BOOK SAMPLER

AI Distilled

A Human Guide to the World of Artificial Intelligence

Steve Packard CISSP

Infographics and Diagrams

35 illustrated pages from AI Distilled.

Neural networks, backpropagation, attention, transformers and language generation.

Employment disruption, autonomous vehicles, insurance and the moral crisis.

From compact neuron diagrams to full-page infographics, including the Irish-last-name example, 20 questions and the murder mystery.

Complete original pages preserve the graphics, captions and accompanying explanations. Use the PDF bookmarks to browse by topic.

AIDistilledBook.com

First edition | Copyright 2026 Stephen M. Packard

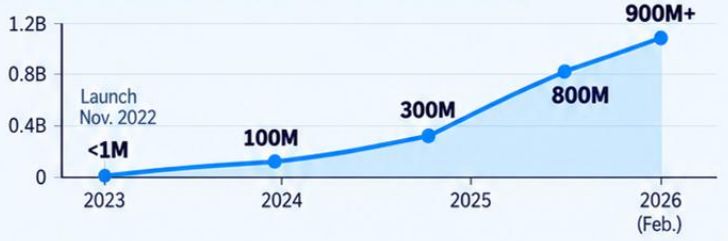
THE GENERATIVE AI TAKEOFF

Four growth curves behind a global shift

In just a few years, generative AI has gone from a research curiosity to a mainstream technology with global economic, social and political impact.

1 A MASS AUDIENCE

ChatGPT weekly active users · key milestones [1]



900M+

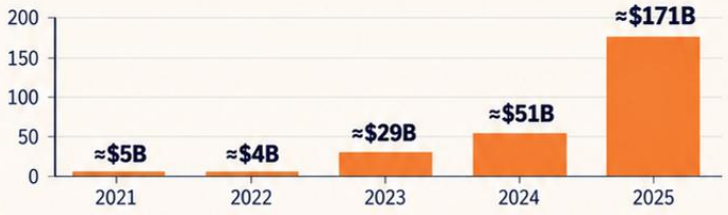
weekly active users

as of Feb. 2026

That's more than **10%** of the world's population using a single AI tool.

2 CAPITAL POURS IN

Global private investment in generative AI · annual, approx. (US$ billions) [2]



$171B

invested in 2025

Private investment in generative AI has grown more than **30×** since 2021.

3 BUSINESSES PUT IT TO WORK

Share of organizations using generative AI in at least one function [3]



79%

of companies use generative AI

in at least one business function in 2025, up from 33% in 2023.

AI has moved from early experimentation to a core part of business operations.

4 HIRING DEMAND SURGES

U.S. job advertisements naming generative AI · annual counts (rounded) [4]



138K

U.S. job ads in 2025 mention generative AI

More than 8× higher than in 2023.

Demand for AI skills is rising across many industries, from tech to finance, healthcare and professional services.

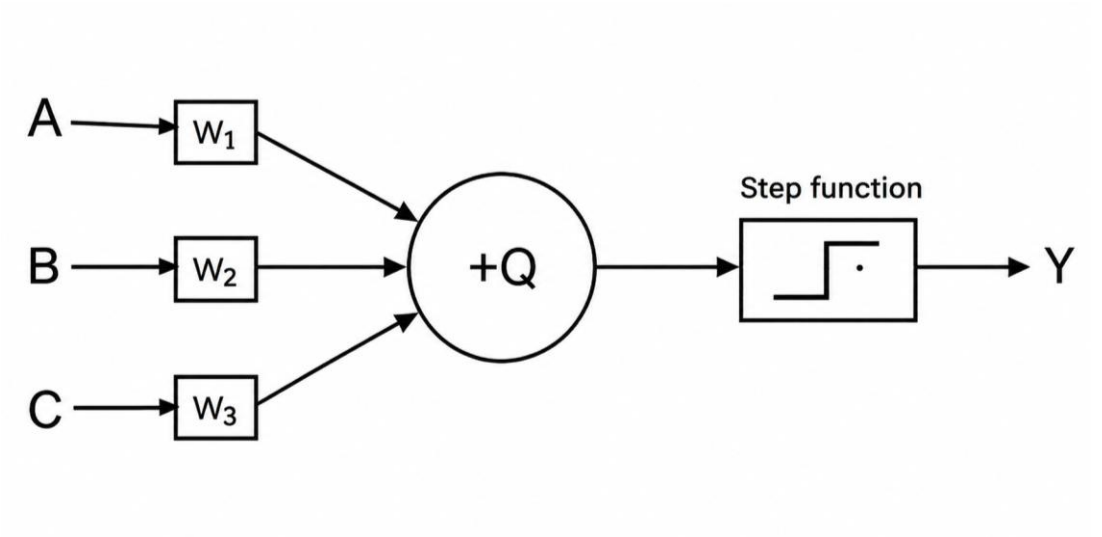
[1] OpenAI. "ChatGPT weekly active users." OpenAI (Feb. 2026). Earlier milestones: OpenAI (Dec. 2024); TechCrunch (Nov. 6, 2023); Tech.eu (Oct. 7, 2025).

[2] **Stanford HAI.** "AI Index Report 2026," p. 179, Fig. 4.2.3. Data: Quid (2025).

[3] McKinsey & Company. "The state of AI in 2025," p. 193, Fig. 4.3.1.

[4] Lightcast. "Generative AI job postings." AI Index Report 2026, p. 207, Fig. 4.4.5.

A neuron in its simplest form: showing weighted inputs, bias and step function output.



In the above diagram, where A, B, and C are inputs, W1, W2, and W3 are weights and Q is the bias, a value added once to the weighted sum, the total is ((A*W1)+(B*W2)+(C*W3))+Q. If that total reaches the activation threshold of the neuron, it fires and outputs Y. The output Y can be considered to be one or zero, depending on whether the activation threshold is met. This is the basic operation of a McCulloch–Pitts (MCP) neuron, though in this case it has activating but no inhibiting inputs.

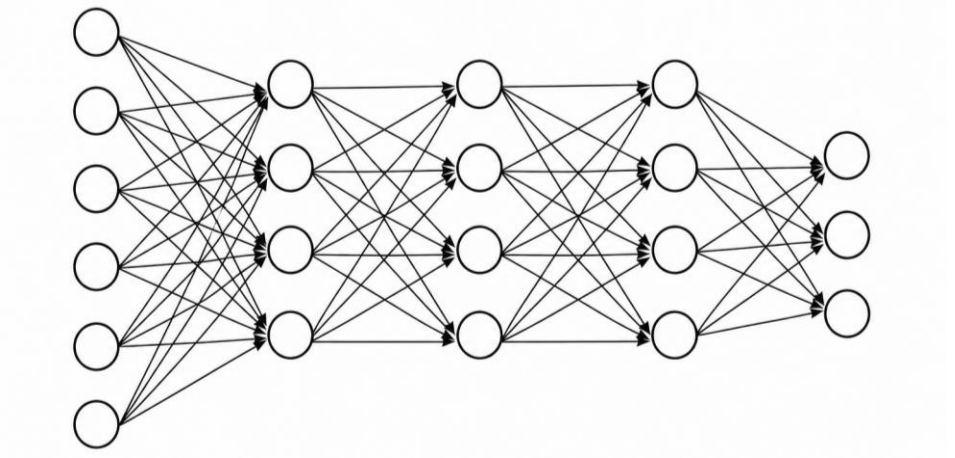
Current neurons do not use step functions, where Y is either on or off. Instead, a smoother function is used to determine the output Y, making it continuously variable. So for the diagram, Y=Activation(((A*W1)+(B*W2)+(C*W3))+Q). Different types of activation functions are used in different circumstances. A purely linear function will not work in a large neural network, because the output collapses to a single linear sum.

The best activation function depends on the network and task, and finding it often takes trial and error. One of the most commonly used is the ReLU function.[1] It's efficient and works well for most tasks. However, many modern networks, such as large language models, use modified versions of ReLU, which have a smoothed activation. Sigmoid activation functions are also useful for some types of neural networks.

In a neural network, the weights and biases of the neurons are variables which are learned through the training process. These are known as parameters. The number of neurons in the network, how they interconnect, and which activation functions are used are chosen ahead of time, by the developer. These specifications are sometimes known as hyperparameters.

Multi-Layer Perceptron

The multi-layer perceptron takes the same idea and adds more layers of computation. As with other hyperparameters, exactly how many layers depends on the task, but some modern networks have hundreds. MLPs were first experimented with in the 1960s, but proved far more complex and challenging to simulate and train. For this reason, MLPs didn't become commonplace until the 1980s, in part due to the discovery of backpropagation as a learning algorithm. This architecture is still used today. Large Language Models, like those used in ChatGPT, Claude and Gemini, contain MLP network layers as part of their architecture. In a large language model, the input is processed through hundreds of multi-layer perceptrons, in addition to "attention" layers, which are discussed later in this book.

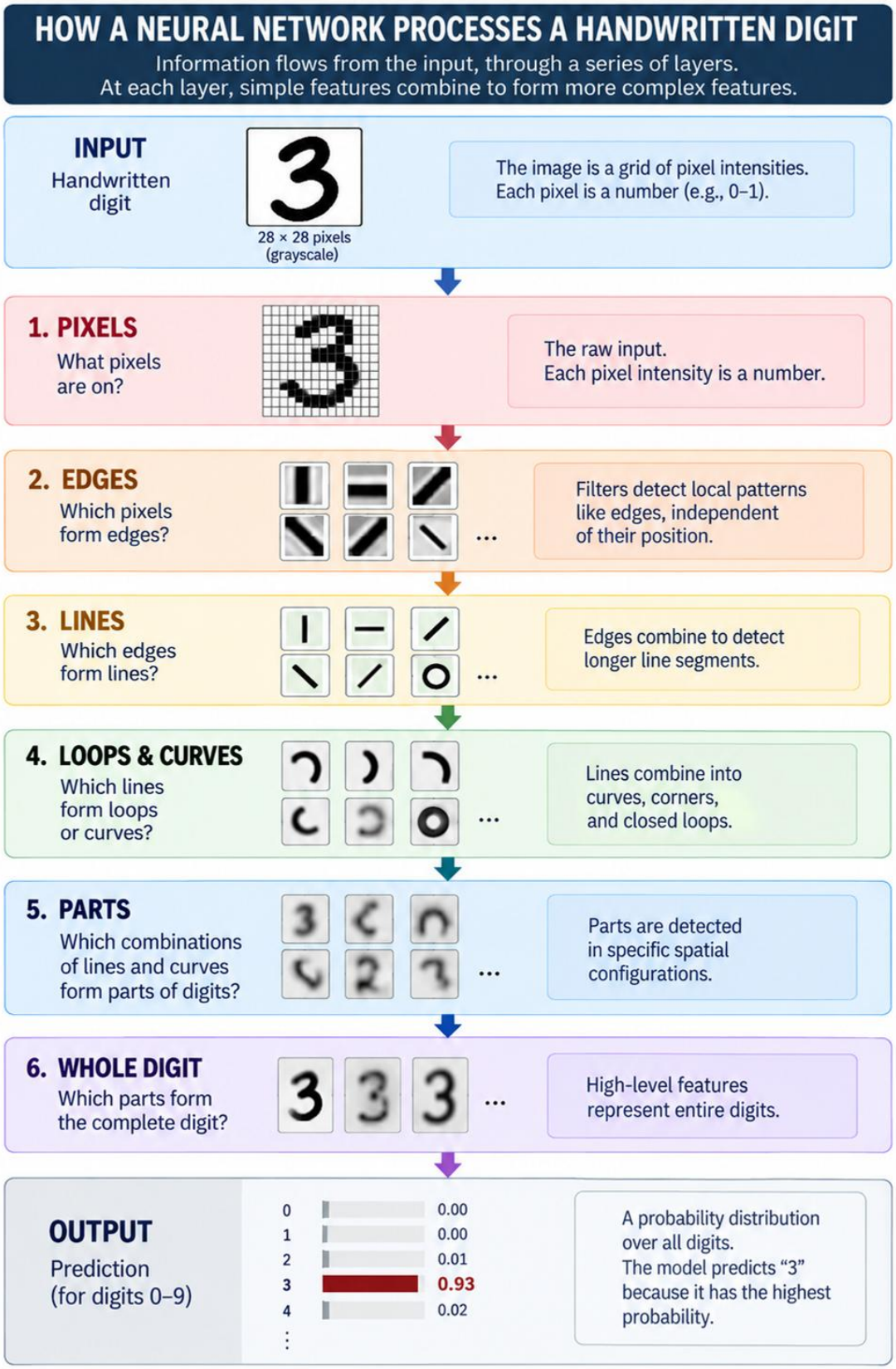


The MLP is the simplest topology of a neural network that can operate as a true function approximator. In other words, with enough layers and properly configured parameters, the MLP can approximate continuous input-output relationships over bounded ranges, although in practice there's no guarantee that training will properly generalize to a given function.[2]

The groups of nodes between the input and output nodes are known as the "hidden layers" and have "hidden activations." These are not seen by the end user. They're just part of the computation and help arrive at the final value.

Some neural networks compress many input values into hidden layers with fewer neurons. These are useful for tasks which benefit from finding commonality and combining features. Other neural networks do the opposite, expanding a smaller input space to larger hidden layers. By expanding the space, more complex relationships between input variables can be processed. The task determines the number and size of the layers. Larger networks are more capable, but require more powerful hardware to run and take longer to train.

Modern neural networks are enormous. Some have billions or even trillions of parameters. If every node in the network is connected to every node in the next layer (as it is with a conventional MLP), it's easy to see how the number of connections can quickly explode.


