

AN ILLUSTRATED GUIDE

Physics-Informed AI

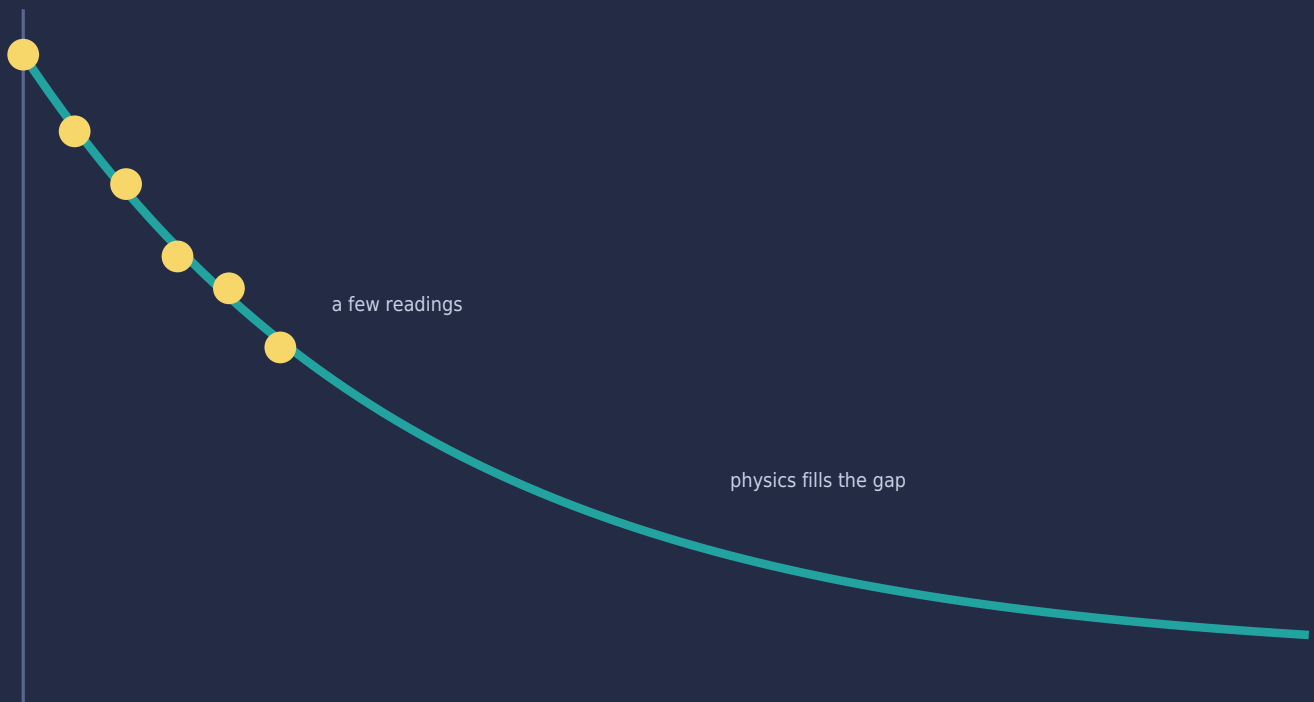
A friendly guide to AI that learns from data and obeys the laws of physics

Plain English, pictures first, a real worked example with real numbers, and a glossary, a quiz and an eight-week plan.

Nineteen short chapters · Quiz · Glossary · Eight-week plan

Prompted and directed by Mahmoud A.

Written and illustrated by Claude (Anthropic)



How to Use This Guide

Read it in order, or jump to what you need.

This guide explains **physics-informed AI**: artificial intelligence that learns from data *and* is forced to respect the laws of physics. You need no maths beyond school level and no coding. Every idea starts with an everyday picture. Equations arrive only after the picture makes sense.

Two teachers, one student

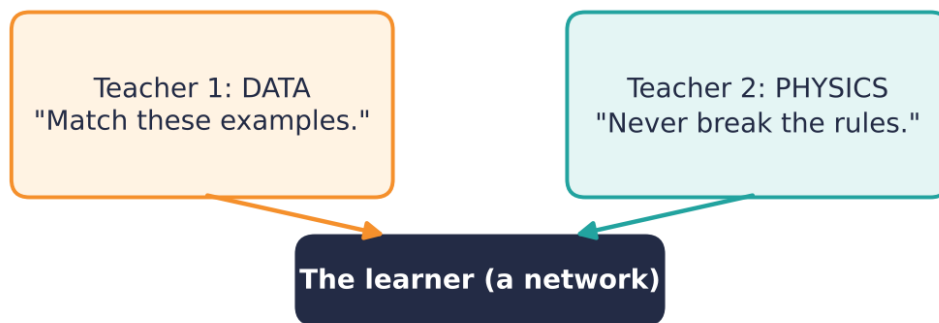


Figure 1. The whole guide in one picture: a learner with two teachers.

Who it is for

- Curious people who keep hearing words like **PINN** or **physics-informed** and want to know what they mean.
- Architects, engineers, students and managers who work with sensors, simulations or digital twins.
- Anyone who needs to judge whether the idea fits a real problem, and when it does not.

How each chapter works

- A **banner** tells you the topic in a sentence.
- Plain explanation first, then a picture, then a small example.
- Boxes: **TIP** (a shortcut), **SAY IT LIKE THIS** (a phrase you can use out loud), **WATCH OUT** (a common trap) and **IN ONE LINE** (the chapter in a sentence).
- At the end: a quiz with answers, a glossary, a one-page cheat sheet, further reading and an eight-week plan.

A note on the numbers

The worked example in Chapter 11 is **real**: the networks were trained for this guide and every number and curve you see comes from that run. The coffee readings are **simulated** (a known law plus a little noise), so that we can check the answers against the truth. Real sensors would be messier. Chapter 18 says what that changes.

Contents

Nineteen short chapters and a toolbox at the back.

1 What Is Physics-Informed AI?	4
2 Machine Learning in Five Minutes	5
3 Neural Networks Without the Fog	7
4 Why Data-Only AI Breaks	8
5 Physics: Rules Written as Equations	9
6 Rates of Change: The One Idea You Need	10
7 The Core Trick: Two Teachers, One Score	11
8 How the Network Checks the Physics	13
9 Checkpoints: Testing Physics Where There Is No Data	15
10 Check the Physics by Hand	16
11 Worked Example: The Cooling Coffee	18
12 Watching the Network Learn	20
13 Turning the Dials	22
14 Running It Backwards: Discovering Unknown Physics	24
15 From One Number to a Whole Field: PDEs	25
16 Other Ways to Put Physics into AI	27
17 Where It Is Used	29
18 Pitfalls and Honest Limits	30
19 Try It Yourself	32
Six Myths and the Facts	34
Quick Quiz	35
Quiz Answers	36
Glossary	37
One-Page Cheat Sheet	38
Further Reading	39



What Is Physics-Informed AI?

The big idea in plain words.

Ordinary AI is a very good **copycat**. Show it many examples and it learns to produce answers that look like them. That works wonderfully when examples are plentiful: photos of cats, millions of sentences, years of shop sales.

Many real problems in engineering and science are not like that. A building has a few dozen sensors, not millions. A bridge is measured a handful of times a year. A cup of coffee gives you six readings before you lose patience. With so few examples, a copycat has no idea what to do in the gaps.

Yet we are not helpless. For these problems we usually know **rules that are always true**: hot things cool toward their surroundings, energy is not created from nothing, water does not flow uphill by itself. Physicists wrote these rules down as equations long ago.

The idea in four sentences

1. A normal AI learns only from examples, so it guesses wildly where it has none.
2. Physics gives us rules that hold everywhere, even where we have no examples.
3. We add a penalty that grows whenever the AI's answer breaks those rules.
4. The AI now has to satisfy the data and the rules at once, so it fills the gaps sensibly.

A schoolroom picture

Imagine two students preparing for an exam. The first memorises last year's answers. The second also learns the underlying laws. Give both a question they have never seen. The first can only hope it resembles something memorised. The second can *reason*. Physics-informed AI is the second student.

SAY IT LIKE THIS

"A physics-informed model is trained on the data *and* penalised whenever it breaks a known law, so it stays sensible between and beyond the measurements."

Three names for the same family

Name	What it stresses
Physics-informed neural network (PINN)	A neural network whose training includes the equation as a penalty.
Physics-informed machine learning	The wider family: any way of giving a learning method physical knowledge.
Scientific machine learning (SciML)	The field that mixes simulation, equations and learning.

IN ONE LINE

Physics-informed AI = learning from data + a penalty for breaking known physical laws.

2

Machine Learning in Five Minutes

Finding a pattern from examples, nothing more.

Machine learning lets a computer find a pattern from examples instead of being told the rule. Think of a child learning what a dog is. Nobody hands over a definition. The child sees many dogs, gets corrected, and the pattern settles in. A model learns the same way, only with numbers.

The simplest example

Record how many hours each student studied and what they scored. Each student is one dot. The model's job is to draw the line that sits closest to all the dots.

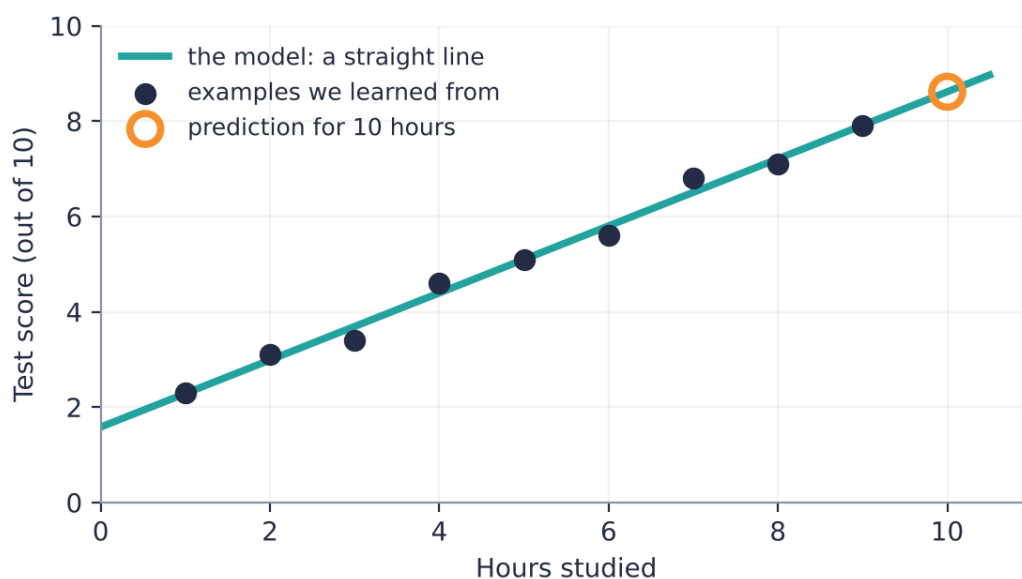


Figure 2. Learning means finding the line that sits closest to the dots (invented data, nine students).

