DAY 1

Language Modeling from Scratch

From data to behavior: the full language-modeling pipeline

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1	2	3	4
???	???	???	???
Click for answer	Click for answer	Click for answer	Click for answer
Show	Show	Show	Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

Data

Curated text that teaches the model what patterns exist.

2

???

Click for answer

Show

3

???

Click for answer

Show

4

???

Click for answer

Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

Data

Curated text that teaches the model what patterns exist.

2

???

Click for answer

Show

3

???

Click for answer

Show

4

???

Click for answer

Show

New York Times sues AI startup for ‘illegal’ copying of millions of articles

Perplexity AI also faces lawsuit from Murdoch-owned Dow Jones and New York Post for its use of copyrighted content

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1	2	3	4
Data Curated text that teaches the model what patterns exist.	??? Click for answer Show	??? Click for answer Show	??? Click for answer Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

Data

Curated text that teaches the model what patterns exist.

2

Architecture

Transformer blocks that turn context into next-token scores.

3

???

Click for answer

Show

4

???

Click for answer

Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1 **Data**

Curated text that teaches the model what patterns exist.

2 **Architecture**

Transformer blocks that turn context into next-token scores.

3 **???**

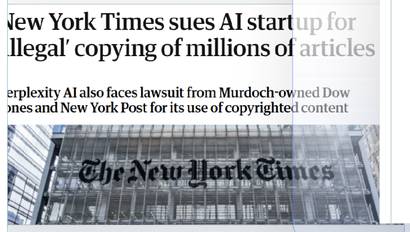
Click for answer

Show

4 **???**

Click for answer

Show



Andrej Karpathy ✓
@karpathy · **Follow**

The Transformer is a magnificient neural network architecture because it is a general-purpose differentiable computer. It is simultaneously:
1) expressive (in the forward pass)
2) optimizable (via backpropagation+gradient descent)
3) efficient (high parallelism compute graph)

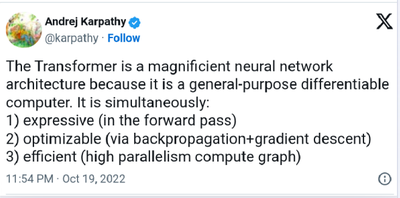
11:54 PM · Oct 19, 2022

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1	2	3	4
Data	**Architecture**	**???**	**???**
Curated text that teaches the model what patterns exist.	Transformer blocks that turn context into next-token scores.	Click for answer	Click for answer
		Show	Show



AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

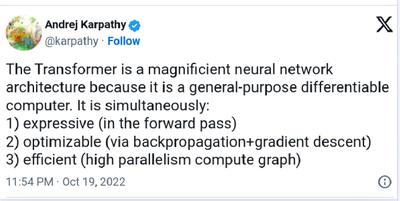
Data

Curated text that teaches the model what patterns exist.

2

Architecture

Transformer blocks that turn context into next-token scores.



3

Learning Signal

Next-token loss plus gradient updates that change the weights.

4

???

Click for answer

Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

Data

Curated text that teaches the model what patterns exist.

2

Architecture

Transformer blocks that turn context into next-token scores.

3

Learning Signal

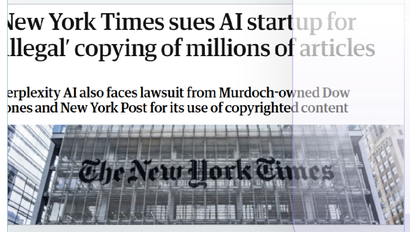
Next-token loss plus gradient updates that change the weights.

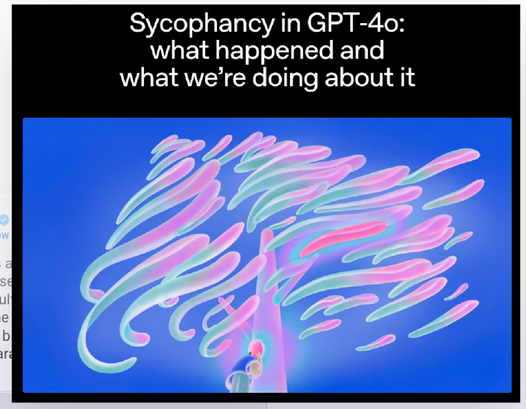
4

???