HOW A NEURAL NETWORK PROCESSES A HANDWRITTEN DIGIT
Information flows from the input, through a series of layers.
At each layer, simple features combine to form more complex features.
INPUT
Handwritten digit
28 × 28 pixels (grayscale)
The image is a grid of pixel intensities. Each pixel is a number (e.g., 0–1).
1. PIXELS
What pixels are on?
The raw input. Each pixel intensity is a number.
2. EDGES
Which pixels form edges?
Filters detect local patterns like edges, independent of their position.
3. LINES
Which edges form lines?
Edges combine to detect longer line segments.
4. LOOPS & CURVES
Which lines form loops or curves?
Lines combine into curves, corners, and closed loops.
5. PARTS
Which combinations of lines and curves form parts of digits?
Parts are detected in specific spatial configurations.
6. WHOLE DIGIT
Which parts form the complete digit?
High-level features represent entire digits.
OUTPUT
Prediction (for digits 0–9)
0 0.00
1 0.00
2 0.01
3 0.93
4 0.02
A probability distribution over all digits. The model predicts "3" because it has the highest probability.

How a Neural Network Understands Something It's Never Seen Before

A neural network learns patterns from examples, then uses those patterns to recognize new things.

1 Train the network

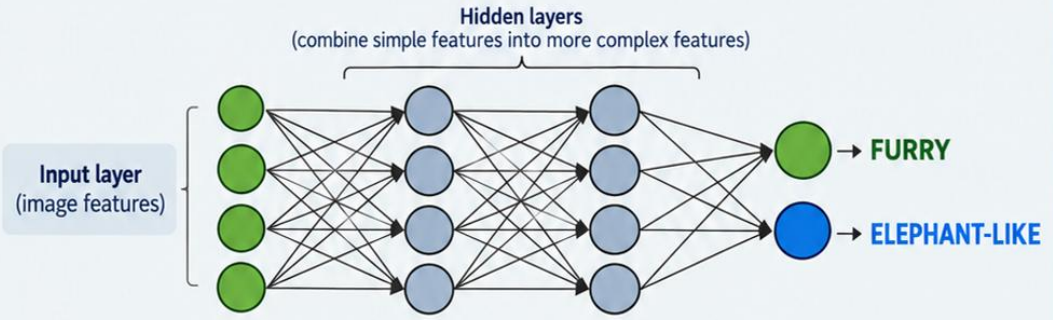
Show the network many examples of two categories:

Furry animals and **Elephants**



2 The network learns patterns

It learns what makes something furry and what makes something elephant-like.



The network builds an internal understanding of features like: fur, shape, ears, trunk, size, texture, body structure, and more.

3 See something new

Now show the network something it has never seen before:



The network recognizes:
It has **FURRY** features
and **ELEPHANT-LIKE** features.

RESULT: WOOLLY MAMMOTH
(UNDERSTOOD!)

Key idea

The network didn't memorize "woolly mammoth."
It learned the underlying patterns, so it can understand new combinations of features.

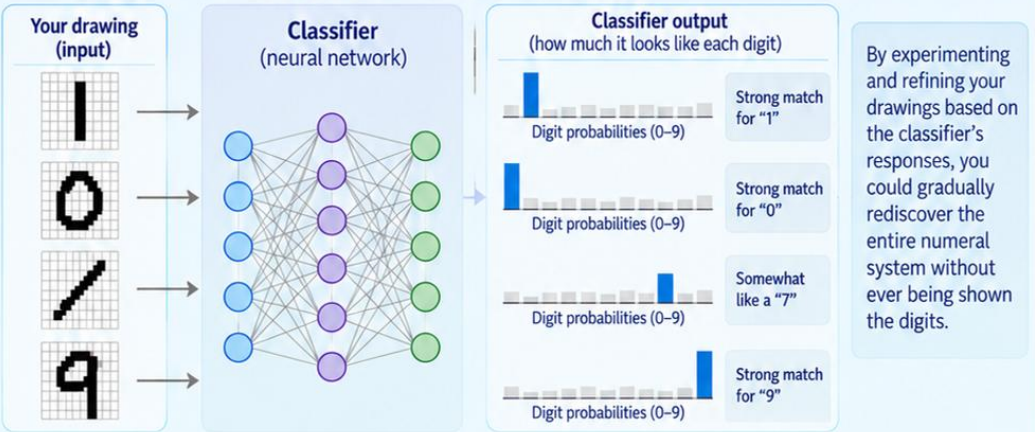
FURRY + ELEPHANT-LIKE = WOOLLY MAMMOTH

A Classifier Can Guide Creation

Even though a neural network is trained to recognize images, the knowledge it learns can also be used to create new examples.

1 Explore with a classifier

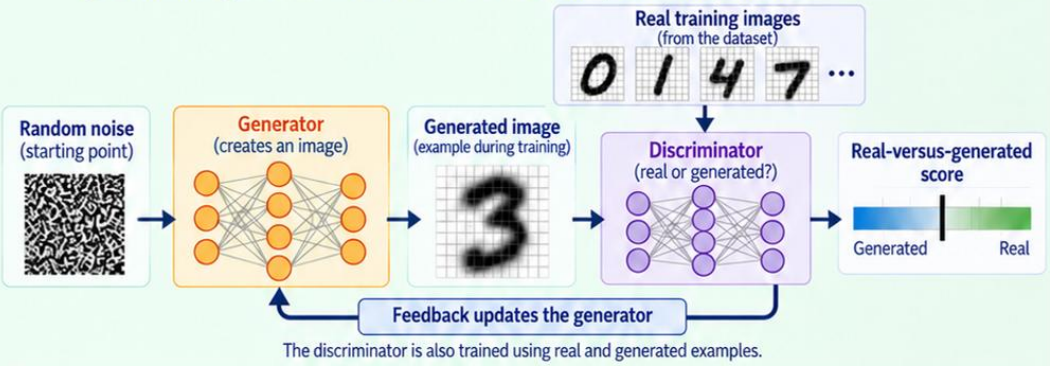
Imagine you've never seen Arabic numerals, but you can draw anything on a small pixel grid and a trained network tells you how much it looks like each digit.



By experimenting and refining your drawings based on the classifier's responses, you could gradually rediscover the entire numeral system without ever being shown the digits.

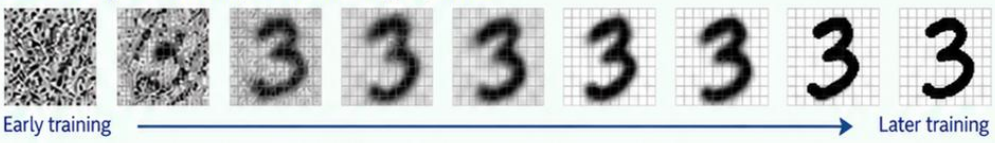
2 A Generative Adversarial Network (GAN)

A generator creates images. A discriminator learns to distinguish real training images from generated images. Feedback from the discriminator helps train the generator.



The discriminator is also trained using real and generated examples.

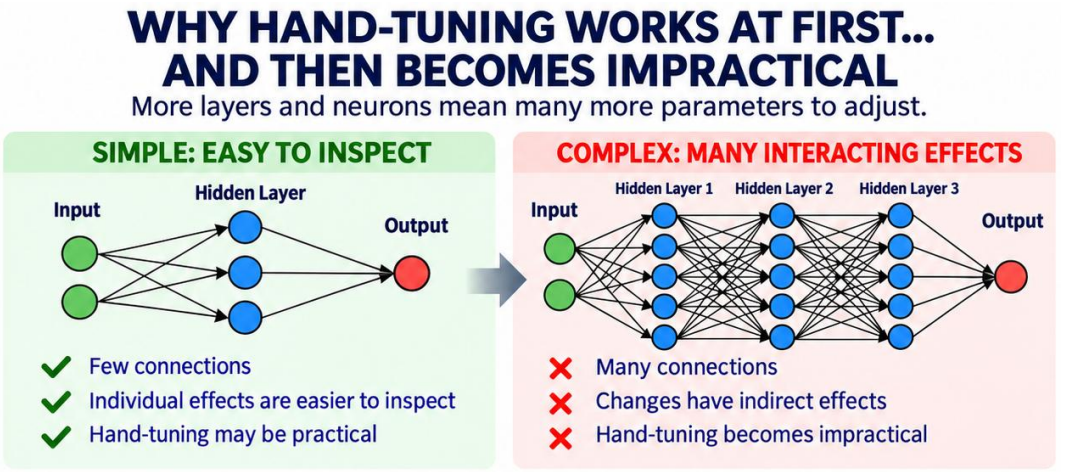
Illustrative generator outputs as training improves



3 Key takeaway

A classifier doesn't just recognize patterns — it contains enough information to help create them. Generative models, like GANs, use this idea by generating images and improving them based on the classifier's feedback.

Classification and generation are closely related.



HOW PARAMETER COUNTS GROW

Hidden layers × neurons per layer	Weights	Biases	Total parameters
1 × 3	9	4	13
2 × 3	18	7	25
2 × 5	40	11	51
2 × 10	130	21	151
3 × 10	230	31	261
3 × 20	860	61	921
3 × 100	20,300	301	20,601

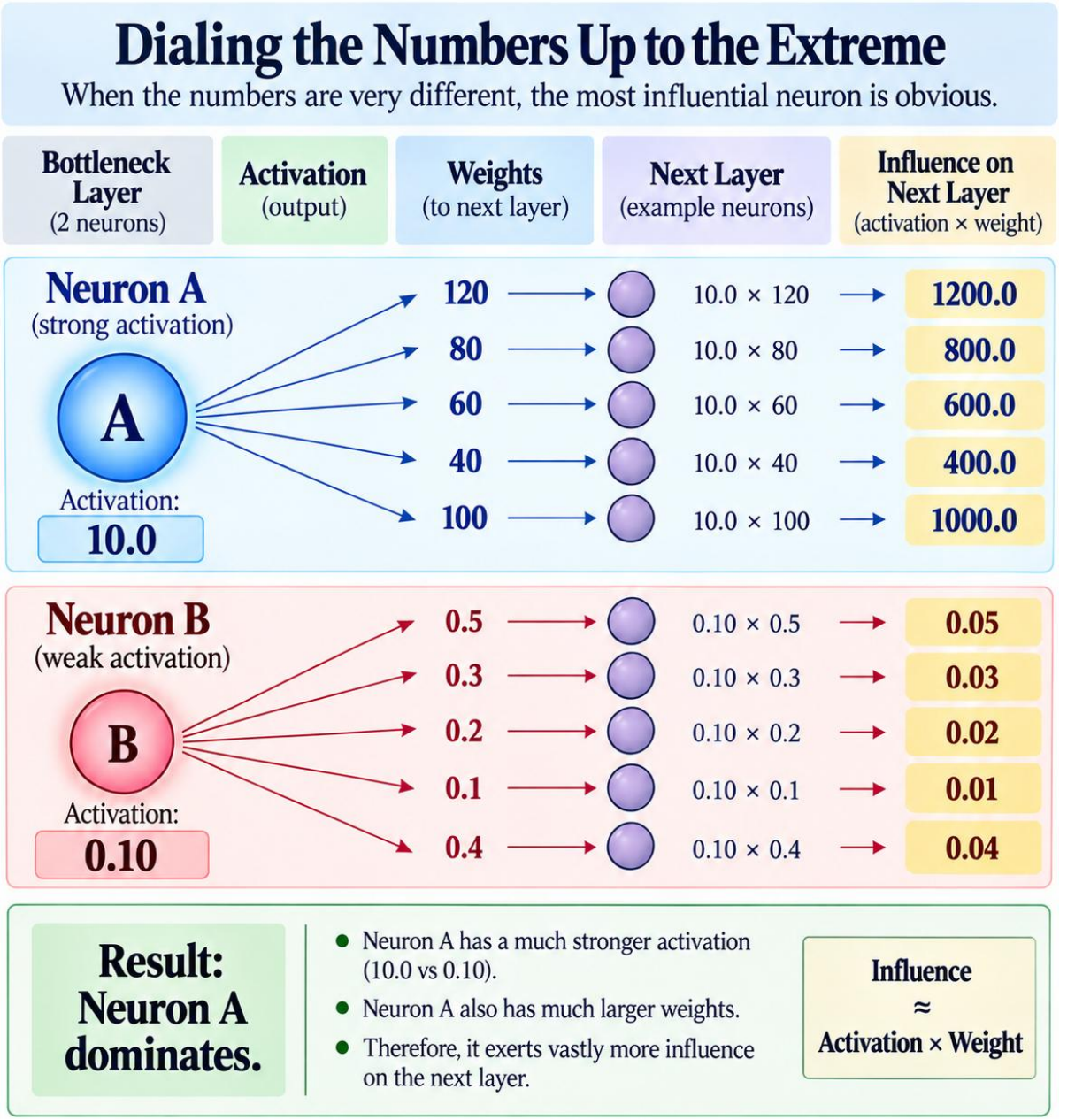
Fully connected networks with 2 inputs, 1 output, and a bias for every hidden and output neuron.

Parameter counts grow rapidly.
At fixed width, counts grow linearly with depth.
At fixed depth, the main term grows quadratically with width.