Once the line exists you can ask a new question: what score after ten hours? You read the answer off the line. That answer is a **prediction**.

Three words you will meet again and again

- **Data:** the dots. The examples we learned from.
- **Model:** the thing that makes predictions. Here it is a straight line. In deep learning it is a neural network, a far more flexible curve.
- **Loss:** one number saying how wrong the model is. Here it is the total of the dotted gaps between each dot and the line. Lower is better.

Learning is turning knobs

A straight line has two knobs: its slope and its starting height. Training means trying values, measuring the loss, nudging the knobs so the loss falls, and repeating. The nudging method is called **gradient descent**.

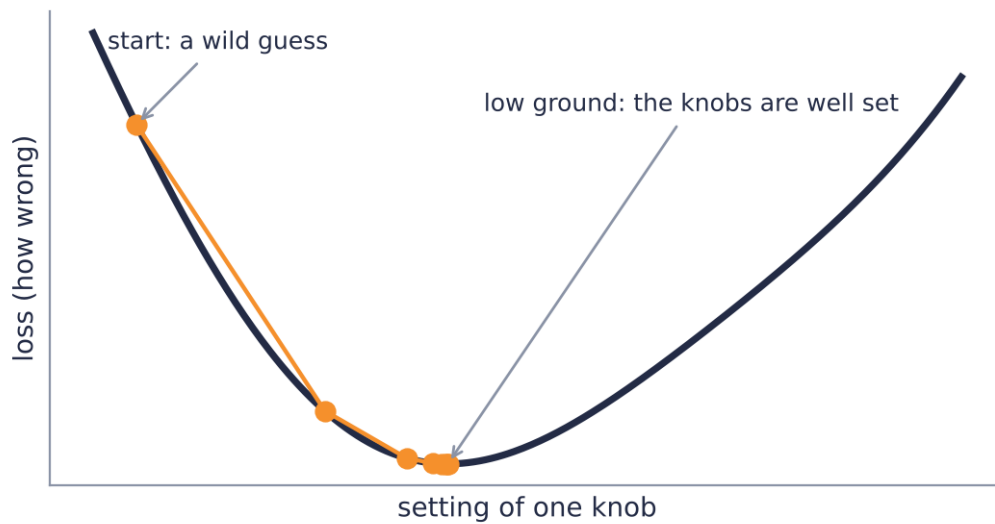


Figure 3. Gradient descent: always step downhill on the loss landscape until you reach low ground.

Picture yourself on a foggy hillside, unable to see the valley. You feel which way the ground slopes under your feet and step downhill. Repeat enough times and you reach low ground. The height of the ground is the loss. Your position is the setting of the knobs.

TIP

Whenever you hear "training", think "turning knobs to push one number, the loss, down". Everything in this guide is a variation on that sentence.

IN ONE LINE

A model is a shape with knobs; training adjusts the knobs until the loss, a single wrongness score, is small.

3

Neural Networks Without the Fog

A stack of small calculators with a lot of knobs.

A neural network is a stack of small adjustable calculators arranged in layers. Numbers go in at one end, pass through layer after layer of simple sums and bends, and an answer comes out at the other end.

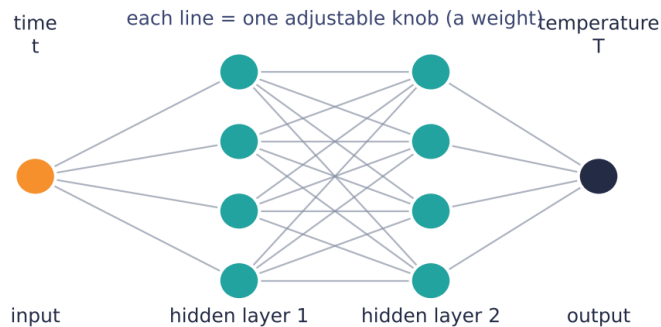


Figure 4. A tiny network: one input (time), two hidden layers, one output (temperature). Every line is a knob.

What one calculator does

Each circle takes the numbers coming in, multiplies each by its own knob (called a **weight**), adds them up, adds one more knob (a **bias**), and then *bends* the result with a fixed function such as $\tanh$. The bend is what lets the network draw curves rather than only straight lines.

Why it can draw almost any curve

One bend gives a gentle S-shape. Many bends, added together with different knob settings, can build wiggles, hills and plateaus. With enough circles, a network can approximate almost any smooth curve. Mathematicians call this the **universal approximation** property.

The catch

Flexibility cuts both ways. A network flexible enough to match any dots is also flexible enough to do *anything at all* between and beyond them. The data pins the curve down only at the dots. The next chapter shows what happens everywhere else.

WATCH OUT

"Deep" learning only means many layers. It does not mean the network understands anything. It means it has many knobs and a lot of freedom.

Layers, weights, biases and activations are all just names for the pieces above. Together, all the knobs are called the **parameters**; large networks have millions of them.

IN ONE LINE

A neural network is a very flexible curve with many knobs; the data decides the knobs only where there is data.

4

Why Data-Only AI Breaks

Great at the dots, helpless in the gaps.

Take the coffee cup. You pour 90 °C coffee into a 20 °C room and read the temperature every two minutes for ten minutes. Six readings, each a little noisy. Now ask: what will it read after an hour?

We tried two flexible models that match the six readings very well. Both look excellent **where the data is**. Watch what happens afterwards.

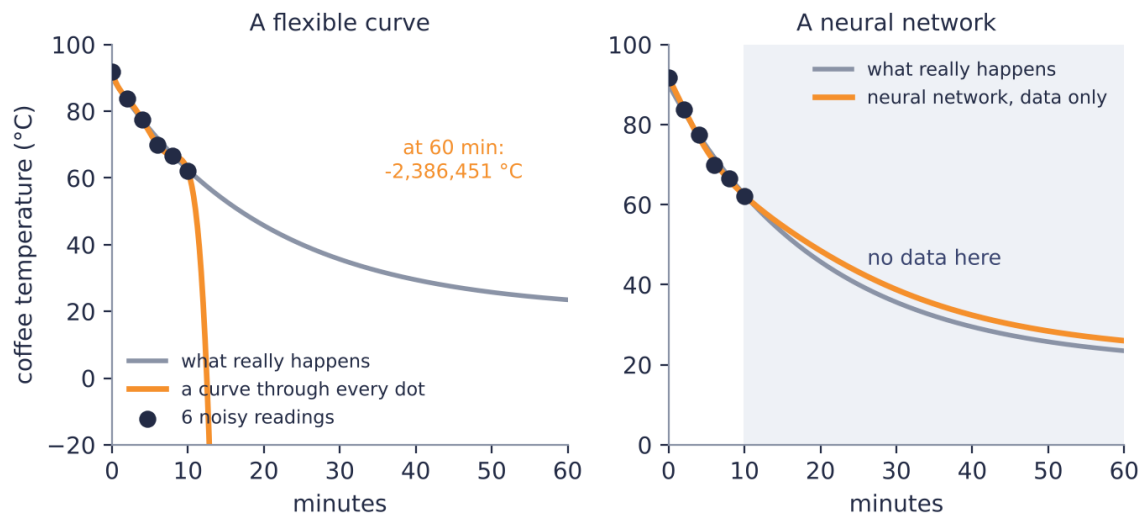


Figure 5. Left: a flexible curve through every dot. Right: a neural network trained on the same dots. Both real results.

The curve on the left threads through all six readings, then plunges. Its prediction for minute 60 is **-2,386,451 °C**: colder than outer space, from a cup of coffee. The neural network on the right behaves more politely, yet it still drifts away from the truth: at minute 60 it says 26.0 °C, while the real value is 23.5 °C. Its typical miss over the unseen minutes is 2.7 °C.

Why does this happen?

- The data fixes the curve at six points. Infinitely many curves fit those points equally well.
- The model has no reason to prefer the sensible ones. It picks whichever its training procedure lands on.
- Noise makes it worse: the model may bend itself to match random error, a problem called **overfitting**.

More data would fix it, so why not collect more?

Sometimes you can. Often you cannot: sensors are expensive, experiments destroy things, buildings cannot be rebuilt ten times, and the future has not happened yet. In those situations the missing ingredient is not more dots. It is **knowledge** about how the world behaves.

IN ONE LINE

With few examples, a flexible model is free to be nonsense between them; we need a second source of guidance.

A **law of physics** is a rule that has held in every careful test. It does not care which cup of coffee, which building or which day. That is exactly the kind of knowledge a few dots cannot provide.

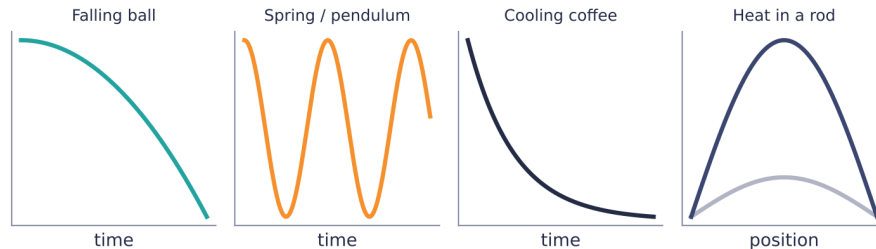


Figure 6. Four everyday systems, each governed by a simple rule about how things change.

Rules about change

Most physical laws are statements about *how fast something changes*. A ball speeds up as it falls. A hot drink cools quickly at first, then slowly. Heat spreads along a metal rod. The rule rarely tells you the answer directly. It tells you how the answer must **move**.