Click for answer

Show





AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

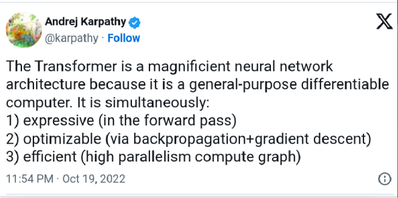
Data

Curated text that teaches the model what patterns exist.

2

Architecture

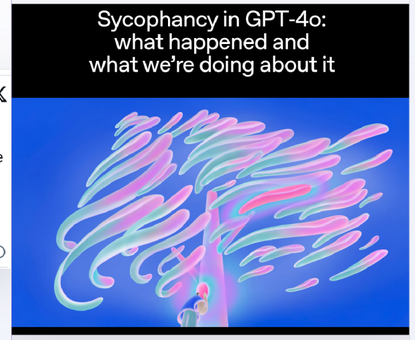
Transformer blocks that turn context into next-token scores.



3

Learning Signal

Next-token loss plus gradient updates that change the weights.



4

???

Click for answer

Show

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

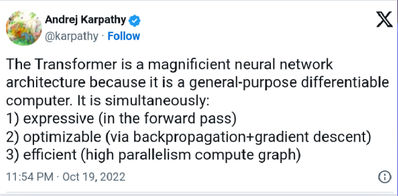
Data

Curated text that teaches the model what patterns exist.

2

Architecture

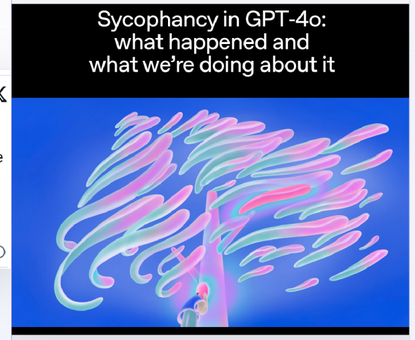
Transformer blocks that turn context into next-token scores.



3

Learning Signal

Next-token loss plus gradient updates that change the weights.



4

Compute Power

Massive hardware that makes large-scale training and inference actually feasible.

AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

Data

Curated text that teaches the model what patterns exist.

2

Architecture

Transformer blocks that turn context into next-token scores.

3

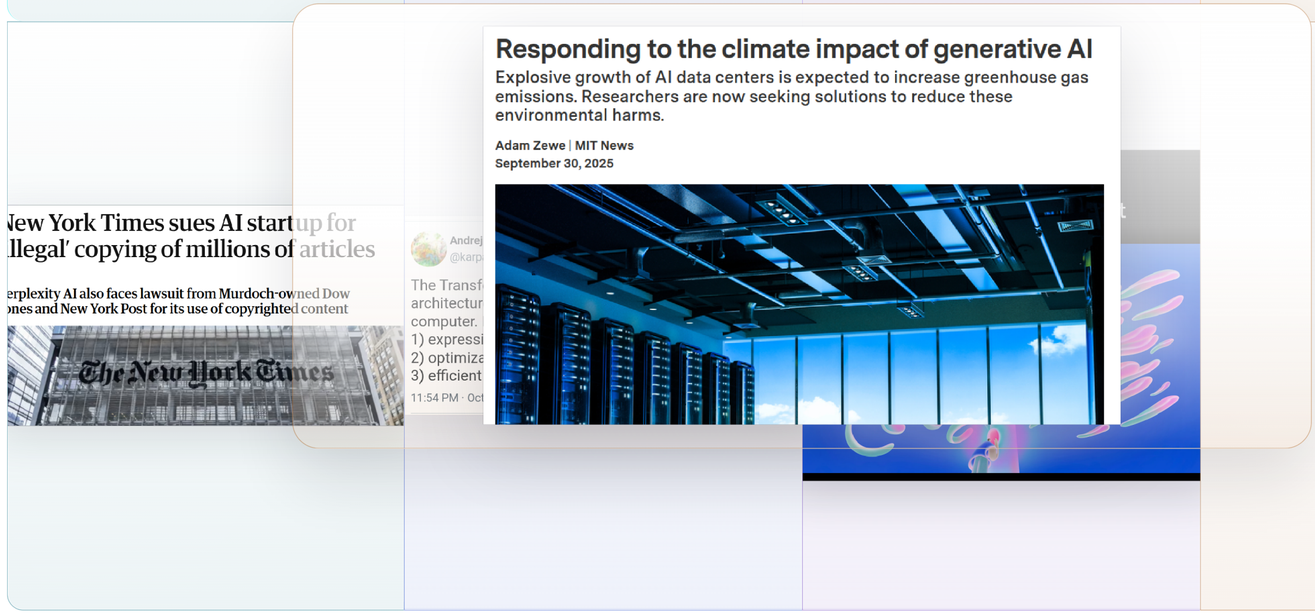
Learning Signal

Next-token loss plus gradient updates that change the weights.