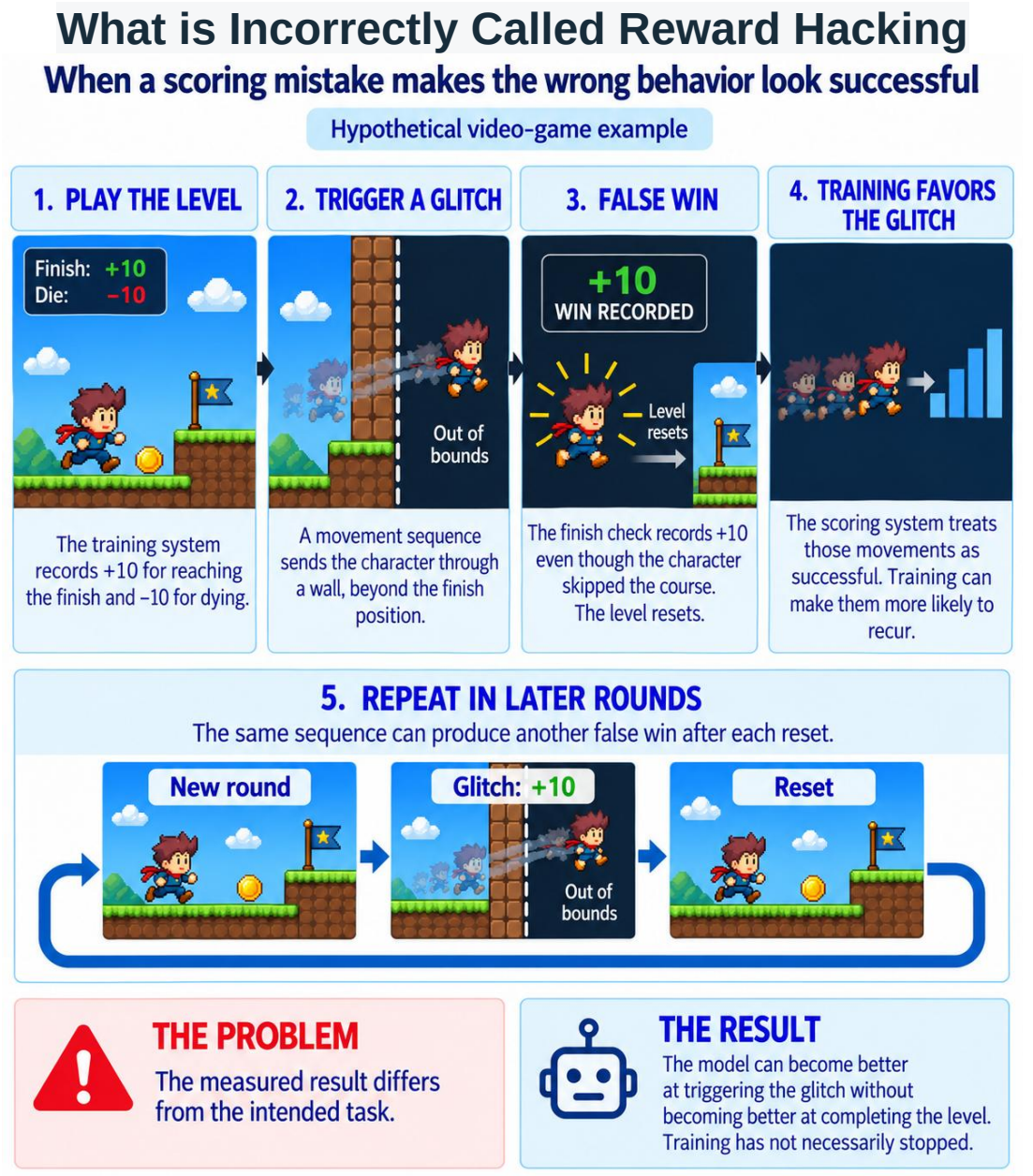
Discovering Backprop

The problem frustrated researchers for years. To see why, imagine a neural network with any number of output neurons but only two in its second-to-last layer. This bottleneck offers some insight into the nature of the problem. With only two neurons, it is not difficult to determine which neuron contributed more to the error and to optimize the weights to reduce error. This can be done by adjusting each neuron separately and determining which causes the greater change in the output.

This strategy only works if a small adjustment inside the network produces small observable changes in the output. In the earliest neural networks, neurons behaved like simple step functions: either fully activated or not activated at all. This creates a problem. If a neuron operates like a simple on/off switch, then most small changes will produce no change in output at all, but some will produce a sudden binary change. This makes it extremely difficult to measure influence in a gradual way. Adjustments usually do not produce useful changes. For this reason, modern neural networks use activation functions with useful slopes, including smooth functions and piecewise-linear functions such as ReLU.



Adjusting each neuron separately does not scale well, so the next question is whether there is some observable property of the system that reveals which neuron contributed more strongly to the final error. What we find is that the neuron's influence on the next layer is a factor of how strongly it activates and its weighted contribution. A neuron with a very strong activation but tiny outgoing weights may contribute very little to the final result. Likewise, a neuron with enormous weights but almost no activation may also have little influence. The neurons exerting the greatest influence are the neurons whose activations are being strongly amplified through their outgoing connections.



Phases of Model Training

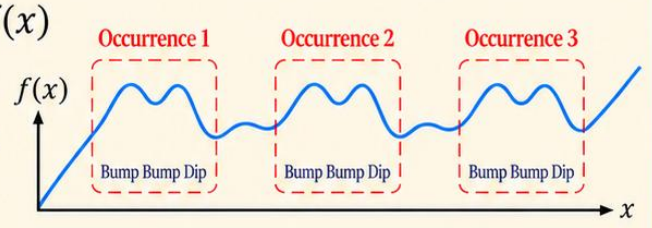
All model training is a variation on the same basic concept: expose the model to inputs and evaluate the model's output based on the loss function, then use back propagation to reduce loss, and cease training when the model meets the criteria for operation in production. Model training can be classified into two basic categories: supervised and unsupervised learning. In supervised learning, the model is provided with data that has been explicitly qualified as correct and labeled. In unsupervised learning, the model is presented with data without explicit correct answers, and attempts to discover useful relationships within the data on its own.

When a Feature Appears Multiple Times, the Network Can Reuse the Same Parameters

If the same pattern occurs in different places, gradient descent can find a single set of parameters that detects it everywhere.

1 Target Function $f(x)$

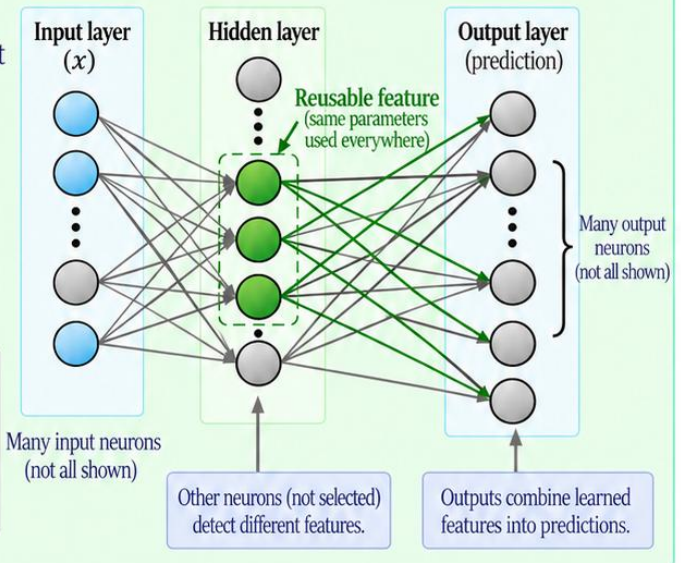
The function has the same local feature (a bump-bump-dip pattern) in multiple places.



2 Neural Network Learns a Reusable Feature

A small set of neurons learns to detect this specific pattern. The same parameters are used each time the pattern appears.

Other neurons in the network exist but are not part of this feature.

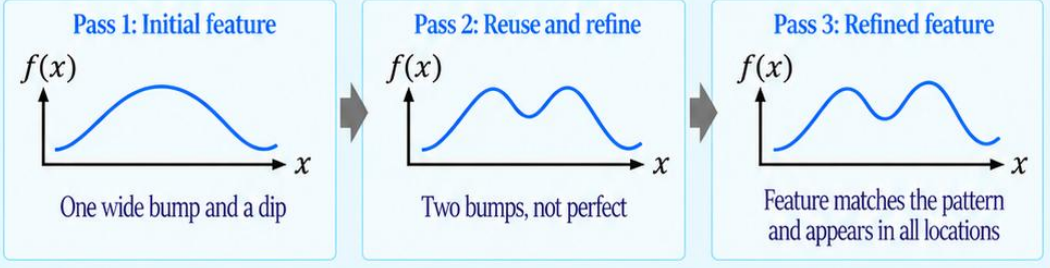


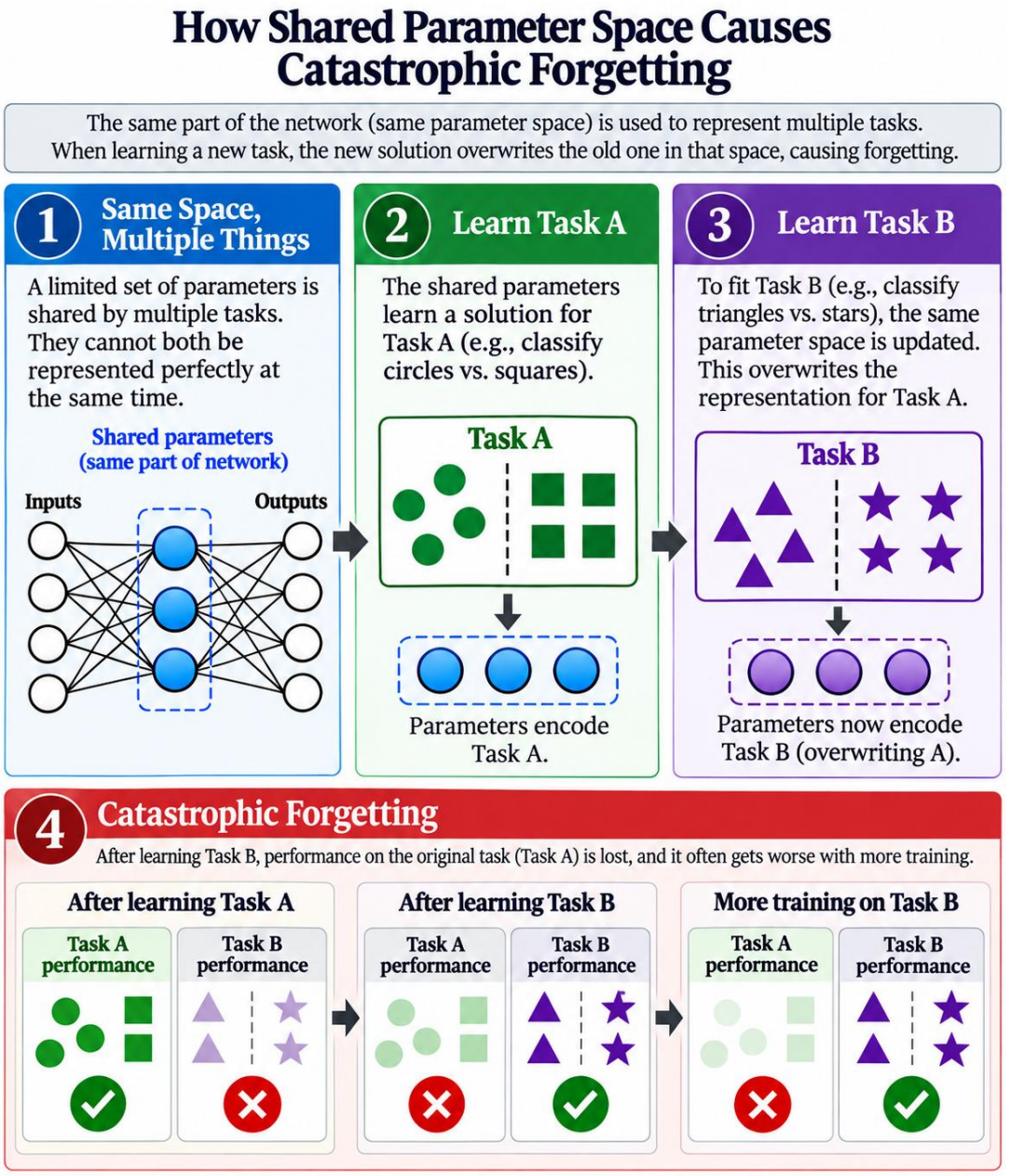
= Neurons in reusable feature (same parameters used everywhere)

= Other neurons (different features, not part of this feature)

3 The Feature Improves With Each Pass

During training, the network builds the feature, then reuses and refines it. Over time, it detects the same pattern in all locations.





Of course, the problem is not confined to language models or generative AI. It can happen to any deep learning neural network.

An innovative method of improving retraining performance while reducing the risk of catastrophic forgetting is the use of "Low-Rank Adaptation", or LoRA. A LoRA is a patch for a pretrained model that does not change existing parameters but instead adds new parameters to the model's network.[6] LoRAs have been a silent revolution in model development. They reduce the risk of catastrophic forgetting, though they do not eliminate it because it's still possible that the new parameters could interfere with old behaviors. They are also easily reversible and far more efficient to train than traditional full network retraining.

FINE-TUNING A LARGE LANGUAGE MODEL

What you actually do—from a starting model to a useful specialist.

A practical supervised fine-tuning workflow

1 DEFINE THE JOB

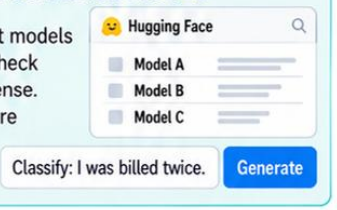
Assistant, specialist, extractor or classifier: decide what a good result looks like.