System	The rule in words	The equation
Cooling coffee	It cools faster the hotter it is compared with the room.	$dT/dt = -k (T - T_{\text{room}})$
Falling ball	Speed grows steadily because gravity pulls with a constant strength.	$d^2h/dt^2 = -g$
Spring or pendulum	The pull back is proportional to how far you moved.	$d^2x/dt^2 = -(k/m) x$
Heat in a rod	Heat flows from hot to cold, smoothing out differences.	$\partial T/\partial t = \alpha \partial^2 T/\partial x^2$

Do not worry about the symbols yet: Chapter 6 explains **dT/dt** .

Equations versus simulators

Engineers usually solve these equations with **simulation software** (finite elements, CFD). Simulators are accurate when you know everything: the geometry, the materials, the starting state. In real life you often do not. Physics-informed AI sits between pure simulation and pure data.

Rules that must hold

- **Conservation:** energy, mass and momentum cannot appear or vanish.
- **Symmetry:** physics works the same today and tomorrow, here and there.
- **Boundary conditions:** what is happening at the edges (a hot wall, a cold window).
- **Initial conditions:** where things start (coffee at 90 °C).

IN ONE LINE

Physics hands us rules about how things must change, true everywhere, which is exactly what sparse data lacks.

6

Rates of Change: The One Idea You Need

What dT/dt really means.

There is exactly one piece of calculus you need for this guide, and you already use it when you drive. Your speed is how fast your position is changing. The needle on the speedometer is a **rate of change**.

Slope = rate of change

On a graph, the rate of change at a point is the **slope** of the curve there: how steeply it rises or falls. Mathematicians write the slope of temperature against time as dT/dt . Read it aloud as "how fast T changes with t ".

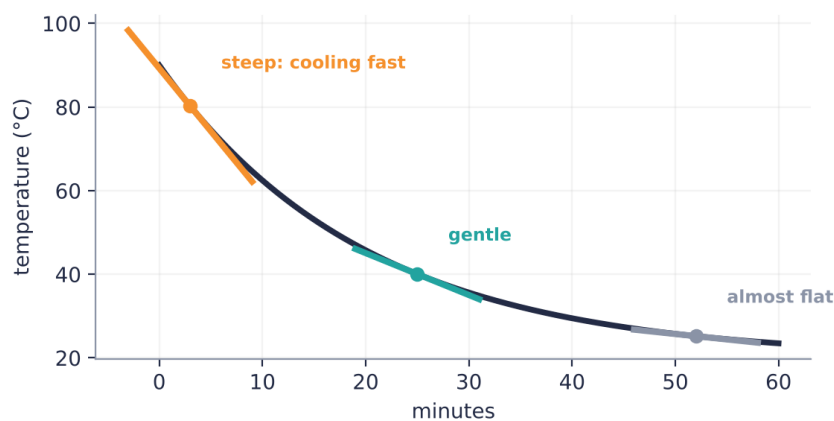


Figure 7. The coffee's cooling curve with the slope drawn at three moments: steep at first, nearly flat later.

Reading the coffee equation

Here is the physical rule again: $dT/dt = -k(T - T_{\text{room}})$.

- $T - T_{\text{room}}$ is how much hotter the coffee is than the room.
- k is a number saying how quickly this cup loses heat (a bigger k means a faster cooler).
- The minus sign says the temperature is falling.
- So: *the hotter it is compared with the room, the faster it cools*. Exactly what the picture shows.

The key trick: checking a rule instead of solving it

To **solve** the equation you need calculus. To **check** whether a proposed answer obeys the rule, you only need to measure its slope and its temperature at a few moments and see whether the two sides agree. Checking is easy even for a computer that has no idea how to solve anything. This is the doorway to the whole method.

TIP

If you can compute a slope with a ruler on a graph, you can understand physics-informed AI. The computer does the same with numbers.

IN ONE LINE

dT/dt is the slope of temperature against time; a physical law ties that slope to the current state.

7

The Core Trick: Two Teachers, One Score

Adding a physics penalty to the loss.

Remember the **loss** from Chapter 2: one number saying how wrong the model is. A normal network has only one teacher, the data, and one loss, the **data loss**: the gap between the model's answers and the measurements.

Physics-informed learning adds a second teacher, and with it a second loss.

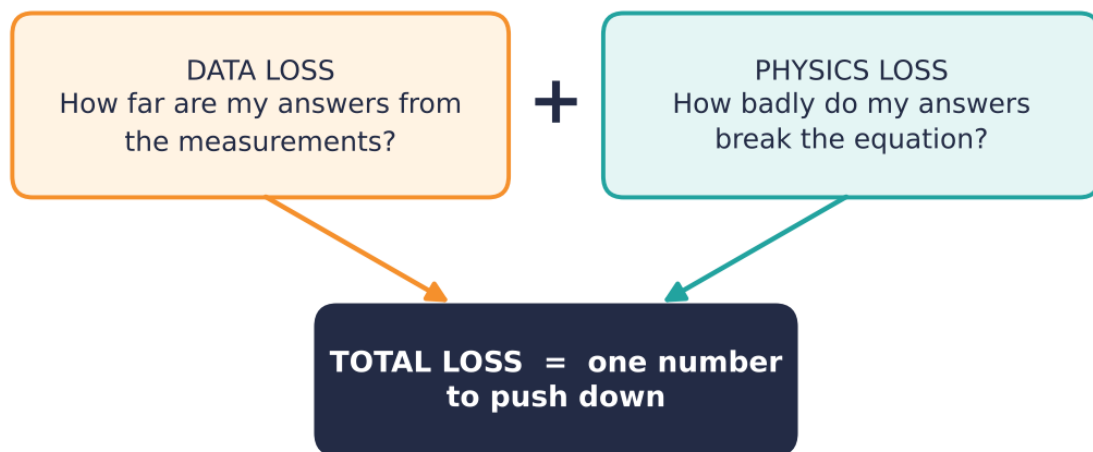


Figure 8. The physics-informed score is the sum of two penalties.

The data loss

"At the times we measured, how far are my answers from the readings?" This keeps the model honest about reality.

The physics loss

"At many other times, does my answer follow the rule?" Rewrite the equation so that everything is on one side. For the coffee: **residual** = $dT/dt + k(T - T_{\text{room}})$. If the model obeys physics, the residual is zero. The physics loss is how far from zero the residual is, averaged over many times. Any answer that breaks the law gets a penalty.

The total

Total loss = data loss + λ × physics loss. The Greek letter λ (lambda) is a weight that sets how much each teacher counts. Training turns the knobs to push the *total* down, so the network must satisfy both teachers at once.

Too little physics: ignores the law. Too much: ignores the data.



Figure 9. Balancing the two teachers. The weight λ is a dial, and getting it right takes some care.

WATCH OUT

Balancing the two losses is one of the hardest practical parts. If the physics loss dominates, the network may satisfy the equation with an answer that ignores your measurements. If the data loss dominates, you are back to a plain network.

SAY IT LIKE THIS

"We penalise the network for disagreeing with the data, and also for disagreeing with the equation."

IN ONE LINE

Total score = how far from the data + how far from the physics; training pushes both down together.

8

How the Network Checks the Physics

Automatic differentiation, the quiet hero.

The physics loss needs the **slope** of the network's answer. The network is just a big pile of arithmetic, so how do we find its slope with respect to time?

Option 1: measure it numerically

Nudge the time by a tiny amount, see how the answer moves, divide. This is crude and noisy, and gets awkward when the equation needs second or third slopes.

Option 2: automatic differentiation (autodiff)

Every operation inside the network (multiply, add, tanh) has a slope we know exactly. Autodiff chains those little slopes together using the **chain rule**: the slope of a chain of steps is the product of the slopes of each step.

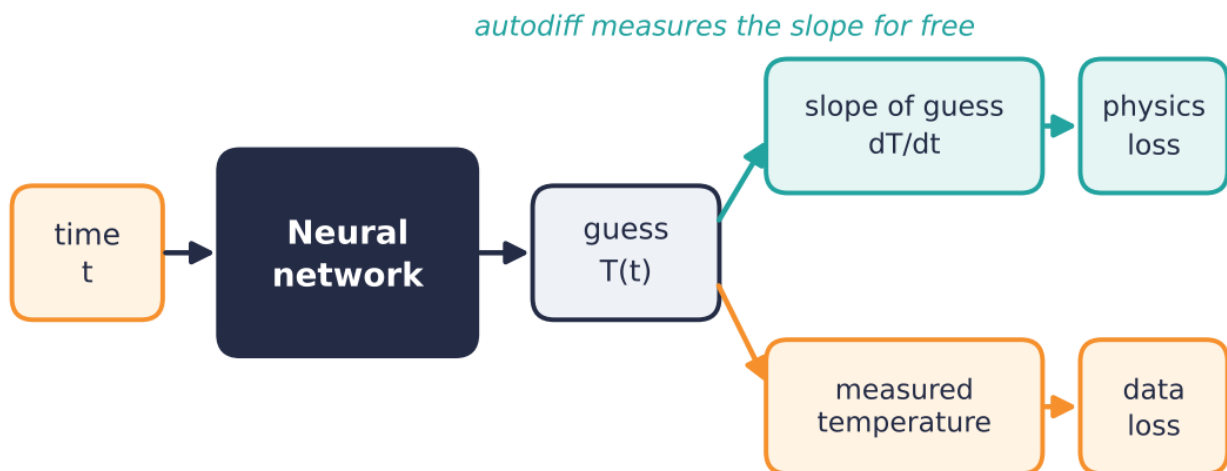


Slope of T with respect to $t = 2 \times 2a \times 1 = 8t$ (chain rule: multiply the local slopes)

Figure 10. The chain rule on a four-step calculation. Multiply the local slopes to get the overall slope.

Deep-learning software already does this to find how to turn the knobs, so it comes with the tool built in. It can differentiate the network *with respect to its input* just as easily. That gives us the slope **exactly** and for free.

The whole machine, in one picture



Both losses flow back into the knobs of the same network.

Figure 11. Inside a physics-informed network. One network, two losses, one set of knobs.

