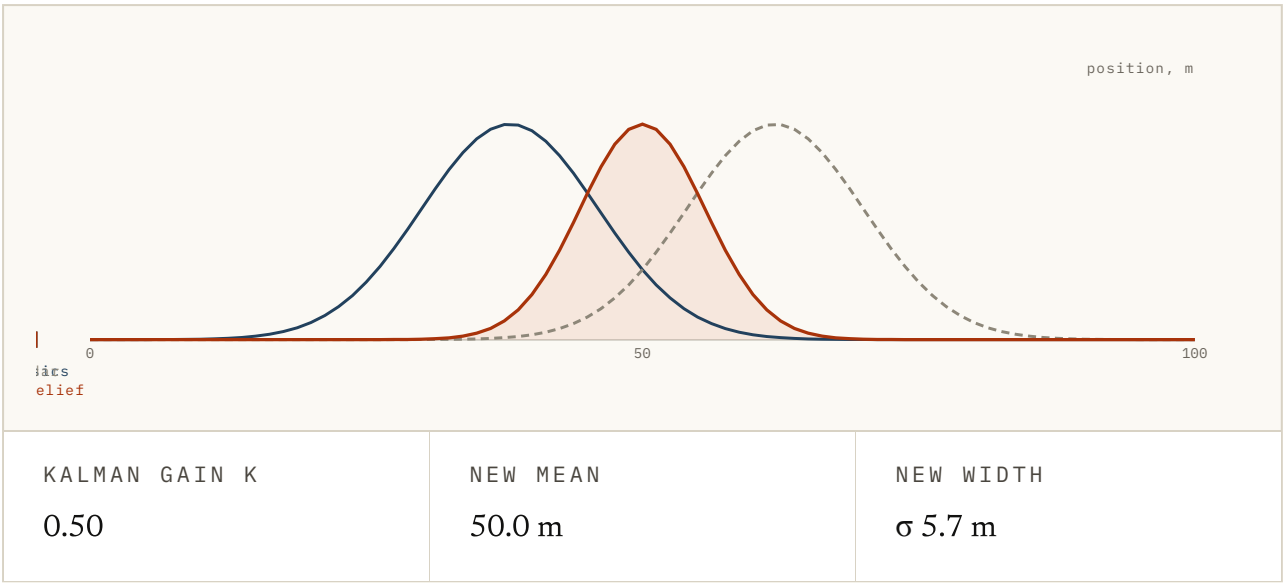
FIG. 5.8 How far you can go is a number you measure, so measure it here. Drag how wrong you are willing to be, and read off how many steps you get before the model's error crosses that line. Then swap in a model ten times better and watch the horizon move by a handful of steps rather than by a factor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



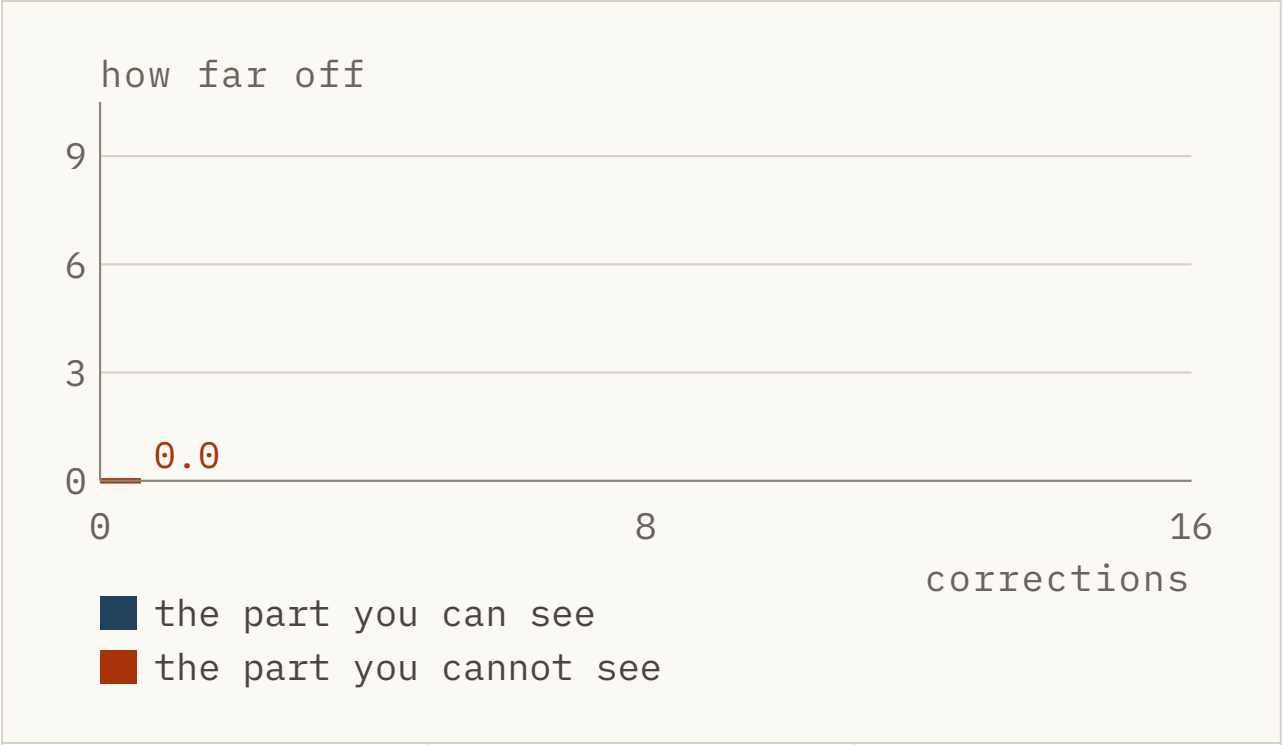
KALMAN GAIN K	NEW MEAN	NEW WIDTH
0.50	50.0 m	σ 5.7 m