Example: sort support requests into categories.



2 CHOOSE YOUR BASE MODEL

Browse open-weight models on Hugging Face. Check task fit, size and license. Try the original before training.



3 CHOOSE HOW TO TUNE

LoRA

Train small adapters; keep base weights frozen.

QLoRA

LoRA with a quantized base to save GPU memory.

Full fine-tuning

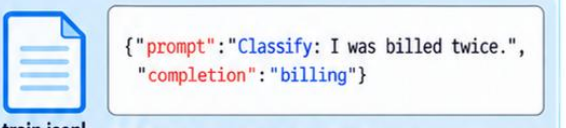
Update all model weights; usually needs more memory.



4 BUILD YOUR TRAINING EXAMPLES

Pair realistic inputs with ideal answers. An LLM can draft examples; verify and deduplicate them.

Split into training, validation and final-test sets.



JSONL: one example per line. Match your trainer's format.

5 GET GPU TIME

Rent cloud GPUs—Runpod, AWS or another provider—or use your own.

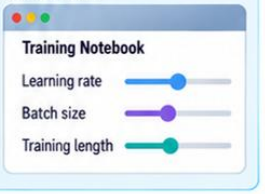
Match GPU memory to the model, training method and sequence length.



6 PREPARE THE WORKSPACE

Upload data and a training notebook. Load the model and tokenizer; check formatting.

Set learning rate, batch size and training length.



7 RUN AND WATCH

Start with a small trial.

Monitor training and validation loss, GPU memory and cost.

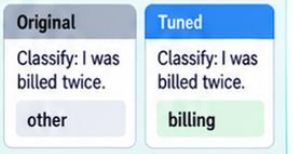
Save checkpoints as you go.



8 TRY REAL TASKS

Compare original and tuned outputs on validation examples.

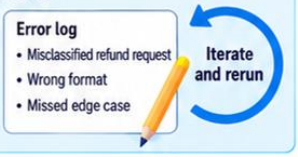
Check accuracy, format, difficult cases and retained capabilities.



9 FIX THE WEAK SPOTS

Log failures. Improve examples or settings, rerun, and compare.

Keep final-test examples out of training and tuning.



10 DECIDE WHEN IT IS READY

Confirm quality, speed and cost on the final test.

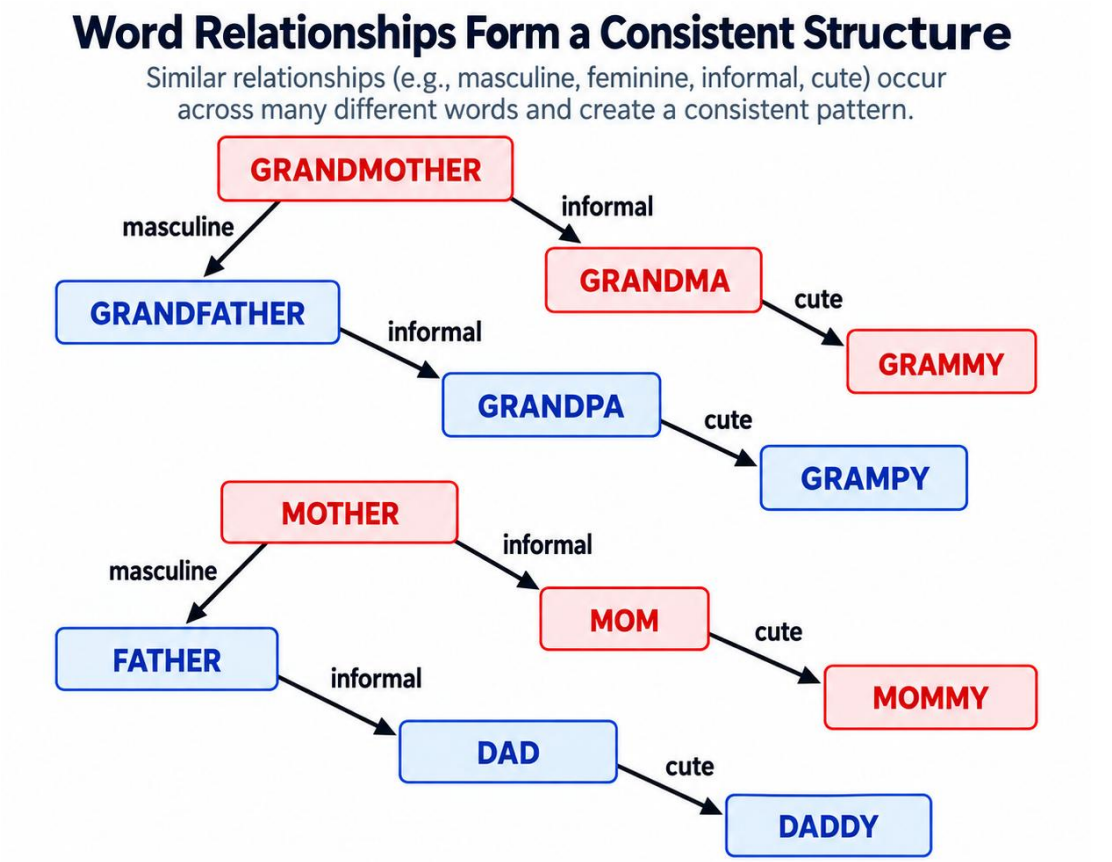
Save the model—or adapter plus base-model version and tokenizer.

Then shut down rented GPUs.



Define → Prepare → Train → Test → Improve

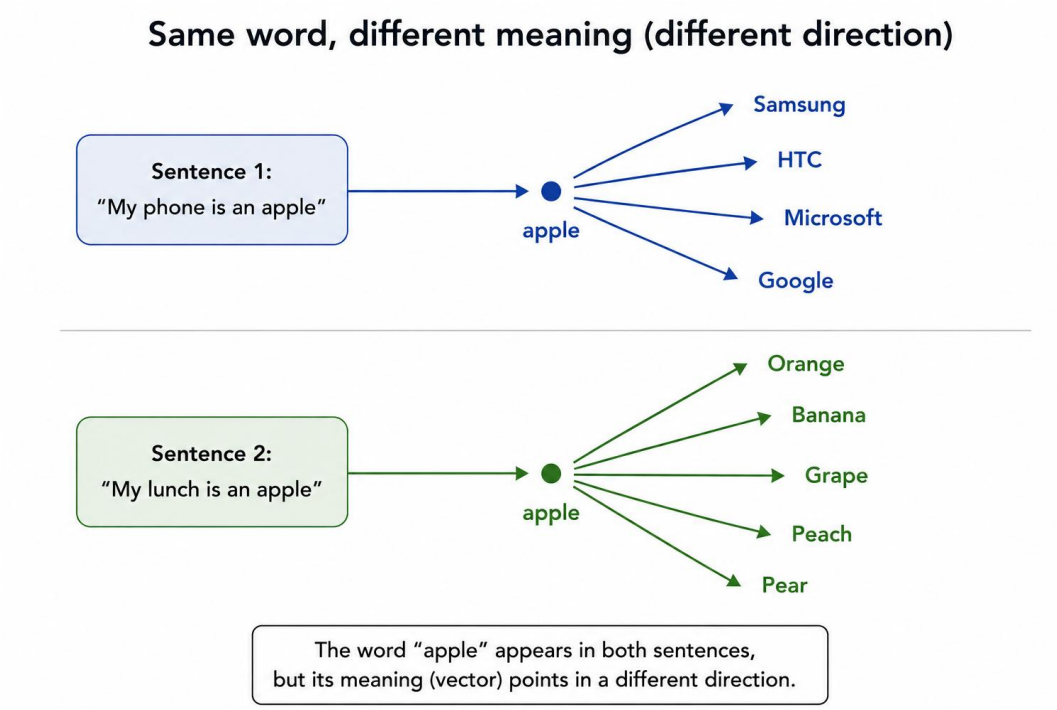
Sources: Hugging Face TRL & PEFT documentation; Runpod; AWS.



The word for mother is directly related to other words, such as mom, mamma and mommy. It is also related to the word father. What is fascinating is that word relationships can be translated into vector directions and the pathways are often consistent. The pathway from mother to mom (shortened, less formal) will similarly take you from father to dad. The path from dad to grandpa is the same as from mom to grandma and the path from grandfather to grandmother takes the same vector direction as that from father to mother. For illustration purposes, this is presented in two dimensions.

In an idealized model, when the word "very" is combined with the word "large," then the result would be similar to the word "huge." The combination of the words "not" and "good" should land close to the word "bad." In reality, it may end up close, but because these terms are not quite linguistically identical, it's not exact.

It's also not a two-dimensional space at all. It's a space of hundreds or even thousands of dimensions. What is true is that concepts and tokens often relate to each other in a consistent manner. This is key to how the network works. It means that if you can construct a sentence to take you from mom to dad, it will likely also take you from grandma to grandpa. The directions are not perfectly clean either. Moving from grandmother to grandfather does not just move you toward "masculine grandparent." It also moves you away from "bakes cookies" and toward "fishing trips."

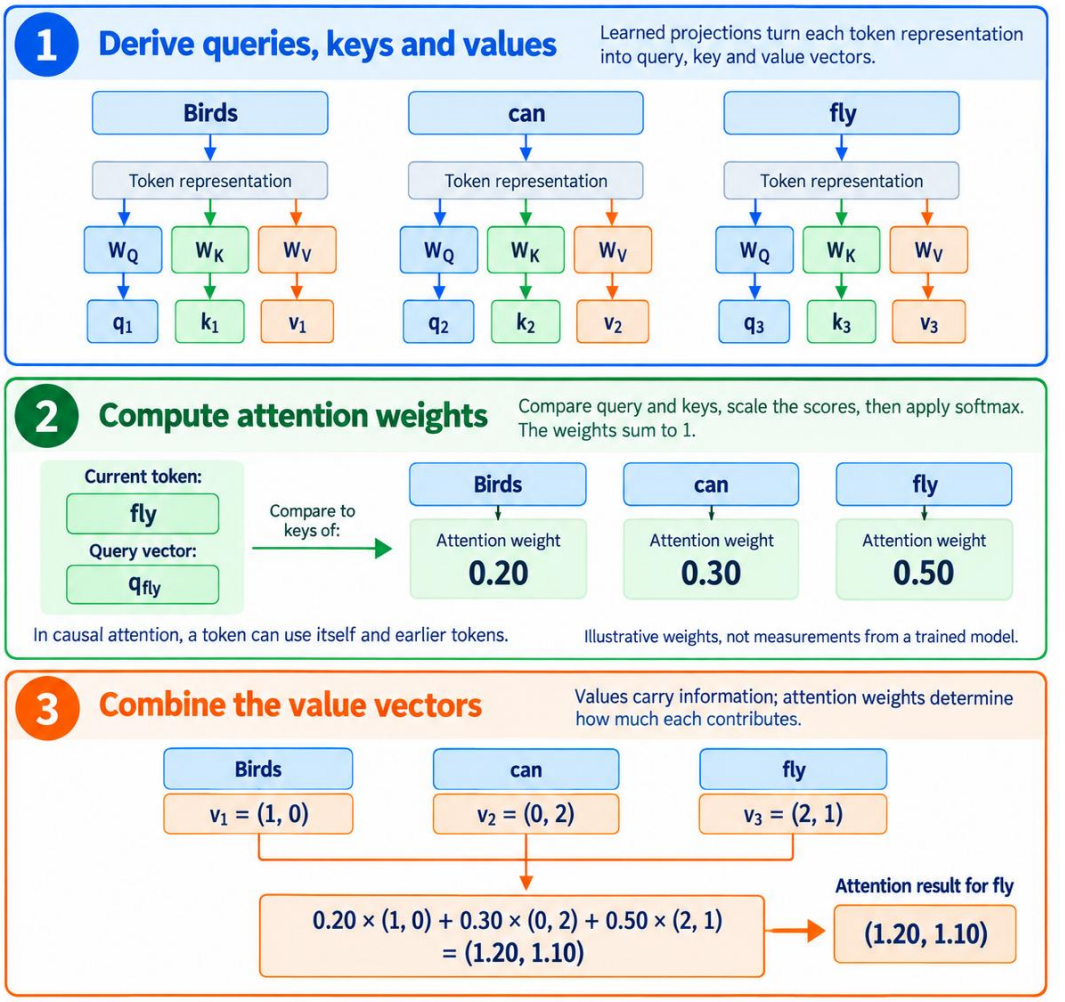


In the terminology of LLMs, the interaction of tokens is known as "attending." A token is said to "attend" to another token if it is strongly influenced by that token. The level of attention that one token gives to others is highly variable, depending on the token and context. Some tokens strongly attend to others, while other tokens may only weakly attend or barely attend to other tokens at all.

This is why the paper that introduced the transformer architecture was called "Attention Is All You Need." Although the attention mechanism had been applied to previous recurrent neural networks, the paper made the bold claim that the attention mechanism, applied to tokens, could fully replace recurrence.

Attention does more than provide the context of the token; it helps decode the entire meaning of long passages. It is far more efficient than recurrence and provides a means by which each token can be evaluated as part of a sequence. It solves the problem of long-range dependencies. In language, a single word can flip the meaning of a word many sentences away. Pronouns may refer to things stated a long distance from their occurrence. In LLMs, each token can fully attend to any token before it, and can do so without the limits of previous networks.

Transformers use something called "multi-headed attention," meaning many different attention operations occur in parallel. Each "attention head" is like a different way of looking at the relationship between tokens. One head might focus on grammatical structure, another on nearby context, another on long-range dependencies, another on tone or quotation structure. The roles of these heads are not hand designed. Their parameters are learned because they help reduce the loss for the next token.