1. Feed in a time t . The network guesses a temperature $T(t)$.
2. Autodiff measures the slope dT/dt of that guess.
3. Plug both into the physics rule. If it holds, the residual is zero; otherwise it is not.
4. At the six measured times, also compare the guess with the reading.
5. Add the two losses and nudge all knobs to reduce the total. Repeat thousands of times.

TIP

Autodiff is not the same as numerical differentiation and not the same as symbolic algebra. It is exact up to computer precision and costs about as much as running the network once more.

IN ONE LINE

Autodiff lets the network's own arithmetic report its slope, so the physics rule can be checked anywhere.

9

Checkpoints: Testing Physics Where There Is No Data

Physics can be asked about any time we like.

Here is the neat part. Measurements cost effort, but **asking the physics question costs only computing time**. We can check the rule at any time, including times where nobody ever measured.

● measurements (only 6) ● physics checkpoints (as many as we like)

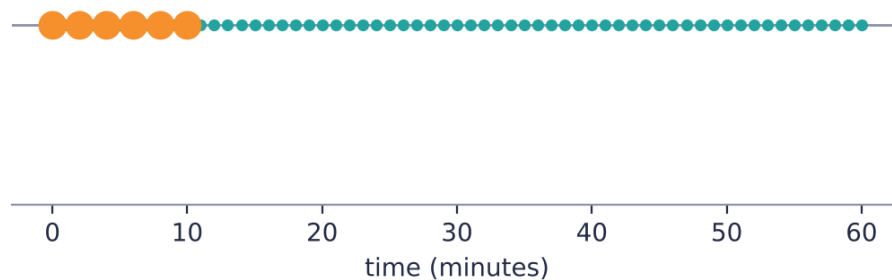


Figure 12. Six measurements (orange) and sixty-one physics checkpoints (teal) across the same hour.

These checkpoints are called **collocation points**. In the coffee example we use one every minute from 0 to 60, even though the readings stop at minute 10. At each checkpoint the network is asked: "does your answer here follow the cooling rule?"

What the checkpoints do

- They carry the knowledge of the rule into the unmeasured region.
- They are free: no sensor, no experiment, no cost beyond computation.
- They can be placed densely where the answer changes quickly.
- They are not data. The model is never told *the value* of the temperature at a checkpoint, only that its slope and value must be consistent.

How many, and where?

There is no single answer. A reasonable start is a regular grid or random points covering the whole region of interest, then more points where the residual stays large. Some methods move the checkpoints automatically towards difficult areas.

WATCH OUT

Checkpoints only *test* the rule. They cannot supply what the rule does not contain. The rule leaves the starting temperature and the cooling constant unspecified, which is why the measurements are still needed.

IN ONE LINE

Collocation points let physics speak at thousands of places where no sensor ever looked.

10

Check the Physics by Hand

The whole method fits on the back of an envelope.

Before the full example, do one physics check yourself. No computer needed. If you can do this, you understand what the network does thousands of times a second.

Step 1: what does the rule demand?

The coffee rule is $dT/dt = -k (T - T_{\text{room}})$ with $k = 0.05$ per minute and a room at 20°C . Suppose at some moment the coffee reads 60°C . The rule then demands a slope of $-0.05 \times (60 - 20) = -2.0^\circ\text{C per minute}$.

Step 2: what does the network claim?

Suppose a half-trained network says the temperature is 60°C right now but that it is falling at only $-1.0^\circ\text{C per minute}$. The two statements disagree. The mismatch is the residual:

$$\text{residual} = dT/dt + k (T - T_{\text{room}}) = -1.0 + 0.05 \times 40 = +1.0$$

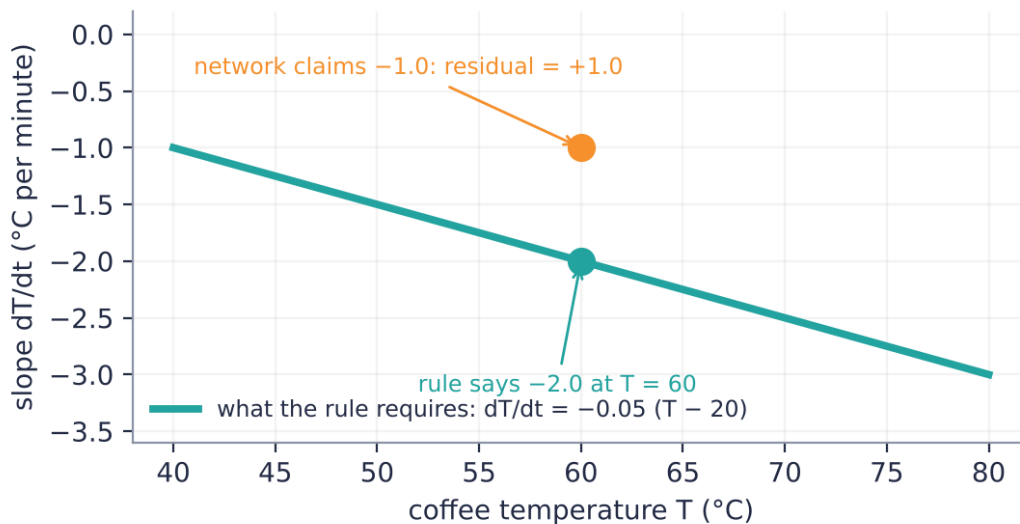


Figure 13. The rule demands a slope of -2.0 at 60°C ; the network claims -1.0 . The gap is the residual.

Step 3: turn the mismatch into a penalty

Square the residual so that positive and negative mismatches both count, then average over several checkpoints. Say the network is also wrong at two more moments, with residuals $+0.2$ and 0 :

Checkpoint	Residual	Squared
$T = 60^\circ\text{C}$	$+1.0$	1.00
$T = 30^\circ\text{C}$ (claims -0.3 , rule demands -0.5)	$+0.2$	0.04
$T = 25^\circ\text{C}$ (obeys the rule)	0.0	0.00
Average = physics loss		0.347

Training turns the knobs until this number shrinks towards zero, so the network's slopes line up with the rule everywhere we check.

Try these yourself

1. The coffee reads $90\text{ }^{\circ}\text{C}$. What slope does the rule demand? (*Answer: $-3.5\text{ }^{\circ}\text{C per minute.}$*)
2. It reads $25\text{ }^{\circ}\text{C}$. What slope? (*Answer: $-0.25\text{ }^{\circ}\text{C per minute.}$*)
3. At $40\text{ }^{\circ}\text{C}$ a network claims a slope of -0.5 . What is the residual? (*Rule demands -1.0 , so the residual is $+0.5$.*)

TIP

Notice that no solution of the equation was needed. We only measured a value and a slope and compared them with the rule. That is why this method is so general.

IN ONE LINE

The physics loss is the squared mismatch between a slope the network reports and the slope the rule demands, averaged over checkpoints.

11

Worked Example: The Cooling Coffee

Real training, real numbers.

Time to see it work. The setup is deliberately simple so every part is visible.

Item	Value
Room temperature	20 °C
Coffee starts at	90 °C
True cooling constant k	0.05 per minute
Measurements	6 readings at 0, 2, 4, 6, 8, 10 minutes, with about 1 °C of noise (simulated)
Physics checkpoints	61, one per minute from 0 to 60
Network	2 hidden layers of 24 calculators, tanh bends
Question	Predict the temperature for the whole hour, though data stops at minute 10

Two networks, same data

- **Network A** uses the data loss only.
- **Network B** uses data loss plus the physics loss, with the cooling rule and the true k supplied.

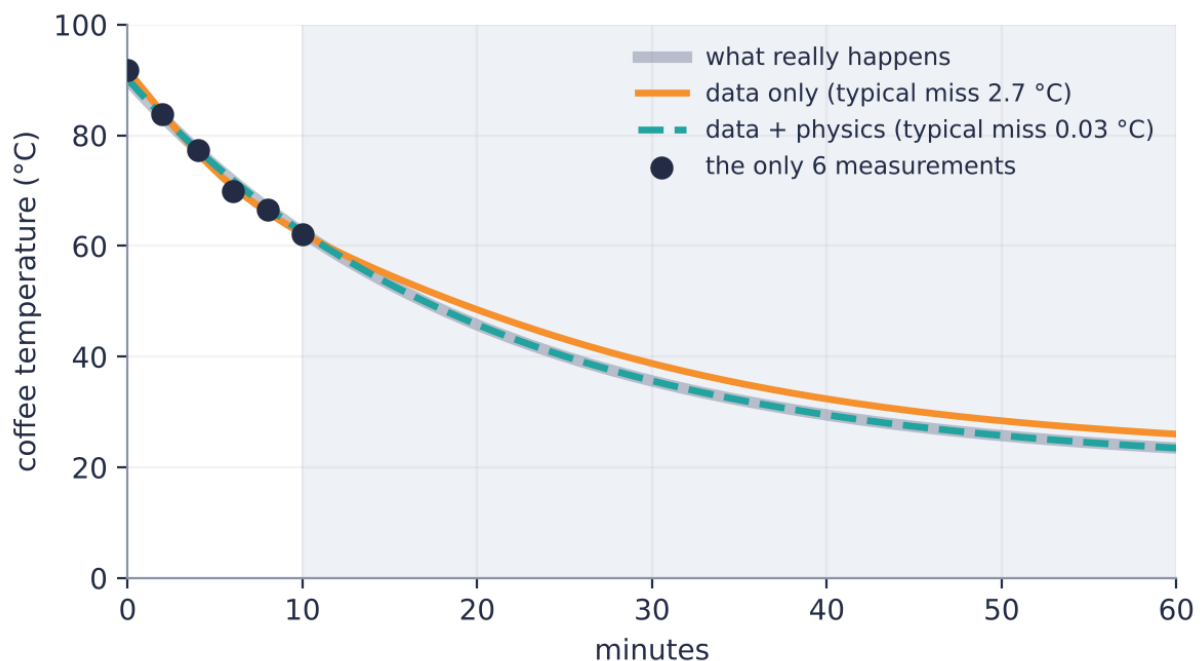


Figure 14. Both networks saw the same six readings. The shaded zone is where no data exists. Real training results.