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

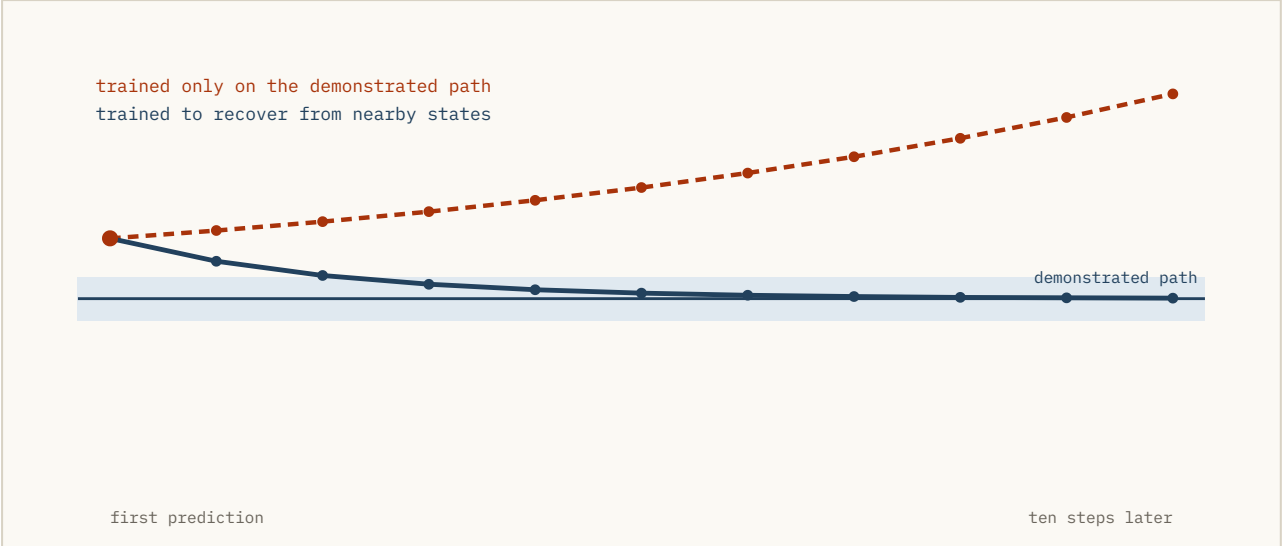
FIG. 5.10 The same blend, repeated, with the two halves of the state kept apart. Press Look again and watch the sawtooth: every prediction pushes both errors up, and every blip pulls the measured half straight back down. Now add a wind the filter was never told about and keep pressing. The measured half 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.



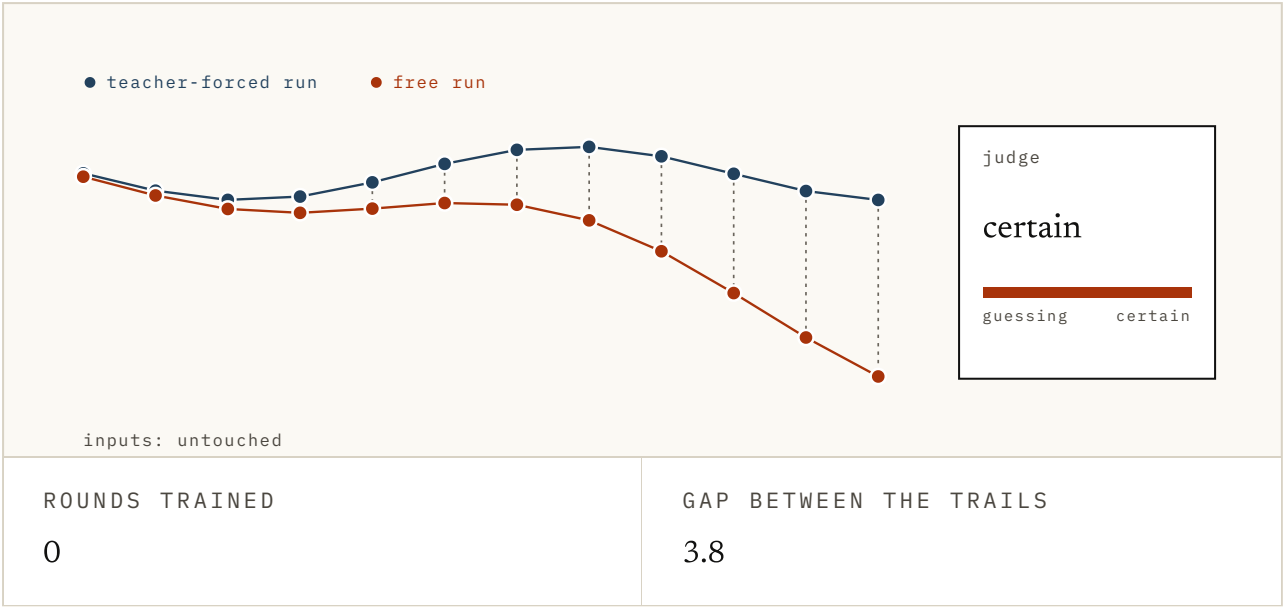
ON THE DEMONSTRATED PATH	TRUTH-ONLY ERROR AT STEP 10	RECOVERY-TRAINED ERROR AT STEP 10
both predict perfectly	1.22	0.003

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.