The model also encodes each token's position, or its position relative to other tokens. This gives attention information about word order, so "dog bites man" and "man bites dog" are represented differently.

The model compares Key and Query values across tokens to get an "attention score."[8] This is used to determine how much of the value vector will be applied to the token receiving the information. This is how a model can determine that "river" moves "bank" away from "financial institution" and toward "water feature."

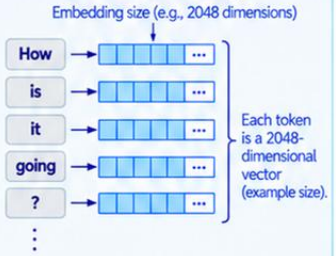
These attention scores are then passed through the softmax function, which converts the raw scores into probabilities.[9] The attention score determines how much influence a token has over another. This process is repeated over many heads and many layers, which is why tokens can become radically transformed from their original embeddings.

Inside the Attention Layer

The attention layer lets each token look at all other tokens and combine information based on learned patterns. It uses three linear projections to create queries, keys, and values.

1 Input Tokens

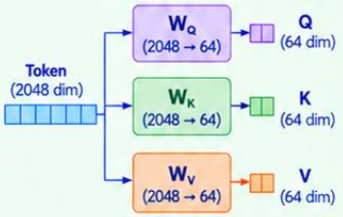
Each token is represented by a vector of numbers (an embedding).



Sequence length: 5 tokens (example)

2 Linear Projections

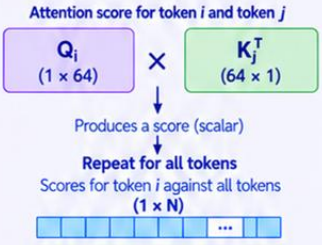
Each token is passed through three separate learned linear layers (single-layer perceptrons) to produce Query (Q), Key (K), and Value (V) vectors.



The learned weight matrices reduce the high-dimensional token embeddings to smaller vectors (e.g., 2048 → 64 dimensions).

3 Compare Queries to Keys

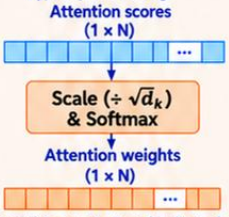
For each token, its query is compared to the keys of all tokens in the sequence using a dot product (similarity).



A higher score means the key is more similar to the query, so that token is more relevant.

4 Softmax

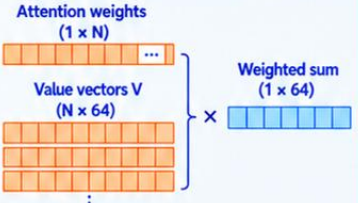
The attention scores are converted to probabilities using softmax (typically with scaling).



The weights sum to 1 and indicate how much each token's value should contribute.

5 Weighted Sum of Values

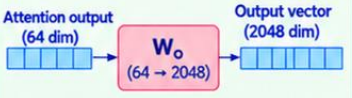
The attention weights determine how much of each token's value vector to use.



Each token *i* receives a weighted combination of the value vectors from all tokens, with higher weights for more relevant tokens.

6 Project Back to Model Space

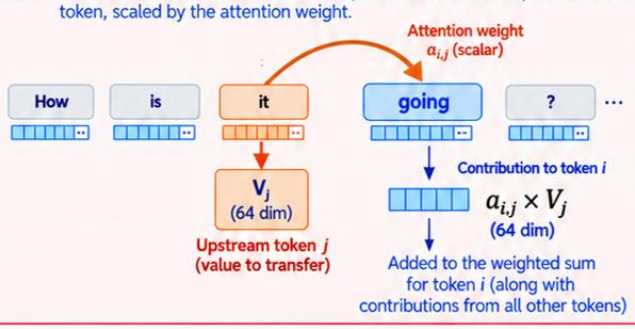
The restulting 64-dimensional attention output is projected back to the original embedding size using another learned linear layer.



This maps the attention output back to the full model dimension so it can be combined with the residual stream.

7 How One Token's Value Influences Another Token

A downstream token attends to an upstream token. The upstream token's value vector is transferred to the downstream token, scaled by the attention weight.



Effect on the downstream token

The downstream token's output is the sum of contributions from all tokens:

$$z_i = \sum_{j=1}^{N} a_{i,j} V_j$$

Attention weight (how much to use) — Value vector (what information to transfer)

A higher attention weight means more of the upstream token's information flows to the downstream token.

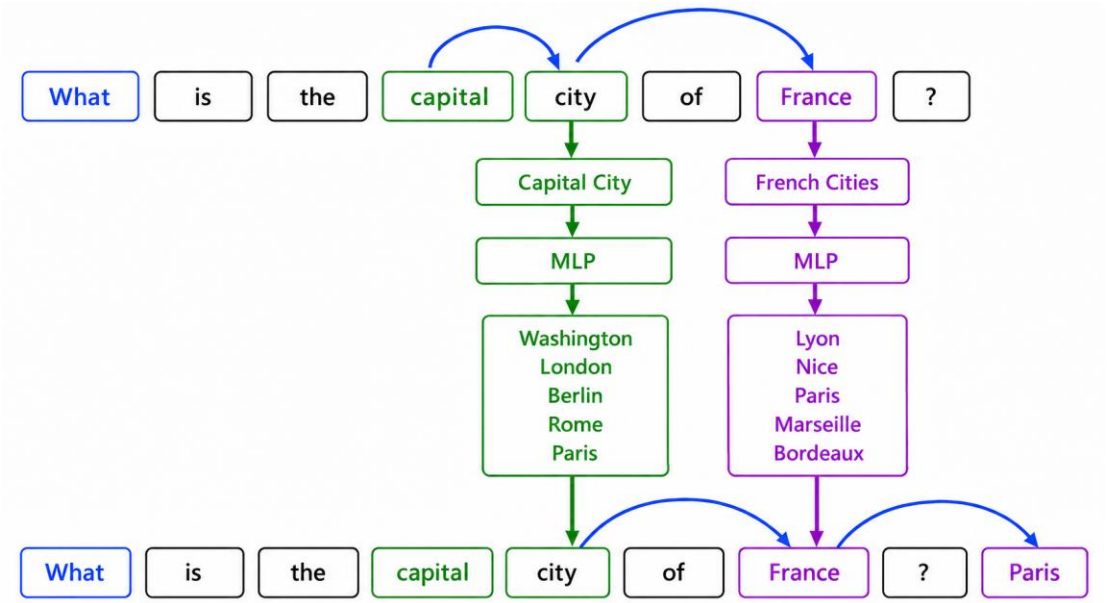
8 Multi-Head Attention (Overview)

In practice, multiple attention heads (with different learned weights) run in parallel, each producing its own output. The outputs are concatenated and projected back to the model dimension.



Multiple heads allow the model to attend to different types of relationships (e.g., syntax, coreference, facts) at the same time.

This very simplified diagram shows how attention and MLP layers interact, to both move information between tokens and transform tokens by adding additional information and features to them, therefore identifying them in context. The MLP transforms the token by adding additional features and direction.

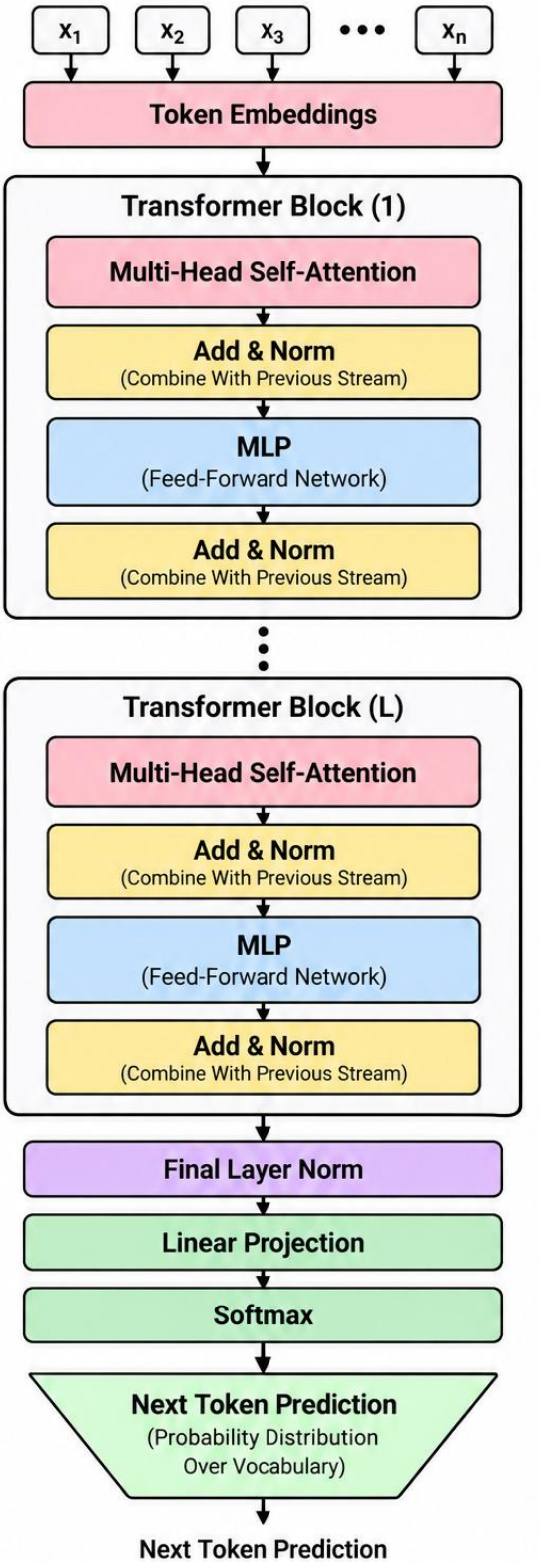


What actually is happening in this illustration is a little more subtle than it might indicate. Attention is contextualizing the word "city" and the MLP is pointing it toward capital cities. The word "France" is contextualized as being the country which a city relates to, so the MLP recognizes this and points the vector toward French geography. A subsequent attention layer then combines these vectors to arrive at Paris.

As with all things in a large language model, things are rarely this simple in reality. LLMs recreate information rather than simply storing it. Most facts and relationships don't show up cleanly in one place or another. Transformers have multiple MLP layers in them, and facts may be spread across them, as well as contained in embeddings and even within the attention matrix. Many appear to be stored redundantly, and some are assembled by attention.

The MLP's output is added to the token's incoming representation through a residual connection, while normalization keeps activations on a consistent numerical scale. In this way, information already present in the token's representation can be preserved. The vector generated by the MLP can amplify or attenuate features of that representation. It can also add new features, such as those which point toward what we would call knowledge, although to the transformer, it's merely a component in predicting the next token.

The Transformer is Composed of Alternating Attention and MLP Layers



The features of a word are not confined to its spelling. In this illustration, how a word is structured and what letters it contains are used to make the point easier to understand. Although what is and is not an Irish-sounding last name may be intuitive to a person, it can also be broken down into features of the word with weighted importance. In a language model, most features are more abstract and distributed.



ChatGPT Generated Top Ten List. No, it's not all that funny. These things are not super geniuses or anything.



10 BELIEVE IT OR NOT, HUMOR CONTAINS PATTERNS
Shocking, right? Humans have spent thousands of years making the same kinds of jokes about misunderstandings, surprises, embarrassment, and bodily functions. LLMs eventually notice.

9 THEY HAVE HEARD AN ABSURD NUMBER OF JOKES
If you read every joke book, sitcom script, comedy sketch, and internet argument ever written, you'd probably start getting pretty good at comedy too.

8 COMEDY IS OFTEN JUST WEIRD COMBINATIONS
Humor frequently comes from combining ideas that do not belong together. Language models do this all day. Sometimes it's called creativity. Sometimes it's called a catastrophic hallucination.

7 THEY UNDERSTAND EXPECTATIONS
Comedy often works by setting up an expectation and then violating it. The model's entire job is predicting expectations. Occasionally it weaponizes this ability.

6 THEY UNDERSTAND CONTEXT
The model knows that "A priest, a rabbi, and a duck walk into a bar" is probably not the beginning of a serious economic analysis.

5 THEY CAN IMITATE STYLES
Shakespeare jokes? Pirate jokes? Cybersecurity jokes? Shakespearean pirate cybersecurity jokes? The model regrets nothing.