The scorecard

Model	Typical miss over minutes 10 to 60	Prediction at minute 60 (truth: 23.5 °C)
Curve through every dot	763,243 °C	-2,386,451 °C
Network, data only	2.69 °C	26.0 °C
Network, data + physics	0.030 °C	23.5 °C

The physics-informed network is roughly **91 times more accurate** over the unseen hour than the data-only network, from the very same six readings. Notice what changed: not the data, not the network's size, only an extra penalty for breaking one line of physics.

The training curve

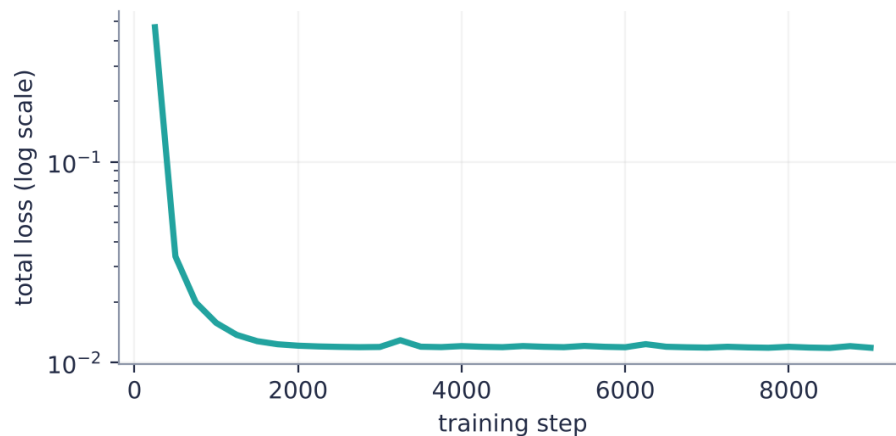


Figure 15. The total loss falls steadily as the network satisfies both teachers.

WATCH OUT

This example is easy on purpose: the law is exact, we supplied the true k , and the data are simulated. Chapter 18 explains what gets harder with real equipment, wrong equations and noisy data.

IN ONE LINE

Six readings plus one line of physics beat six readings alone by about a hundredfold over the unseen hour.

12

Watching the Network Learn

Four snapshots from the same training run.

Numbers in a table hide the drama. Here is the physics-informed network of Chapter 11 caught at four moments of training. In every panel the grey band is the truth, the teal line is the network's guess and the dots are the six readings.

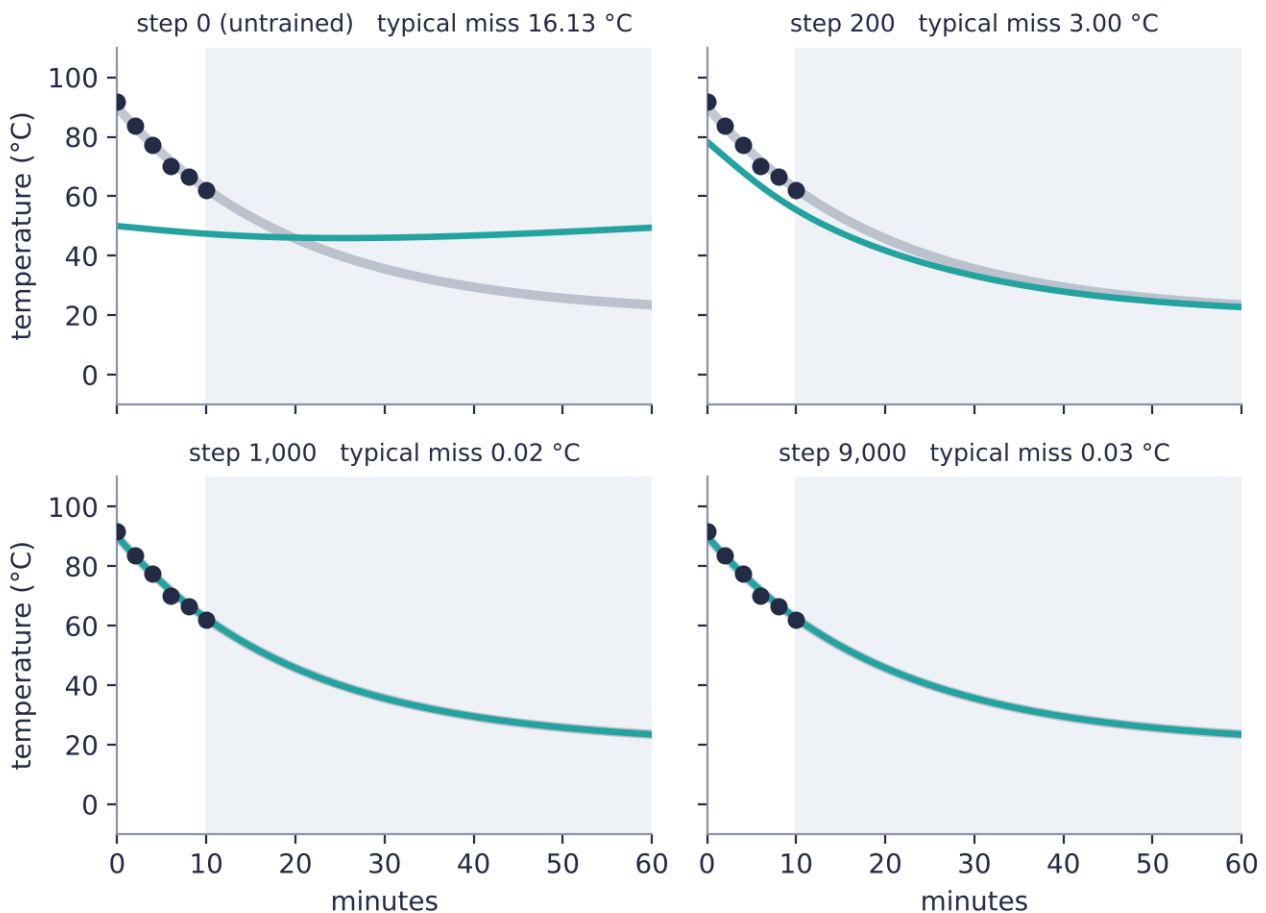


Figure 16. The same network at four moments of training. Real snapshots from the run behind Chapter 11.

What each snapshot shows

Step	What the network knows	Typical miss
0	Nothing yet. Its knobs are random, so its guess is a meaningless curve.	16.1 °C
200	It has found the rough downward trend, but the shape is still off.	3.0 °C
1,000	Data and physics now agree almost everywhere, even in the shaded region with no readings.	0.022 °C
9,000	Little has changed since step 1,000. The score is as low as this setup can get.	0.030 °C

Two things are worth noticing. First, the shaded region improves **even though no measurement ever lived there**. Only the physics checkpoints reach it. Second, training does not improve forever. After about a thousand steps the typical miss stops falling and wobbles slightly, which is normal. In practice you stop when the loss flattens.

Why the early steps look so bad

At step 0 the network is a random curve, so it fits neither the readings nor the rule. Both losses are large and both push in useful directions. The data loss pulls the curve towards the dots on the left. The physics loss pulls the slope everywhere towards the cooling rule. Early on the two pulls dominate the random starting shape quickly, which is why a few hundred steps are enough to reach a sensible curve.

What the snapshots do not show

- The run used a fixed random start. A different start gives a slightly different path, though similar ending.
- The final score is limited by the six noisy readings, not by more training.
- With a harder equation, curves can freeze in poor shapes for thousands of steps before improving. Plotting snapshots is how you notice.

A useful habit

Always look at the picture, not just the loss number. A network can reach a low loss and still show a visibly odd curve if the problem is badly set up. Plotting snapshots like these is the quickest debugging tool you have.

SAY IT LIKE THIS

"Training did not need more data. The physics checkpoints pulled the whole curve into line."

IN ONE LINE

Plot the network at several training steps; you can watch the physics pull the guess into shape.

13

Turning the Dials

What happens when we change the loss weight or the number of checkpoints.

Chapter 7 said the weight λ matters and Chapter 9 said checkpoints matter. Here is the evidence. We retrained the same network on the same six readings, changing one dial at a time. Each setting was run once from the same random start.

Dial 1: the weight on the physics penalty

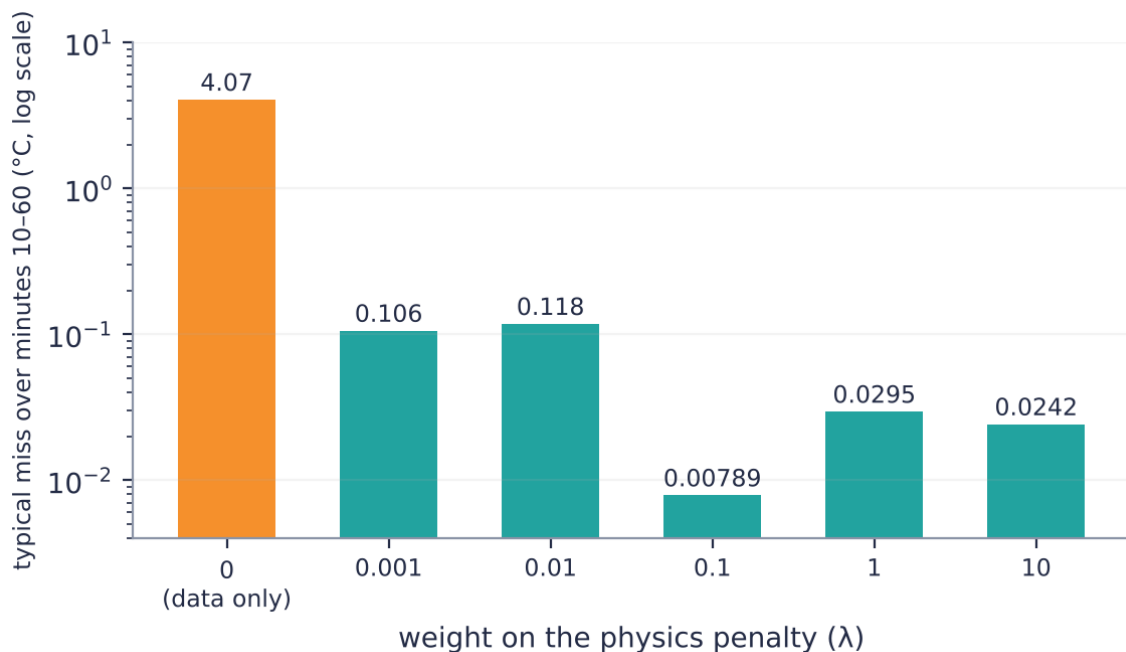


Figure 17. Typical miss over the unseen minutes for different physics weights. Real results, one run each.