4

Compute Power

Massive hardware that makes large-scale training and inference actually feasible.



AUDIENCE PROMPT

What four ingredients make an LLM work?

If you built one from scratch, what all would you need?

1

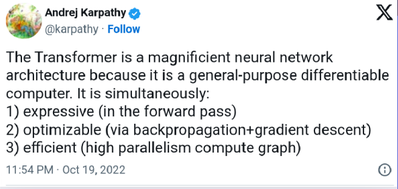
Data

Curated text that teaches the model what patterns exist.

2

Architecture

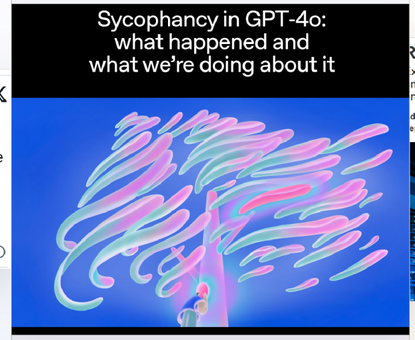
Transformer blocks that turn context into next-token scores.



3

Learning Signal

Next-token loss plus gradient updates that change the weights.



4

Compute Power

Massive hardware that makes large-scale training and inference actually feasible.

ROADMAP

Workshop roadmap

DAY 1 — TODAY

Why LLMs behave the way they do

- → Inference vs. training
- → Next-token prediction
- → Loss → gradients → updates
- → Why this objective works
- → Data, scale, and alignment

¡ Today: treat the model as a **system being shaped.**

DAY 2 — TOMORROW

How a Transformer processes text and computes the next word

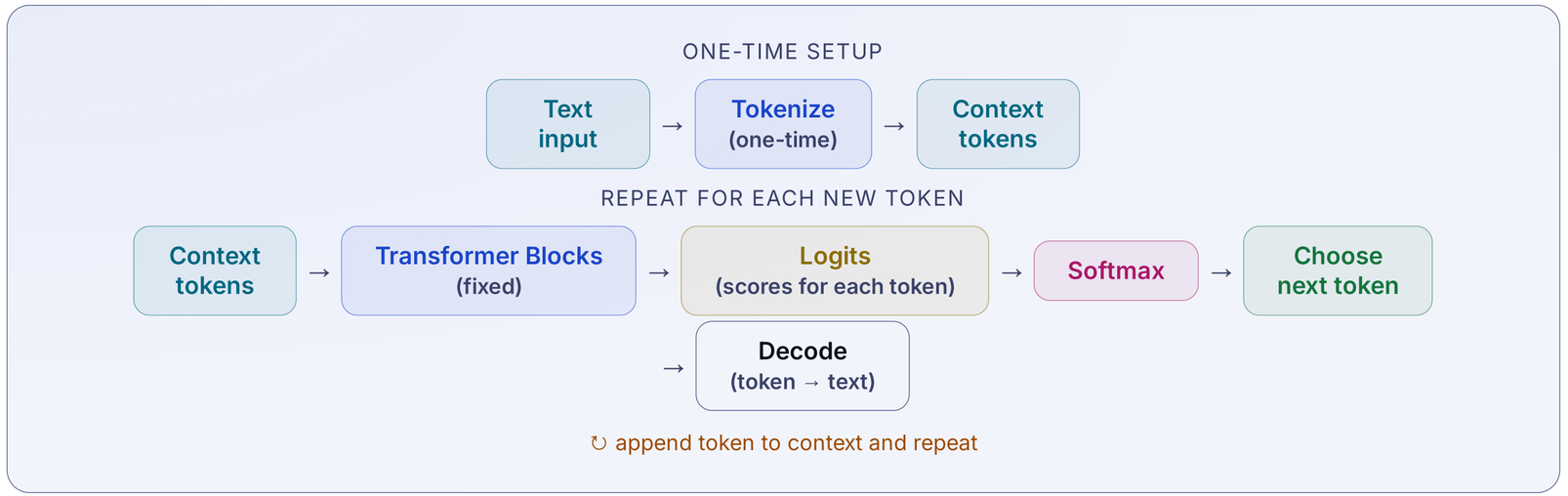
- → Tokenization & embeddings
- → Self-attention
- → Feed-forward layers
- → Full forward pass end-to-end

¡ Tomorrow: treat it as a **machine executing computation.**

INFERENCE LOOP

The inference (use-time) loop

At use time, the model runs a fixed computation. The input text is tokenized once. After that, the model repeats the same loop: score next tokens, choose one, decode it, append it, and continue.