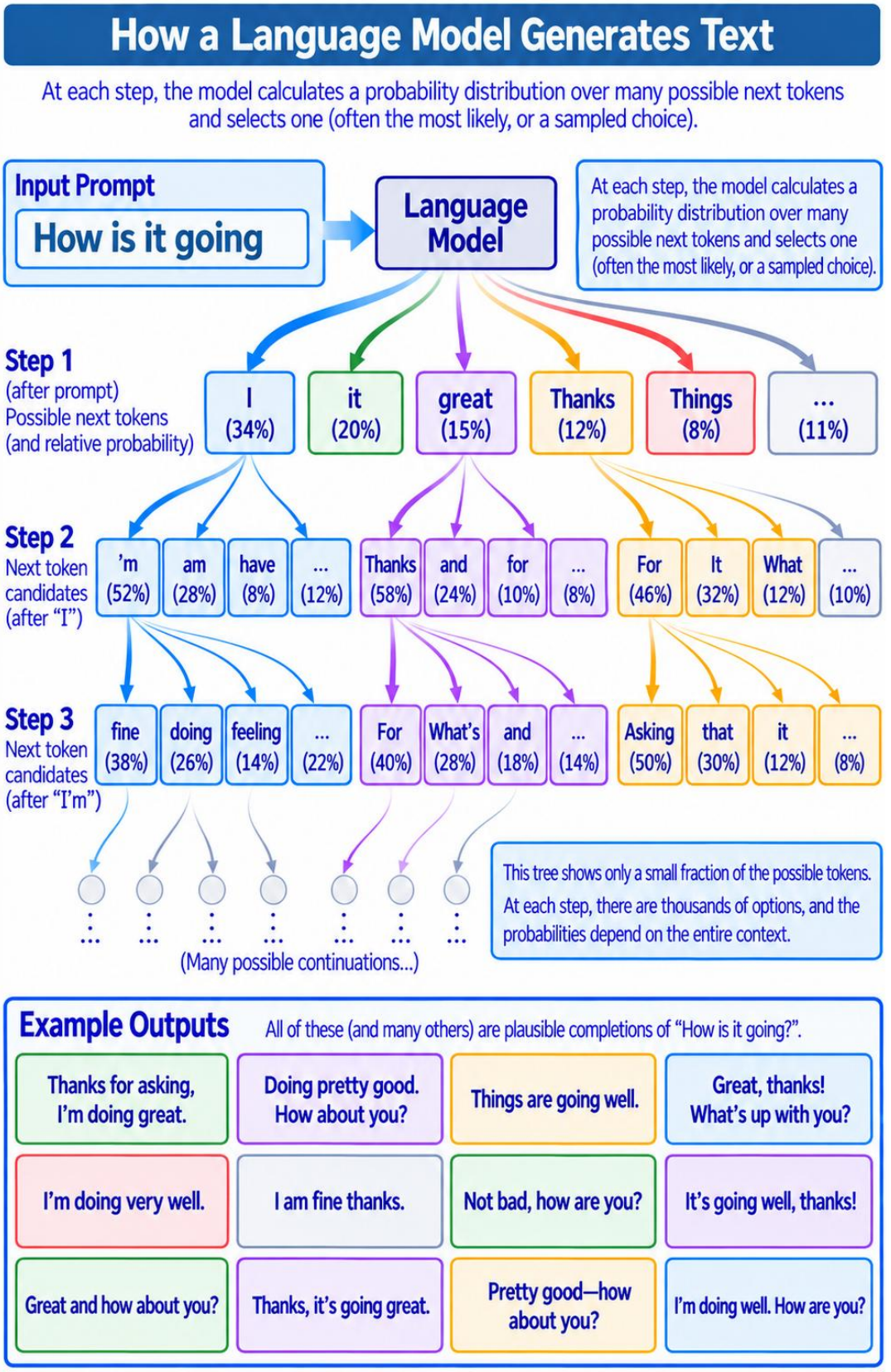
4 THEY DISCOVER JOKE STRUCTURES NOBODY PLANNED
Sometimes the funniest jokes are accidents. The model is extremely good at accidents.

3 THEY NEVER GET TIRED
Human comedians eventually run out of ideas. A language model can generate seventeen thousand jokes about malfunctioning printers without experiencing the slightest concern for its own dignity.

2 THEY CAN STACK MULTIPLE TYPES OF HUMOR
Wordplay. Irony. Absurdity. References. Callbacks. Complete nonsense. Sometimes all in the same joke, much like a committee meeting.

1 THIS JOKE COULD NOT POSSIBLY BE MORE META

- The language model generated a joke explaining how language models generate jokes.
- You are reading it in a book about language models.
- The model knows that you know that it knows that you know that it generated the joke.
- At this point the joke is no longer a joke.
- It has become an interpretability problem.

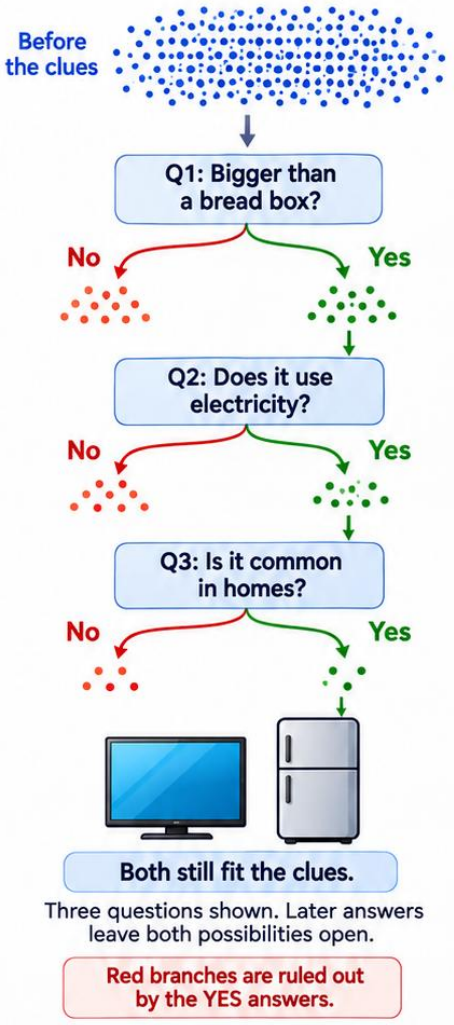

How a Language Model Generates Text
At each step, the model calculates a probability distribution over many possible next tokens and selects one (often the most likely, or a sampled choice).
Input Prompt
How is it going
Language Model
At each step, the model calculates a probability distribution over many possible next tokens and selects one (often the most likely, or a sampled choice).
Step 1
(after prompt)
Possible next tokens
(and relative probability)
I (34%)
it (20%)
great (15%)
Thanks (12%)
Things (8%)
… (11%)
Step 2
Next token candidates
(after "I")
'm (52%)
am (28%)
have (8%)
… (12%)
Thanks (58%)
and (24%)
for (10%)
… (8%)
For (46%)
It (32%)
What (12%)
… (10%)
Step 3
Next token candidates
(after "I'm")
fine (38%)
doing (26%)
feeling (14%)
… (22%)
For (40%)
What's (28%)
and (18%)
… (14%)
Asking (50%)
that (30%)
it (12%)
… (8%)
(Many possible continuations…)
This tree shows only a small fraction of the possible tokens.
At each step, there are thousands of options, and the probabilities depend on the entire context.
Example Outputs
All of these (and many others) are plausible completions of "How is it going?".
Thanks for asking, I'm doing great.
Doing pretty good. How about you?
Things are going well.
Great, thanks! What's up with you?
I'm doing very well.
I am fine thanks.
Not bad, how are you?
It's going well, thanks!
Great and how about you?
Thanks, it's going great.
Pretty good—how about you?
I'm doing well. How are you?

PLAYING 20 QUESTIONS WITH A LARGE LANGUAGE MODEL

Each reply uses the conversation so far.
Earlier answers constrain what can come next.

A NARROWING FIELD OF POSSIBILITIES

Many objects could fit at first.



Three questions shown. Later answers leave both possibilities open.

Red branches are ruled out by the YES answers.

ONE GUESS, TWO POSSIBLE ENDINGS

The reply follows the stated constraints.

You: Is it a sock?

LLM: No. I said it was bigger than a bread box.

Respects the earlier answer.

THE FINAL GUESS

You: Is it a television?

RIGHT

LLM: Yes! You got it!



OR

WRONG GUESS — CONSISTENT REVEAL

LLM: No. You're out of questions! It was a refrigerator.



Different ending. Same constraints.

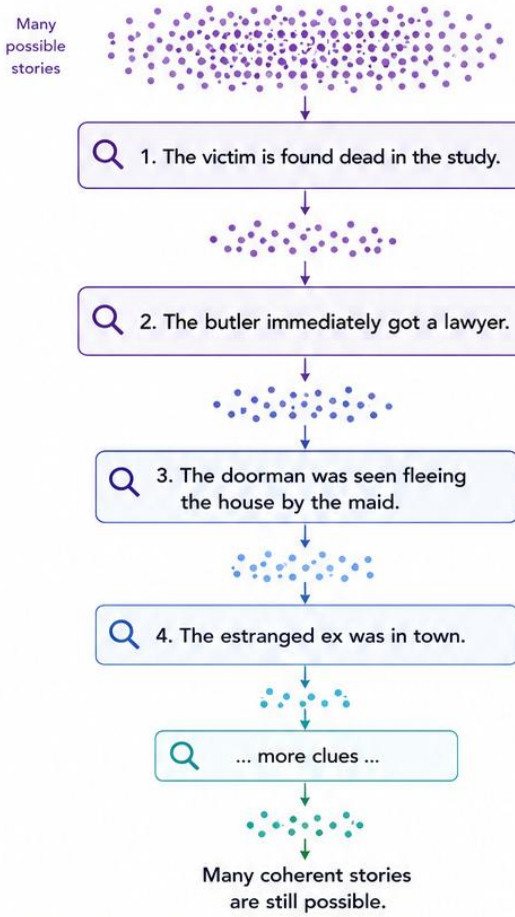
CONSISTENCY DOES NOT PROVE A FIXED SECRET

A coherent exchange can emerge from the converstation's constraints without a hidden object being fixed in advance.

More than one ending can remain plausible.

An LLM Written Murder Mystery: Collapsing Possibilities

An LLM generates one token at a time, choosing from a probability distribution.
Each new clue collapses the space of possibilities—but many coherent stories remain.



What's happening?

Each clue rules out some stories and strengthens others.

The model doesn't "know" the truth—it just follows patterns to build plausible narratives.

- Initial state: thousands of possible mysteries.
- After clue 1: many options eliminated.
- After clue 2: fewer remain.
- After clue 3: narrow further.
- After clue 4 and more: still several strong stories left.

All of these endings are plausible.

The model might land on any of them—randomly.

Ending A

The butler did it. That's why he got a lawyer. The doorman is a red herring.

Ending B

The lawyer is a red herring. The doorman did it, and that's why the maid saw him. The ex was another red herring.

Ending C

The maid did it. She lied about the doorman, because she was in cahoots with the ex.

Ending D

The ex did it. The doorman was her lover and was in on the plot.

Ending E (PLOT TWIST)

The ex wanted to patch things up, but the maid didn't want it and blamed the doorman.

The model has no idea which one is "true."

It just picks a path and writes the most coherent story it can from that point forward.

NEURAL SCALING LAWS

Why language models improve with scale

In LLMs, increasing model capacity, useful training data and compute often reduces prediction loss in regular, measurable ways. These empirical relationships are called scaling laws.

Why scale helps

More capacity

Additional parameters can represent finer distinctions and more complex combinations of features.

More training

More compute allows further optimization: repeated gradient updates refine the model's predictions.

Broader experience

Diverse, relevant examples fill gaps, challenge brittle shortcuts and support reusable patterns.

More examples can challenge a brittle rule

Narrow examples

The **key** **is** missing.

The **keys** **are** missing.

A shortcut also fits: match the nearest noun.

Counterexamples

subject → *verb*

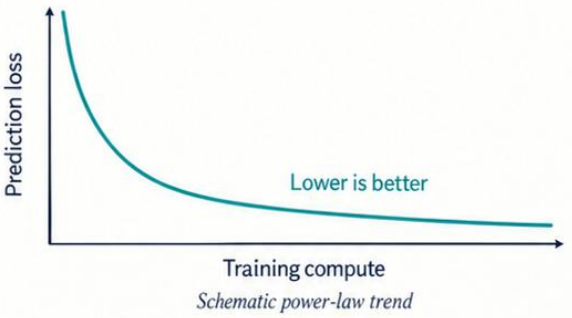
The **keys** to the cabinet **are** missing.

The **key** to the cabinets **is** missing.

The nearest-noun shortcut fails. Agreement with the subject works across all four examples.

Illustrative example; broader data supports generalization but does not guarantee it.

The evidence is about prediction



Schematic power-law trend

Under studied conditions, loss often follows a power law. Further reductions generally require multiplying resources.

Training balance matters

70B vs 280B

parameters

Chinchilla (2022) outperformed Gopher across many evaluations with the same training compute and about four times the training data.

Hoffmann et al., 2022

What scaling does not establish

01 Quality still matters

More tokens cannot guarantee useful information. Data, compute costs and architecture constrain progress.

02 A test is not every task

Lower text-prediction loss does not establish reliable performance in arbitrary, open-ended situations.

03 Capability is not consciousness

Scaling results do not establish awareness, autonomous motives or unlimited intelligence.

Remarkable gains are real. Unlimited claims go beyond the evidence.

Sources: *Kaplan et al. (2020); Hoffmann et al. (2022); Tu et al. (2020); Muennighoff et al. (2023).*

What Is a Reasoning Model?

Adding reasoning capabilities to large language models.

A reasoning model uses extra computation to think through a problem, rather than just predicting the next token. It generates intermediate steps, considers multiple options, and evaluates them before giving a final answer.

Traditional LLM

- Predicts the next token based on patterns in training data.
- Responds directly, with no explicit thinking.
- No intermediate steps shown to the user.
- Works well for many tasks, but can struggle with complex reasoning, multi-step problems, and novel situations.

Fast, fluent, and useful — but limited in deeper reasoning.

Add reasoning capabilities

Reasoning-Enhanced LLM

- Generates intermediate steps, plans, and evaluates before producing the final answer.
- Uses additional compute during generation to "think" (e.g., chain-of-thought, tree search, or other methods).
- Can revise its own reasoning and backtrack if needed.
- Often shows its reasoning steps (sometimes hidden).
- More capable on complex problems involving math, logic, planning, and tool use.

Same model architecture, but with more computation and reasoning.

Why Add Reasoning?