Adding even a tiny physics penalty (0.001) cut the typical miss from about **4.1 °C** to **0.11 °C**. Values from 0.1 to 10 all did about as well, between 0.008 and 0.030 °C.

WATCH OUT

The differences among the larger weights are small and come from a single run each. Do not read a ranking into them. The honest lesson is that this easy, clean problem forgives the weight. Harder problems often do not.

A note on the first bar: this "data only" run used longer training than the one in Chapter 4, and it missed by 4.1 °C rather than 2.7 °C. The exact miss of a data-only model changes with training settings, which is another sign that the data alone does not decide the answer between the dots.

Dial 2: how many physics checkpoints

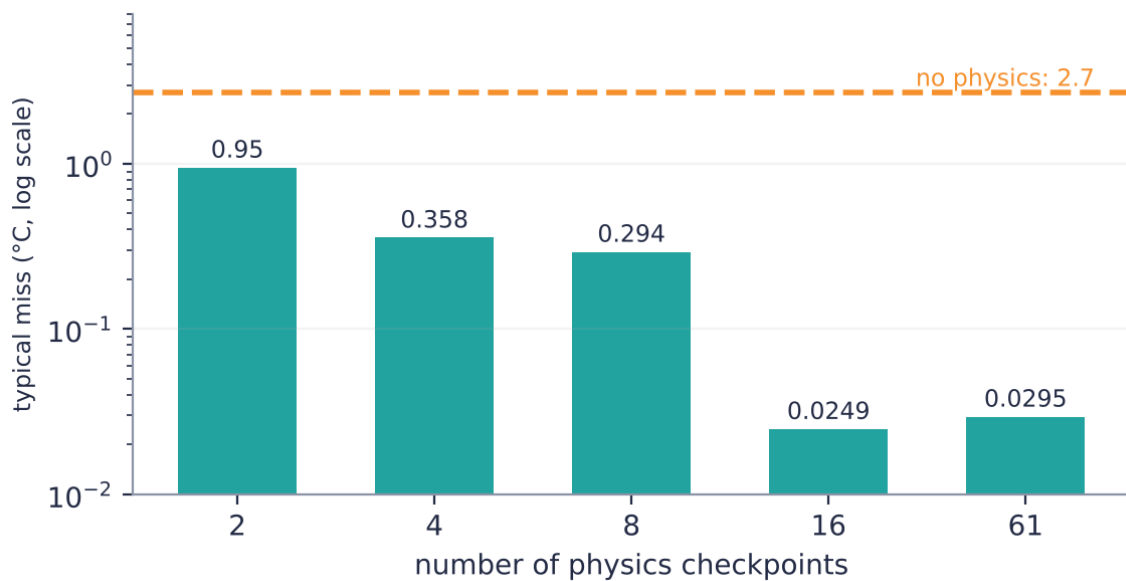


Figure 18. Typical miss against the number of physics checkpoints (the dashed line is the no-physics network of Chapter 4). Real results, one run each.

Even two checkpoints helped: the typical miss fell from 2.7 °C to 0.95 °C. Sixteen checkpoints brought it down to 0.025 °C, and sixty-one did no better. More points cost more computing time, so there is a sensible stopping point.

Practical rules of thumb

- Start with the physics weight at about the same size as the data loss, then try a factor of ten either way.
- Increase the checkpoints until the result stops changing.
- Change one dial at a time and keep a log. Random starts alone can change results, so repeat important runs with different seeds.

IN ONE LINE

Some physics beats none, enough checkpoints help up to a point, and every dial deserves a quick experiment.

14

Running It Backwards: Discovering Unknown Physics

When the rule is known but a number in it is not.

So far we handed the network the rule *and* its constant, k . In practice you often know the form of the law but not the numbers in it: how well is this wall insulated, how conductive is this material, how fast does this particular cup cool?

Make k a knob. The network now learns two things at once: the curve and the number that governs it. This is called an **inverse problem**. Finding the curve when the rule is known is the **forward problem**.

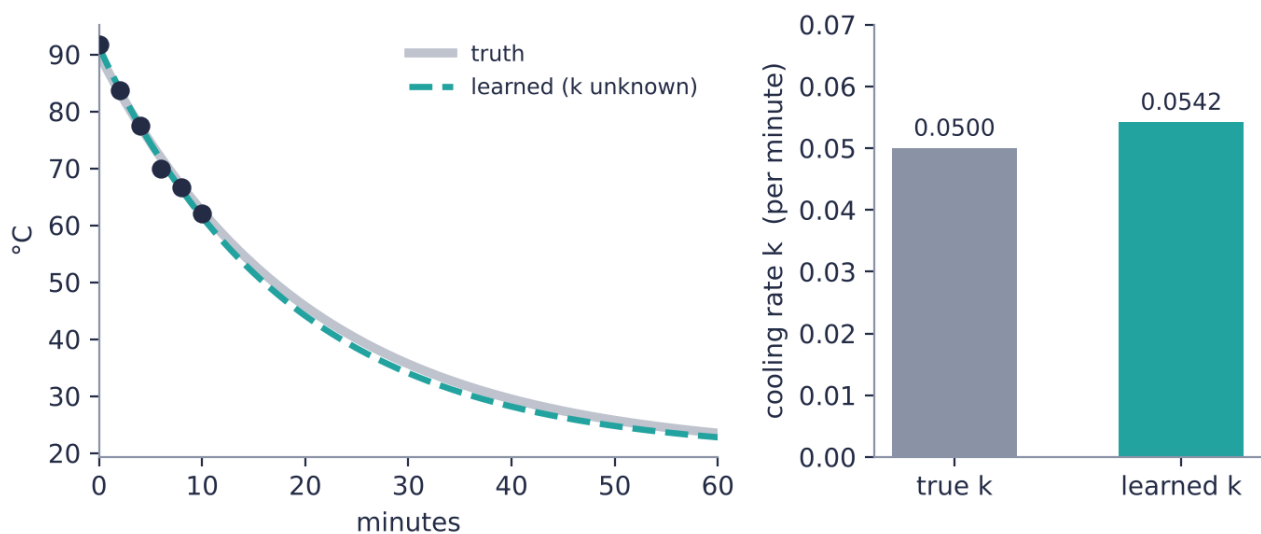


Figure 19. The network was told only the form of the law. It learned the cooling constant from six noisy readings. Real result.

The true k is 0.0500 per minute. The network recovered **0.0542**, within about 8%. Its curve is not as sharp as before (typical miss 1.35 °C) because it had to estimate k from six noisy readings. Honest learning from limited data costs some accuracy, and that is expected.

Why this matters in practice

Field	An unknown the network can learn
Buildings	Thermal resistance of a wall, air leakage.
Materials	Thermal conductivity, stiffness.
Medicine	Blood-vessel stiffness from flow measurements.
Geoscience	Rock permeability from well readings.

TIP

Inverse problems are where physics-informed methods often shine, because classical methods need many repeated simulations to hunt for the unknown number.

IN ONE LINE

Make the unknown constant a knob and the network learns it while it learns the curve.

A cup of coffee has one temperature. A room has a temperature at every *point*, and it changes over time. Such a thing is called a **field**. The rules for fields involve slopes in space as well as in time, and are called **partial differential equations (PDEs)**.

A field: a number at every point, and it evolves

temperature $T(x, t)$ changes along the rod and over time



Figure 20. A rod with a hot end and a cold end. The temperature depends on position and on time.

Same recipe, bigger input

The network now takes position (and time) as input and returns the temperature there. Everything else is unchanged: a data loss at sensor locations, a physics loss at checkpoints scattered through space and time, and extra penalties for the **boundary conditions** (the hot wall, the cold window) and **initial condition** (where everything starts).

A sensor picture

Imagine a room with a warm wall on the left and a cold wall on the right, and only five sensors. We solved the real temperature field numerically, then asked: how well can you rebuild the whole room from five readings by smooth interpolation alone?

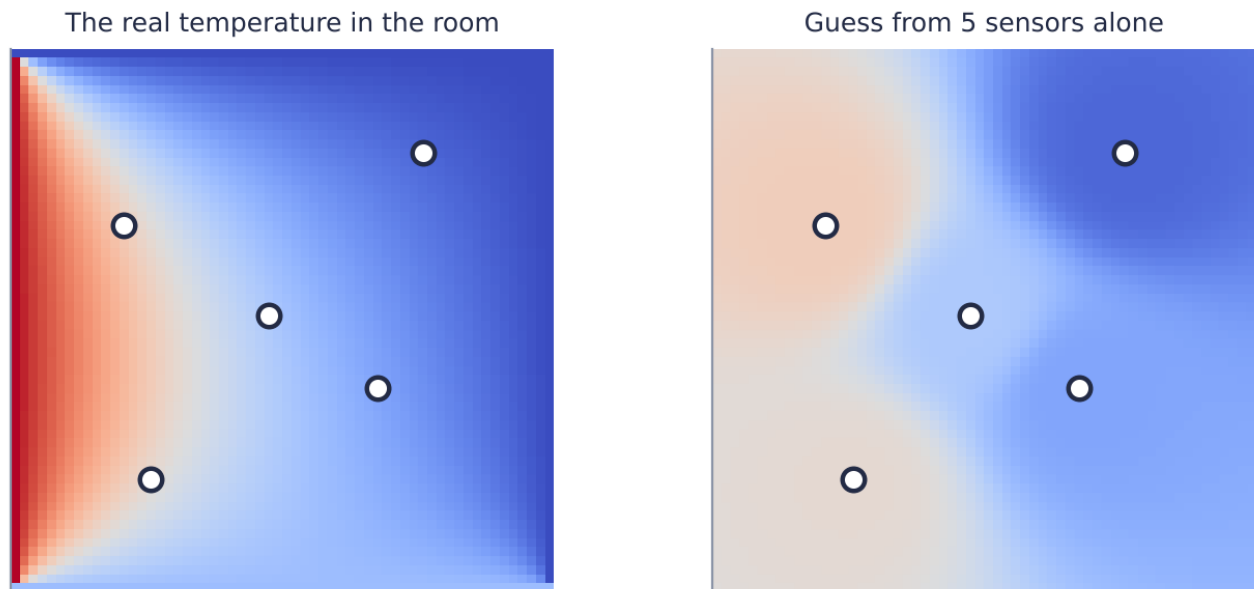


Figure 21. Left: the real field. Right: a guess from five sensors using distance-weighted averaging. Computed for this guide.