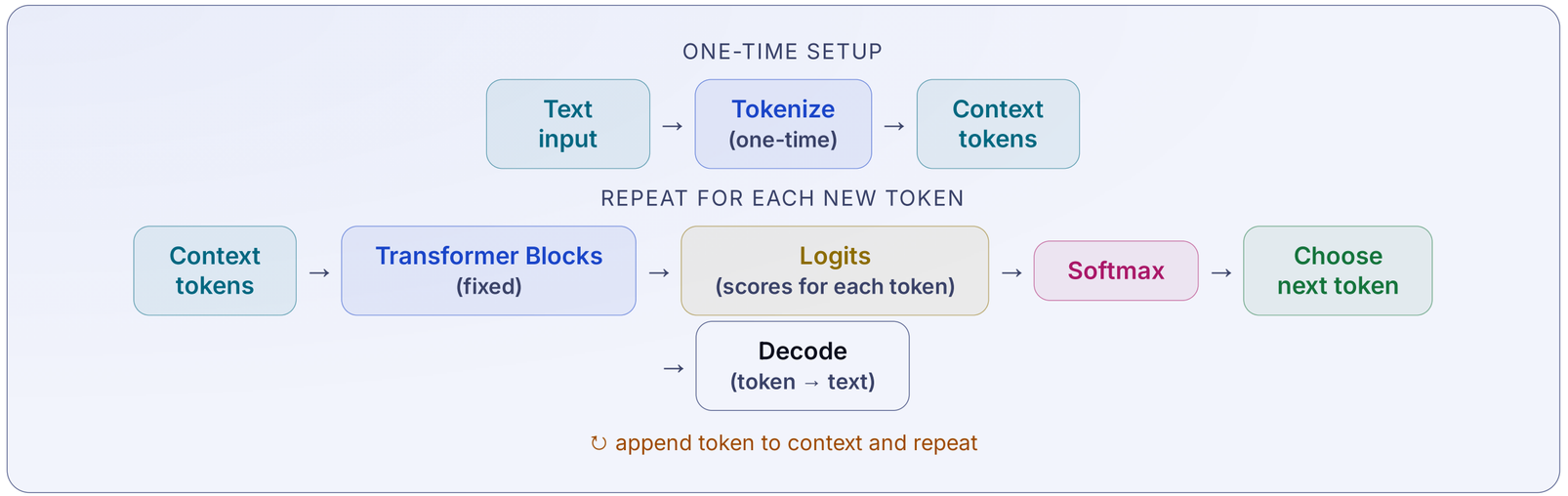


The	capital	of	Pakistan	is	next?

INFERENCE LOOP

The inference (use-time) loop

At use time, the model runs a fixed computation. The input text is tokenized once. After that, the model repeats the same loop: score next tokens, choose one, decode it, append it, and continue.