- ✓ Handles harder problems that require multiple steps.
- ✓ Breaks complex tasks into manageable steps.
- ✓ Self-checks and corrects mistakes.
- ✓ More reliable and capable on complex queries (math, logic, planning, coding).
- ✓ Better follows instructions for structured tasks.
- ✓ Enables new capabilities like planning, tool use, and multi-step problem solving.
- ✓ Produces more accurate and trustworthy answers.

Reasoning turns a good language model into a more capable and reliable assistant.

Where Reasoning Happens

1. Behind-the-Scenes (Implicit) Reasoning

The model thinks, but you don't see it.

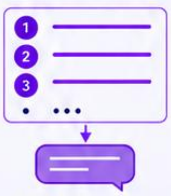


Reasoning tokens are hidden. You just see the answer.

Common in many production models.

2. Explicit (Visible) Reasoning

The model shows its step-by-step thinking.



Useful for learning, debugging, and trust.

You can see the model's reasoning process (e.g., chain-of-thought).

3. Controlled Disclosure (Hybrid)

Some reasoning is shown; some is kept hidden.



Companies decide what the user should—or shouldn't—see.

Balances transparency and model performance.

4. Pre-Response (Planning) Reasoning

The model reasons and plans before it starts speaking.



Builds a plan, picks the best path, then produces the response.

Helps with complex multi-step tasks.

5. Latent-Space Reasoning

Reasoning occurs in the model's hidden activations (no tokens).



The "thinking" is in vectors, not in words.

Happens entirely inside the model.

How a Reasoning Model Works (High Level)



1. User Question
A complex or uncertain task.
Provides the prompt or question.

2. Internal Reasoning
Model explores options, checks facts, and plans.
May generate many tokens (visible or hidden).

3. Evaluate & Select
Chooses the best answer based on evidence and self-checks.
Can revise or correct its own reasoning.

4. Final Response
Outputs the answer (visible to the user).
A more reliable and well-supported response.

Key Takeaway

Reasoning models still predict tokens—one at a time.

What's new is that they allocate some of those tokens (or hidden computation) to think before committing to an answer.

The same core mechanism. More computation. Better results.

Example: Solving a Multi-Step Problem

Question:
A store sells 3 notebooks for $4.50. How much do 7 notebooks cost?

1. Understand
Figure out what's being asked.
Find the price per notebook, then multiply by 7.

2. Plan
Steps:
1. $4.50 ÷ 3 = $1.50
2. $1.50 × 7 = $10.50

3. Check
Does the answer make sense?
Yes. 7 is a little more than twice 3, so the cost should be a little more than twice $4.50.

4. Answer
The cost is **$10.50.**

Same architecture. More thinking. Greater capability.

RAG Optimization Strategies

Make your retrieval smarter, faster, and more relevant.

1. PREPARE KNOWLEDGE

Turn raw content into a searchable knowledge base.



Raw Documents
PDFs, docs, web, databases, etc.

→



Chunk
Split into manageable passages

→



Embed
Convert chunks to vector representations

→



Index & Store
Store in vector database for fast search

2. RETRIEVE

Find the most relevant information for the query.



Search
Find the most relevant chunks for the query

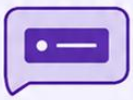
→



Rank & Filter
Re-rank, filter duplicate or low-quality results

3. GENERATE

Use the retrieved context to produce a better answer.



Augment & Generate
Provide context to LLM and generate accurate, grounded answers

Key RAG Optimization Strategies

Smart Chunking



Split content in ways that preserve meaning and context.

Why it helps

- Semantic chunking
- Overlap chunks
- Adaptive chunk sizes

Pre-Embed



Embed documents beforehand and store for reuse.

Why it helps

- Faster queries and lower latency at inference time
- Batch embedding
- Update incrementally
- Reuse across apps

Better Indexing



Use the right index and structure for your data.

Why it helps

- Higher recall, faster search, and scalability
- HNSW / IVF / PQ
- Metadata filtering
- Hybrid (vector + keyword)

Knowledge Map



Organize content as a graph of concepts and relationships.

Why it helps

- Improves multi-hop reasoning and deeper insights
- Build entity/relation graphs
- Link to source chunks
- Traverse for richer context

Caching



Cache frequent queries, embeddings, and retrieval results.

Why it helps

- Reduces latency and compute costs
- Query result cache
- Embedding cache
- TTL and invalidation

Re-Ranking



Re-rank initial results with cross-encoders or LLMs.

Why it helps

- Delivers the most relevant content to the model
- Cross-encoder rerankers
- LLM-based reranking
- Diversity-aware ranking

Additional Best Practices



Data Quality
Clean, deduplicate, and keep content up to date.



Metadata Matters
Add metadata and source info for better filtering and trust.



Evaluate & Monitor
Track retrieval quality, answer accuracy, and user feedback.



Iterate Continuously
Optimize chunking, indexing, and prompts based on real usage.



Goal: Retrieve the right context, every time – so the model can generate with confidence.

How a Diffusion Model Works

It starts with random noise and gradually denoises it step by step until it matches the prompt.

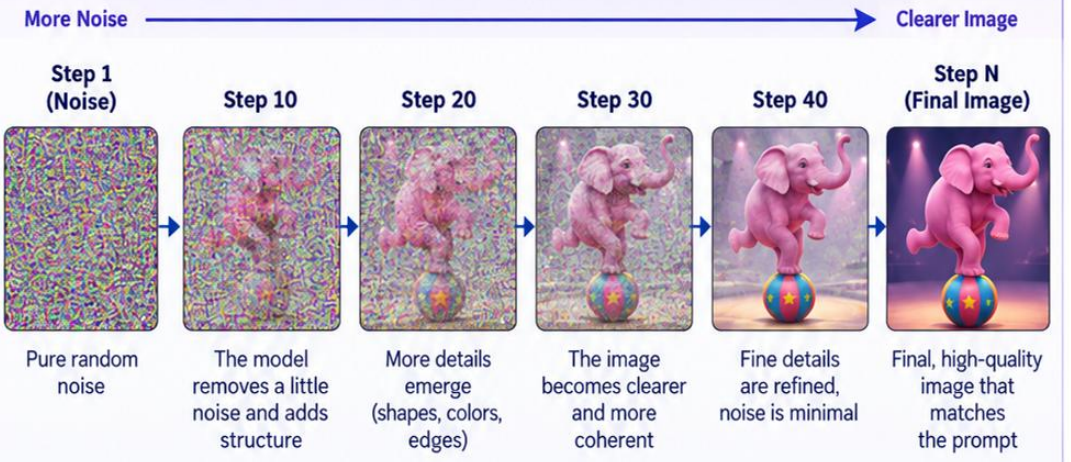
1 Prompt

"A pink elephant dancing on a ball"

The text is converted into a numerical embedding that conditions the image generation.

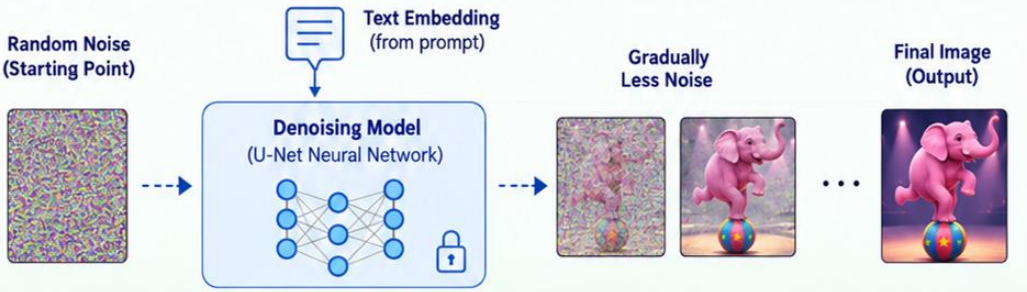
2 Denoising Process (Many Steps)

The model starts with pure random noise and gradually removes noise, using the prompt as guidance, until a clear image emerges.



3 What's Happening?

A diffusion model is trained to reverse the noising process. At each step, it predicts the noise and removes a little more, guided by the text prompt.



Key Idea: The model learns what an image should look like by seeing millions of examples. It uses that knowledge to turn noise into something that matches your prompt.

From Images to 3D Worlds

How NeRFs, Gaussian Splats, and Generative AI Turn Pictures into 3D Environments

Multiple views, learned representations, and generative models can create rich, navigable 3D worlds from images.

1 Capture or Collect Images

Multiple photos (or video frames) show the same scene from different angles.

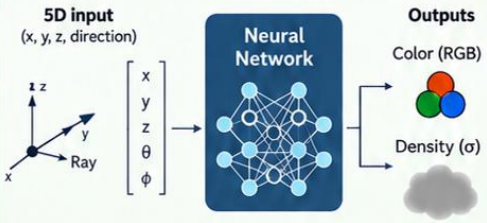


Many overlapping views
(photos, video, or internet images)

Key idea:
Each image provides a different viewpoint. Overlapping views let the system figure out the 3D structure through geometry and learning.

2 NeRF: A Continuous 3D Scene

Neural Radiance Fields (NeRFs) learn a function that predicts color and density at any 3D location.



The network is trained so that rendered images match the input photos. It learns a smooth, continuous 3D representation of the scene.

Result: Render from any viewpoint

After training, you can place a virtual camera anywhere and render a new view.



3 Gaussian Splats: A Fast 3D Repressentation

Instead of a neural network, the scene is represented as millions of 3D Gaussian blobs ("splats").



Each Gaussian has:
- 3D position
- Size and shape
- Color
- Opacity (alpha)

During rendering, the splats are projected onto the image plane and blended together in real time. It's faster to train and render than NeRFs, and produces high-quality results.

Result: Real-time novel views

Gaussian splats enable high-quality, fast rendering of new viewpoints.



4 Generative AI: Create 3D from Few or Single Images

Recent models can generate complete 3D scenes from limited input — even a single image — by combining 3D reconstruction techniques with powerful generative priors.

Input
One or a few images
(e.g., a photo or sketch)



Generative 3D Model

- Reconstructs geometry
- Fills in missing views
- Generates plausible content
- Outputs a 3D representation (NeRF, Gaussian splats, or mesh)

3D Scene (Generated)



Uses generative priors to infer:
- Hidden geometry (back sides, interiors)
- Realistic textures and lighting
- Consistent structure and layout
- Plausible completions for unseen areas

Works with:
- Photos
- Text descriptions (text-to-3D)
- Sketches or concept art
- Partial or low-quality images

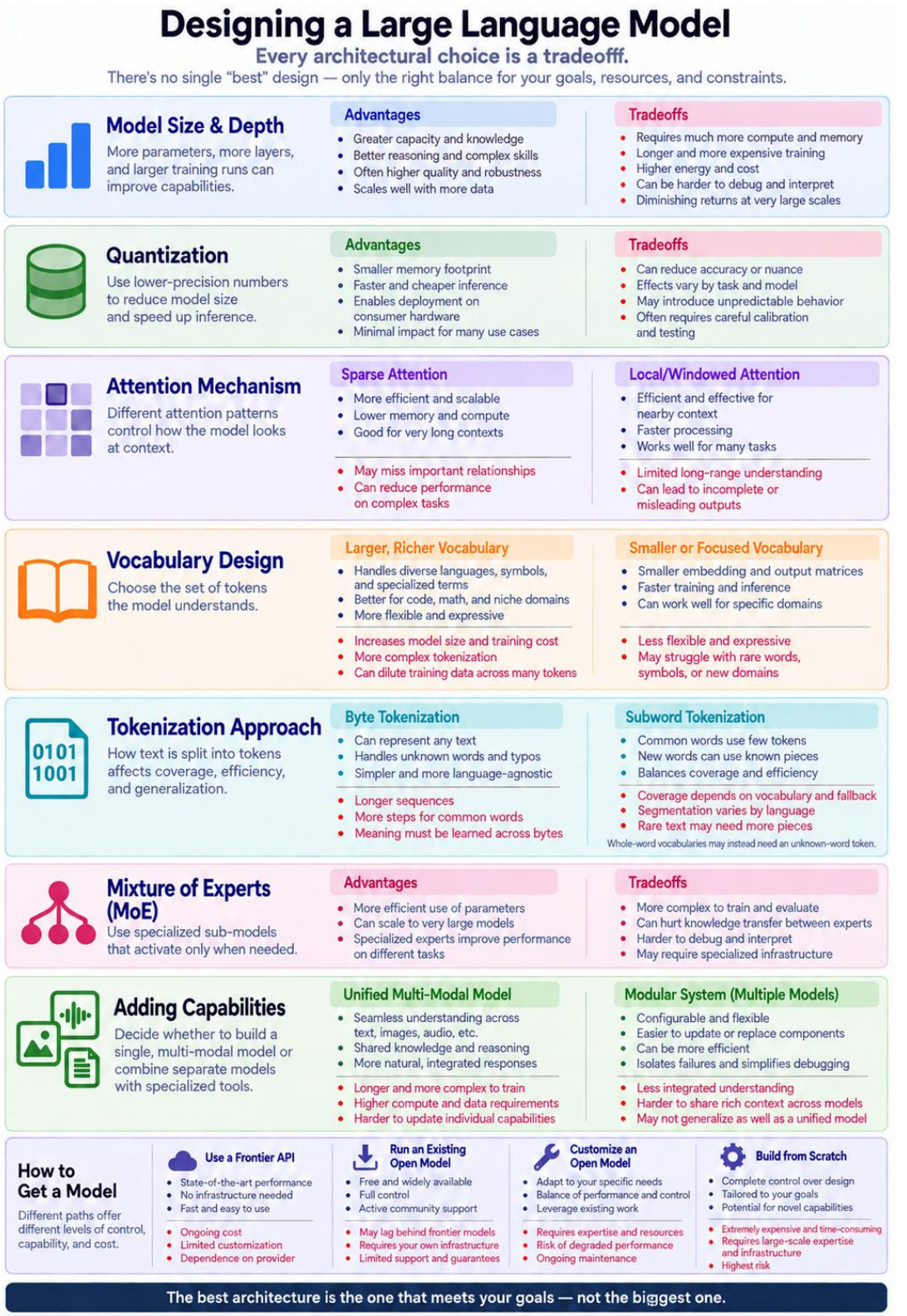
5 Applications

Once a 3D scene is created, it can be explored, edited, or used in many ways.