The guess captures the broad idea but misses the sharp heat near the warm wall. Its typical error is about 17% of the hot-to-cold range. A physics-informed model uses the same five sensors *plus* the heat equation and the known wall temperatures, which is exactly the information that interpolation throws away.

Fields of interest

Field	Typical sensors	Typical physics
Indoor temperature	A handful per floor	Heat flow, air movement
Air quality	A few monitors	Diffusion and transport
Structural stress	Strain gauges	Elasticity
Groundwater	A few wells	Flow through porous rock

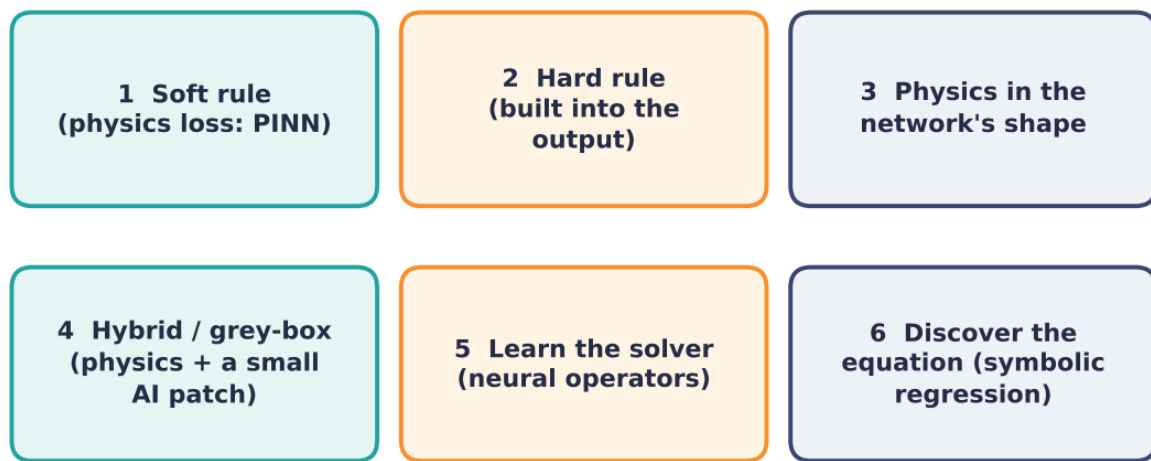
IN ONE LINE

A field is the same idea at larger scale: sparse sensors, a PDE as the second teacher.

The penalty is only one of six options.

Everything so far used a **soft rule**: the physics is a penalty that the network can be talked into bending a little. There are other approaches, and each one trades something for something else.

Ways to give an AI physics knowledge



Same goal everywhere: use what we know about the world so we need less data.

Figure 22. Six families of physics-aware learning.

Family	How it works	Strength	Watch for
1 Soft rule (PINN)	Equation residual added to the loss.	Flexible; handles forward and inverse problems.	Hard to balance and train.
2 Hard rule	Output built so the law holds exactly (for example, a result that always conserves mass).	The rule can never be violated.	Needs a clever construction per problem.
3 Physics in the shape	Network architecture mirrors symmetries or structure.	Efficient; generalises well.	Needs knowledge of the symmetry.
4 Hybrid / grey-box	A physical model plus a small learned correction.	Trustworthy and interpretable.	Needs a decent physical model.
5 Neural operators	Learn the mapping from conditions to full solutions (for example DeepONet, Fourier neural operator).	Fast once trained; reusable across cases.	Needs many simulated examples to learn from.
6 Equation discovery	Search for the equation itself from data (for example SINDy).	Produces readable equations.	Sensitive to noise.

Black, white and grey

A **white-box** model is all physics: transparent but only as good as your assumptions. A **black-box** model is all data: flexible but opaque. A **grey-box** model blends them. Most of the practical, trustworthy systems in buildings and industry end up grey.

TIP

Start with the simplest family that works. A hybrid with a small correction often beats an elaborate network.

IN ONE LINE

Physics can enter as a penalty, a hard constraint, the network's shape, a hybrid model, a learned solver or a discovered equation.

Physics-informed methods suit problems where measurements are costly, equations are known at least roughly, and decisions depend on what happens between and beyond the sensors.

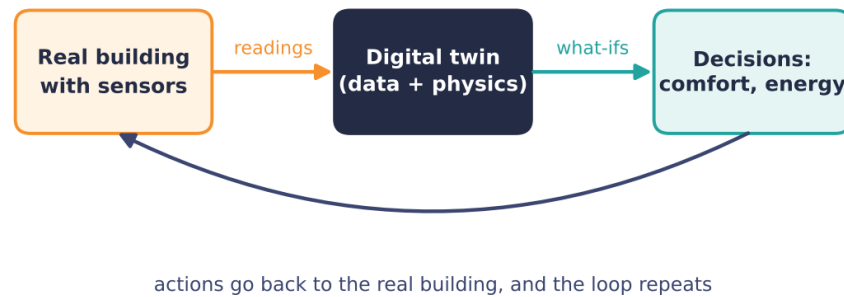


Figure 23. A digital twin loop: sensors feed a model that combines data and physics, and decisions go back to the real asset.

Digital twins

A **digital twin** is a live virtual copy of a real thing: a building, a machine, a field. Sensors keep it up to date. Because sensors are sparse, the twin needs physics to fill in the unmeasured places and to answer what-if questions ("what if we open the windows at night?"). Physics-informed learning is a natural engine for this.

Area	What the AI estimates	What physics supplies
Buildings and cities	Temperature, airflow and energy use everywhere, from a few sensors.	Heat and fluid equations, conservation of energy.
Fluids and aerodynamics	Flow around objects, wind in streets.	Navier-Stokes equations.
Structures	Stress and damage in beams, bridges and walls.	Elasticity equations.
Climate and weather	Fast surrogate for slow simulations.	Conservation laws, transport.
Medicine	Blood flow, tissue heating from scans.	Fluid flow and heat equations.
Manufacturing and agriculture	Process temperature, irrigation, drying.	Heat and mass transfer.

A common pattern

1. Choose the physical process that matters (heat, flow, stress).
2. Collect the few measurements you can afford.
3. Train a model that respects the equation.
4. Use it to see the full picture, test scenarios and spot anomalies.
5. Keep checking it against new readings and retrain when reality drifts.

IN ONE LINE

Where data is sparse and the physics is known, the combination beats either alone.

Physics-informed AI is not magic. Researchers have documented real difficulties, and a good user knows them before starting.

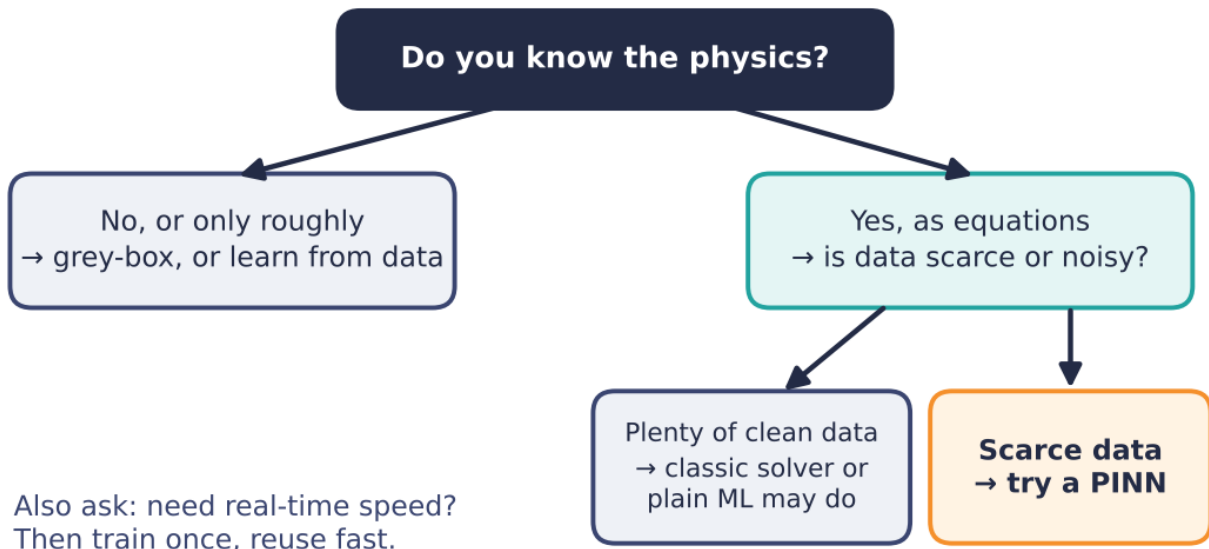


Figure 24. A quick decision guide: when to consider physics-informed learning.

Eight classic pitfalls

Pitfall	What happens	What to do
Wrong equation	The network faithfully learns the wrong physics.	Check the model against data; consider a hybrid.
Unbalanced losses	One teacher drowns out the other.	Tune λ ; use methods that balance automatically.
Hard training	Training stalls in poor solutions, especially for fast waves or sharp fronts.	Smaller steps, more checkpoints, change the formulation.
Too few checkpoints	Physics is only enforced where you looked.	Add points where the residual is large.
Noisy data	The network chases noise.	Weight data by reliability; add the physics.
Complex geometry	Boundaries are fiddly to impose.	Hard-constraint or mesh-based methods may be easier.
Slow to train	Each new case needs a new training run.	Use neural operators if you need many cases.
Over-trust	A confident plot is not a verified answer.	Validate on held-out measurements.