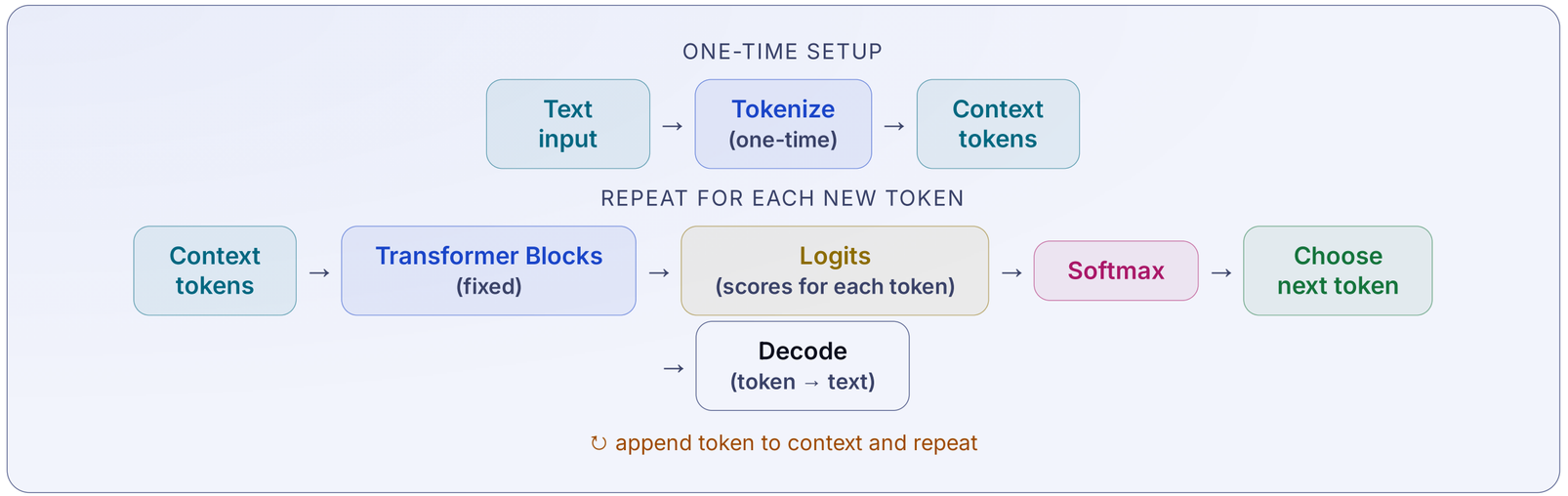


The	capital	of	Pakistan	is	Islamabad	next?

INFERENCE LOOP

The inference (use-time) loop

At use time, the model runs a fixed computation. The input text is tokenized once. After that, the model repeats the same loop: score next tokens, choose one, decode it, append it, and continue.

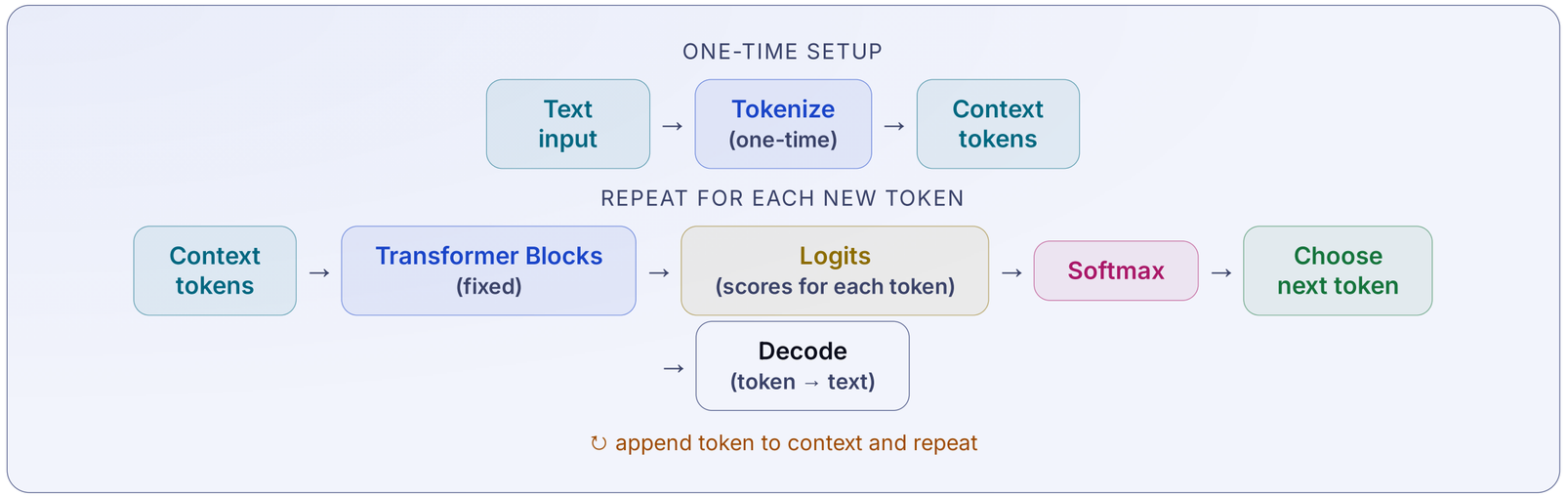


The	capital	of	Pakistan	is	Islamabad	.	next?