Virtual walkthroughs (AR/VR, real estate, gaming) | 3D editing and content creation (add, remove, modify) | World generation (from text or images) | Simulation and robotics (navigation, training) | 3D asset generation (for film, games, design)

REAL IMAGES. LEARNED REPRESENTATIONS. NEW WORLDS. From photos to possibility.



Although language models are traditionally built on the transformer architecture, there are alternatives being investigated. For example, state space models use a single vector to build context for token prediction.

THE CHALLENGE OF AUTONOMOUS VEHICLES

99% OF DRIVING CAN BE AUTOMATED.

1% IS STILL DAMN NEAR IMPOSSIBLE.

BRILLIANT MOST OF THE TIME. BUT THAT LAST 1% IS A WHOLE DIFFERENT WORLD.



WHY IS THAT 1% SO HARD?

Because the world isn't a simulation. It's a wild, unpredictable, human place.



A WILD WORLD

From animals and debris to weather and weird infrastructure, the real world is full of the unexpected.



RAPIDLY CHANGING CONDITIONS

Weather, lighting, road layouts, construction, emergencies — conditions can change in an instant.



ADVERSARIAL HUMANS

People don't always follow rules. Some take advantage, cause mischief, play games, or just have road rage.



PEOPLE MAY PLAY TRICKS

Some might deliberately test, confuse, or prank a self-driving car — for laughs, attention, or worse.



TRULY NOVEL SITUATIONS

No matter how much data it sees, a self-driving car will eventually encounter something it's never experienced before.



NO DECISION OWNERSHIP

When something goes wrong, who's responsible? The car? The manufacturer? The software? The owner? The law isn't clear.



INVITES THEFT AND MISUSE

A vehicle traveling alone is a tempting target — for theft, vandalism, or repurposing.



HARD TO FIT IN TODAY'S LAWS

Existing traffic laws were written for human drivers. Self-driving vehicles challenge everything from liability to what "due care" means.

Extraordinary progress. Extraordinary challenges.

Autonomous vehicles can already do 99% of the driving. But to earn our trust, they must master the 1% — and that 1% includes all the messy, unpredictable, fascinating, frustrating things that make the real world, well ... real.





There has been a great deal of talk about policies to address AI-related job loss. Unfortunately, large scale layoffs get all the attention, because they are newsworthy events and easy to measure. However, the most vulnerable to labor market shifts are rarely the ones society thinks of helping.

HOW PCI DSS MAKES PAYMENT SECURITY WORK

Common standards. Clear responsibilities. Controls that stay in place.

PCI DSS turns payment-data protection into an everyday business obligation: shared requirements, documented controls and recurring checks.

2004: First PCI DSS standard | **2006:** PCI Security Standards Council

1 A CONVENTIONAL RISK-CONTROL SYSTEM

Set requirements → Implement controls → Check compliance → Fix and repeat

Card brands and acquiring banks manage compliance programs. PCI SSC develops standards and qualifies assessors. [1]

2 12 REQUIREMENTS. PRACTICAL DEFENSES.

- **Secure systems** — Harden settings and control network access.
- **Protect card data** — Minimize storage and protect stored and transmitted data.
- **Maintain software** — Patch vulnerabilities and defend against malware.
- **Restrict access** — Limit privileges and verify identities.
- **Watch and test** — Review logs, scan systems and test defenses.
- **Assign responsibility** — Train staff, manage providers and prepare for incidents.

3 STANDARDIZED SECURITY AT THE CHECKOUT

- **PCI DSS** — Secure operation and inspection
- **PCI P2PE** — Validated encryption solutions



- **PCI PTS** — Lab-tested device security
- **EMV** — Chip authentication and interoperability

Common requirements and recognized testing let different manufacturers build to shared standards. [1,2]

4 EVIDENCE THAT THE CONTROLS MATTER

In its 2019 report, Verizon said none of the payment-card breaches its investigators had examined involved an environment fully compliant with PCI DSS at the time. [3]

This describes investigated cases; it is not a guarantee of zero breaches.

5 THE PAYOFF FROM LAYERED PAYMENT SECURITY

87% LOWER
Counterfeit-fraud losses
At fully chip-enabled U.S. merchants: Q1 2019 versus Q3 2015. Visa data. [4]

$40 BILLION
Estimated fraud losses prevented
Reported by Visa for its wider payment-security efforts in 2023. [5]

These results measure EMV and Visa's wider defenses; they do not isolate savings from PCI DSS.

Shared rules + verified controls + ongoing upkeep = stronger payment security

Sources: [1] PCI SSC [2] EMVCo [3] Verizon, 2019 PSR [4] Visa, June 2019 [5] Visa, 2023 responsibility report

To cyber professionals, this is a humiliating and ongoing crisis that makes them look incompetent. For security professionals, it's impossible to ignore one's own country being attacked.

HEALTHCARE RANSOMWARE

Attacking care. Endangering patients.

THIS SHOULD NEVER HAVE BECOME NORMAL.

A GROWING THREAT TO CARE

Healthcare / public health ransomware complaints to FBI IC3 [1]



+93%

reports in 2025 versus 2024

Reported complaints; not a count of all attacks.

THIS HAS ALREADY HIT THE U.S. AND NATO ALLIES

ENGLAND · MAY 2017

WannaCry

19,000+

appointments estimated cancelled.

Five emergency departments diverted patients. [2]



CZECHIA · MARCH 2020

Brno University Hospital

Patients transferred. Procedures postponed.

The hospital shut its network and returned to paper as the pandemic began. [3]



UNITED STATES · FEBRUARY 2024

Change Healthcare

Millions affected

Disrupted prescription and care access nationwide. Patients faced cash charges and trouble getting refills. [4]



LONDON · JUNE 2024

Synnovis

10,152 appointments

1,710 planned procedures

postponed at King's and Guy's/St Thomas' trusts after laboratory services were disrupted. [5]





YES, THIS CAN KILL.

The UK government confirmed that the Synnovis attack contributed to a patient's death. [6]

Delayed medicines. Postponed treatment. Diverted ambulances. Patients who cannot safely wait bear the risk.



A THREAT TO LIFE

"If you prevent a high impact ransomware attack, you're preventing an act of cyber terrorism."

— John Riggi, American Hospital Association [7]



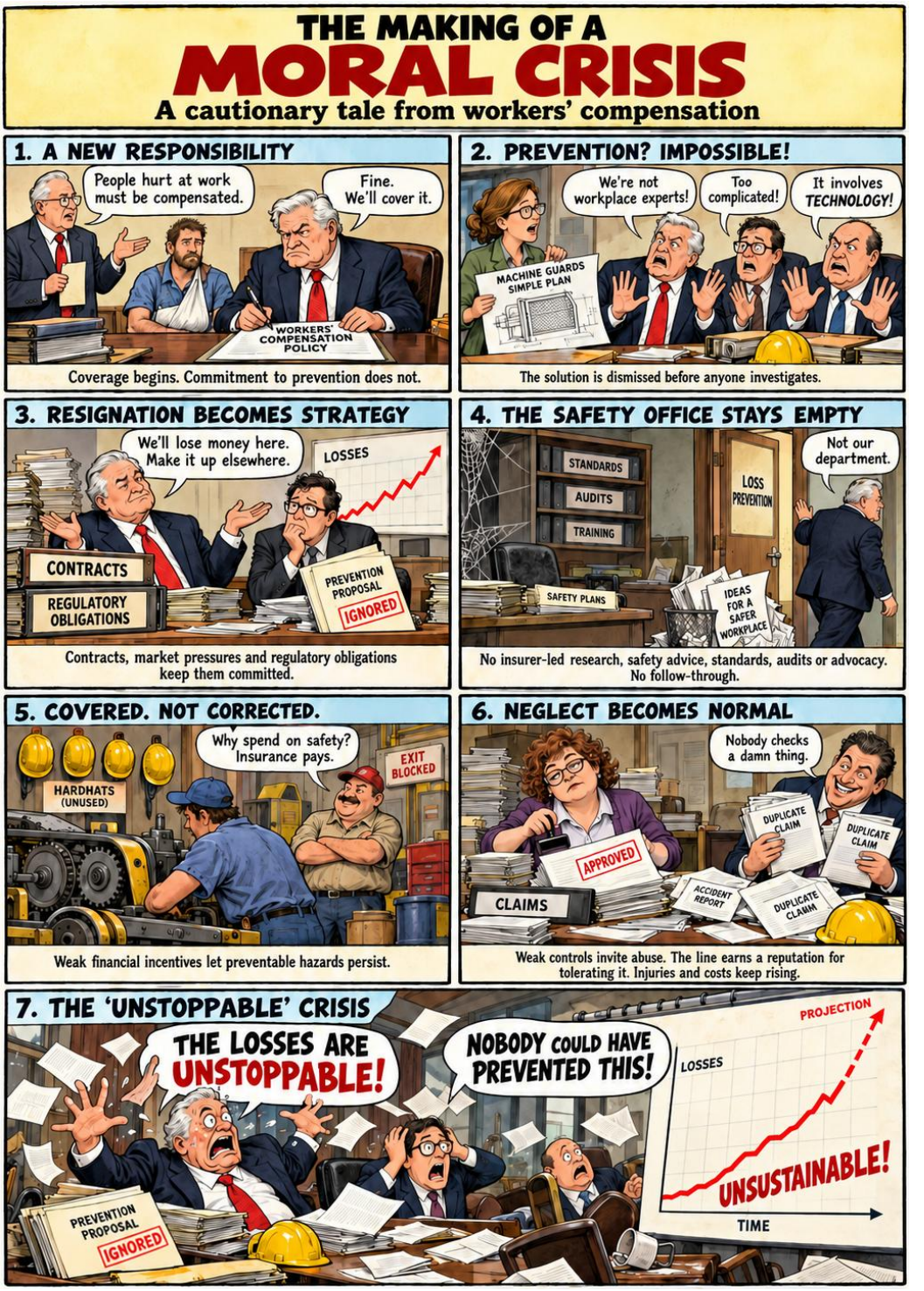
EVEN WAR HAS RULES

The Geneva Conventions protect medical care in armed conflict. Cyber operations against protected hospitals can violate those rules. [8]

REPEATED DOES NOT MEAN ACCEPTABLE.

Patients must never become bargaining chips.

[1] FBI IC3 2022–25 [2] NAO [3] WHO [4] U.S. Senate [5] NHS England [6] UK Parliament [7] AHA [8] ICRC



By the 1990s, years of neglect had resulted in losses so high they threatened the solvency of the entire insurance sector. It was not simply an insurance crisis. At its root, it was a workplace safety crisis. After years of losses, carriers were resigned to high losses, poor compliance and rampant fraud.

Facing an existential threat, the industry spent billions to fix the problem it helped to create. This is exactly the kind of thing that can happen when losses are accepted without question and allowed to grow unchecked. We can see the signs that this may be happening again!

WHY INSURANCE UNDERWRITING MATTERS

Insurance is a major, often overlooked pillar of the economy. It puts a financial structure around uncertain losses, helping households and businesses plan, invest and recover.

UNDERWRITING CONNECTS RISK TO FINANCIAL DECISIONS

Which risks to cover • What to charge • What safeguards to require

THE INSURANCE SYSTEM AT WORK

Insurers, risk engineers, research institutes and reinsurers contribute in different ways.

1



UNDERSTAND THE LOSSES

Claims and exposure data reveal where damage occurs, what causes it and how costly it becomes.

2



PUT A PRICE ON RISK

Estimates of loss frequency and severity inform premiums and coverage decisions.

3



FUND THE SCIENCE

Insurers and industry institutes support research into hazards and effective protection.

4



TEST WHAT WORKS

Crash tests, fire experiments and storm simulations evaluate materials, designs and safeguards.

5



DEVELOP PRACTICAL STANDARDS

Evidence becomes engineering guidance, product criteria and contributions to building codes.

6



HELP BUSINESSES PREVENT LOSS

Risk engineers inspect facilities and recommend changes to equipment, operations and maintenance.

7



SHARE SAFETY KNOWLEDGE

Industry institutes bring competing insurers together to develop research and publish findings.

8



MAKE PREVENTION PAY

Premiums, deductibles and coverage conditions can reward stronger safeguards.

9



SPREAD FINANCIAL EXPOSURE

Insurance pools risks; reinsurance spreads large exposures across a wider financial system.

10



FINANCE RECOVERY

Claims payments support repair and rebuilding. Loss experience informs the next round of prevention.

A DIRECT FINANCIAL STAKE IN SAFETY

Insurers bear covered losses. That exposure gives them a reason to understand risk and help reduce it.

IN PRACTICE | **IIHS:** crash testing | **HLDI:** loss data | **IBHS:** building research | **FM:** risk engineering

Sources: OECD • NAIC • IIHS/HLDI • IBHS • FM

Examples emphasize property and casualty insurance.