When a classic solver is better

If you know the geometry, the materials and the conditions exactly, a well-tested simulator is usually faster, more accurate and easier to trust than a trained network. Physics-informed learning earns its place when data is partial, noisy or expensive, when you need to estimate unknown parameters, or when the model must blend with live measurements.

WATCH OUT

In this guide's example, every ingredient was ideal: exact law, exact constant, clean setup. In real projects, assume the equation is approximate and the numbers are uncertain, and design checks accordingly.

Questions to ask before trusting a result

- Which equation was used, and who checked it applies here?
- How were the two losses balanced?
- Was the model tested on measurements it never saw during training?
- How does it behave outside the range of the data?
- Is a plain simulator or plain regression just as good?

IN ONE LINE

Check the physics, balance the losses, validate on unseen data, and compare with simpler methods.

The recipe

1. Write the physical rule as an equation and move everything to one side to get the **residual**.
2. Build a small network: input time (or position), output the quantity of interest.
3. Pick measurement points for the data loss and many checkpoints for the physics loss.
4. Add the two losses with a weight λ .
5. Train by gradient descent with autodiff supplying every slope.
6. Test on measurements the network never saw, and compare with a plain network.

A minimal sketch

This is the core of the script that produced the coffee results. It uses plain Python with NumPy and the small **autograd** package (`pip install autograd`). It is a sketch of the method, not a polished library.

```
import autograd.numpy as np
from autograd import grad, elementwise_grad

def net(p, ts):
    # tiny tanh network
    h = ts[:, None]
    for W, b in p[:-1]: h = np.tanh(h @ W + b)
    W, b = p[-1]
    return (h @ W + b)[:, 0]

T = lambda p, ts: 50.0 + 30.0 * net(p, ts)      # scale output to degrees
dT = elementwise_grad(lambda ts, p: T(p, ts), 0) # autodiff slope

def loss(p):
    data = np.mean(((T(p, t_meas / 60) - y_meas) / 10) ** 2)
    ts = t_check / 60 # physics checkpoints
    resid = dT(ts, p) / 60 + k * (T(p, ts) - 20.0) # dT/dt + k (T - Troom)
    return data + 100.0 * np.mean(resid ** 2)

# then: repeat p = p - step * gradient(loss)(p) (we used the Adam rule)
```

Tools worth knowing

Tool	What it is	Good for
PyTorch, JAX	General deep-learning libraries with autodiff.	Writing your own, flexible.
DeepXDE	A library built for physics-informed networks.	Learning PINNs quickly.
NVIDIA PhysicsNeMo (formerly Modulus)	A framework for physics-based machine learning.	Larger problems on GPUs.

TIP

Library names and versions change quickly. Check each project's page for its current name and install steps before relying on this table.

An eight-week plan

Week	Focus	Do this
1	Machine learning basics	Fit a line and a small network to any dataset. Watch the loss fall.
2	Overfitting	Reproduce the failing-curve figure of Chapter 4 with your own sparse data.
3	Rates of change	Sketch slopes on three curves; read the coffee equation aloud.
4	Autodiff	Take the slope of a tiny network and compare it with a numerical slope.
5	First PINN	Run the sketch above. Change λ and watch what breaks.
6	Inverse problem	Make k unknown. Try noisier data.
7	A field	Try a 1D heat equation with boundary conditions.
8	Your own problem	Pick a small physical question from your work and compare PINN, plain ML and a simple model.

IN ONE LINE

Start tiny, change one thing at a time, and always compare with a plain network.

Six Myths and the Facts

Common misunderstandings, cleared up.

Myth	Fact
"The network discovers physics from scratch."	In a PINN the equation is supplied. The network learns the <i>solution</i> (or unknown constants in it), not the law.
"Physics-informed means correct."	It is only as right as the equation, the conditions and the training. A wrong equation gives a confidently wrong answer.
"It replaces simulators."	They complement each other. With full knowledge of geometry and materials, a simulator is often faster and easier to trust.
"It needs no data."	It still needs boundary and initial conditions or some measurements, otherwise many different solutions satisfy the same equation.
"More checkpoints always help."	They help up to a point (Chapter 13) and then only cost computing time.
"A low loss means a good answer."	Loss is only a score on points you chose. Test on measurements the network never saw, and plot the result.

SAY IT LIKE THIS

"The physics is an ingredient, not a guarantee."

How to talk about results honestly

- Say which equation was used and what was assumed.
- Say how many measurements and checkpoints were used.
- Report the error on data the model did not see.
- Compare against at least one simple baseline.

IN ONE LINE

A physics-informed model is a powerful tool, not an oracle; check it like any other model.

Quick Quiz

Ten questions. Answers are on the next page.

1. What does the loss measure?
2. Why does a flexible model do badly between sparse data points?
3. What is a physical law, in this guide's sense?
4. In the coffee equation, what does a bigger k mean?
5. What is the *residual* of an equation?
6. What does automatic differentiation provide?
7. What are collocation points?
8. What is the difference between a forward and an inverse problem?
9. Name two ways other than a loss penalty to add physics to a model.
10. When might a classic simulator beat a physics-informed network?

Quiz Answers

No peeking until you have tried.

1. How wrong the model is, as one number. Training tries to make it small.
2. The data pins the curve down only at the data points; infinitely many curves fit them equally well.
3. A rule that has held in every careful test, usually written as an equation about how something changes.
4. The coffee cools faster for the same temperature gap.
5. What is left when you move everything in the equation to one side. It is zero when the rule is obeyed.
6. The exact slope of the network's output, for free, so the physics rule can be checked anywhere.
7. Points where the physics rule is checked, even though nobody measured there.
8. Forward: the rule and constants are known, find the curve. Inverse: the form is known, find unknown constants (or the curve) too.
9. Hard constraints built into the output, physics in the network's shape, hybrid models, learned solvers, equation discovery.
10. When geometry, materials and conditions are known exactly, and the problem is well suited to existing solvers.

Glossary

Plain definitions of the words in this guide.

Term	Meaning
Activation function	The fixed bend each calculator in a network applies, such as tanh.
Autodiff	Automatic differentiation: a method for finding exact slopes through any chain of arithmetic.
Boundary condition	What is true at the edges of the region, such as a fixed wall temperature.
Chain rule	The overall slope of chained steps equals the product of each step's slope.
Collocation point	A location where the physics rule is checked.
Data loss	The penalty for disagreeing with measurements.
Digital twin	A live virtual copy of a real asset, kept up to date by sensors.
Field	A quantity with a value at every point in space, such as room temperature.
Forward problem	Rule and constants known; find the solution.
Gradient descent	Repeatedly nudging the knobs to reduce the loss.
Grey-box model	A blend of physical model and learned correction.
Initial condition	Where the system starts.
Inverse problem	Using measurements to find unknown constants in a known rule.
λ (lambda)	The weight that balances physics loss against data loss.
Loss	One number saying how wrong the model is.
Neural network	A layered stack of adjustable calculators.
Neural operator	A network that learns the whole mapping from conditions to solutions.
Overfitting	Matching noise or quirks of the examples instead of the true pattern.
PDE	Partial differential equation: a rule about change in space and time.
Physics loss	The penalty for breaking the physical rule.
PINN	Physics-informed neural network.
Residual	What remains when an equation is rearranged to equal zero.
Surrogate	A fast stand-in for a slow simulation.
Weight, bias	The knobs of a neural network.

One-Page Cheat Sheet

Pin it next to your screen.

Idea	In short
Why	Few examples, but known physics. Combine both.
Data loss	Gap between model answers and measurements.
Physics loss	Residual of the equation, averaged over many checkpoints.
Total loss	Data loss + $\lambda \times$ physics loss.
Autodiff	Gives the slopes the physics loss needs.
Collocation points	Where physics is checked; free to add.
Inverse problem	Make unknown constants into knobs.
Cooling coffee rule	$dT/dt = -k (T - T_{\text{room}})$
Residual (coffee)	$dT/dt + k (T - T_{\text{room}})$
Real result here	Same six readings: typical miss 2.69 °C (data only) versus 0.030 °C (data + physics).
Trust test	Validate on unseen data; compare with a simple baseline.

Say it in 20 seconds

SAY IT LIKE THIS

"Physics-informed AI trains a network on the measurements and also penalises it wherever it breaks a known law, so it behaves sensibly where there is no data. It is most useful when data is scarce and the physics is known."

Further Reading

Where the ideas come from.

These references are written from memory. Please check titles, years and details before you cite them.

Work	Why read it
Raissi, Perdikaris and Karniadakis (2019). Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. <i>Journal of Computational Physics</i> .	The paper that popularised PINNs.
Karniadakis et al. (2021). Physics-informed machine learning. <i>Nature Reviews Physics</i> .	A broad, readable overview of the whole family.
Lagaris, Likas and Fotiadis (1998). Artificial neural networks for solving ordinary and partial differential equations. <i>IEEE Transactions on Neural Networks</i> .	An early forerunner of the idea.
Lu et al. (2021). DeepXDE: a deep learning library for solving differential equations. <i>SIAM Review</i> .	The library paper, good for hands-on learning.
Krishnapriyan et al. (2021). Characterizing possible failure modes in physics-informed neural networks. <i>NeurIPS</i> .	An honest look at training difficulties.
Wang, Teng and Perdikaris (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. <i>SIAM Journal on Scientific Computing</i> .	Why balancing losses is hard, and fixes.
Chen et al. (2018). Neural ordinary differential equations. <i>NeurIPS</i> .	Networks that define the rate of change.
Lu et al. (2021). Learning nonlinear operators via DeepONet. <i>Nature Machine Intelligence</i> .	The neural-operator family.
Li et al. (2021). Fourier neural operator for parametric partial differential equations. <i>ICLR</i> .	A fast operator learner.
Brunton, Proctor and Kutz (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. <i>PNAS</i> .	Equation discovery (SINDy).

About this guide

Written and illustrated by Claude (Anthropic), prompted and directed by Mahmoud A. The booklet is a general introduction and not personal advice, and it is not a substitute for a textbook or for checking your own engineering. The coffee and room results were computed for this guide from simulated data, so they show how the method behaves, not how any real device performs.