INFERENCE LOOP

The inference (use-time) loop

At use time, the model runs a fixed computation. The input text is tokenized once. After that, the model repeats the same loop: score next tokens, choose one, decode it, append it, and continue.

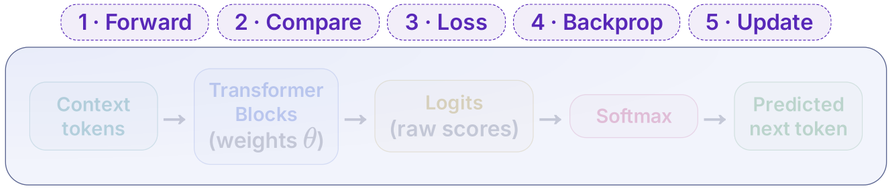


The	capital	of	Pakistan	is	Islamabad	.	⟨EOS⟩

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.

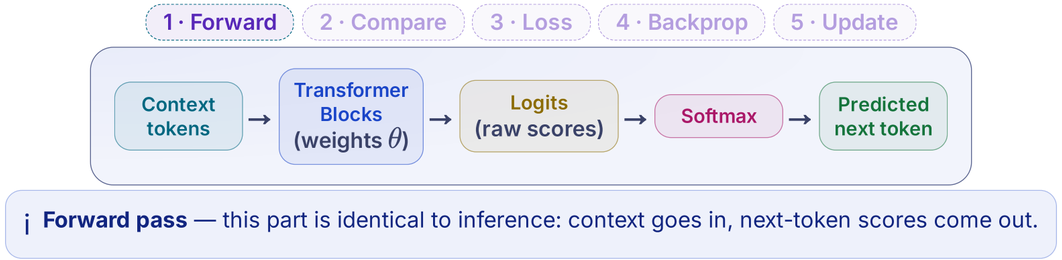


¡ **Reveal the loop in order.** Start with the forward pass, then compare with the correct token, compute loss, backpropagate error, and update the weights.

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.

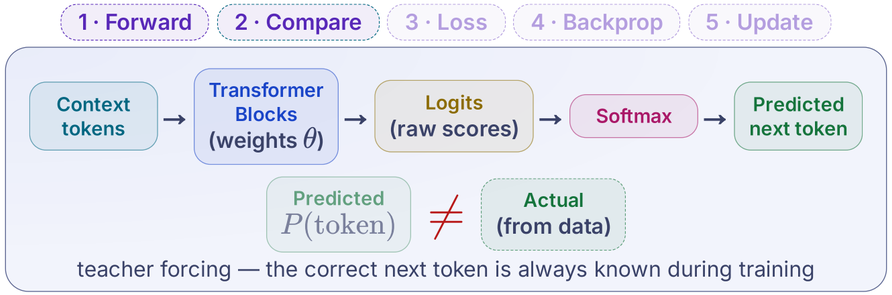


¡ **Forward pass** — this part is identical to inference: context goes in, next-token scores come out.

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.

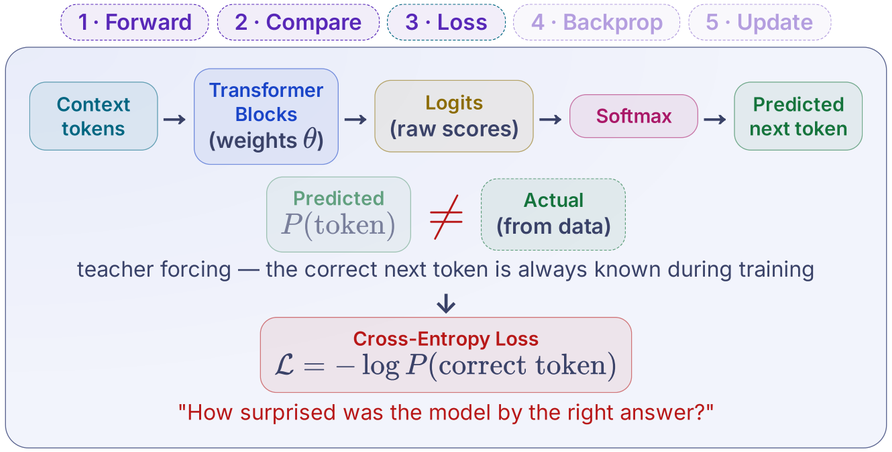


¡ **Ground truth** — training already knows the correct next token from the dataset, so the model can compare prediction vs. answer.

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.

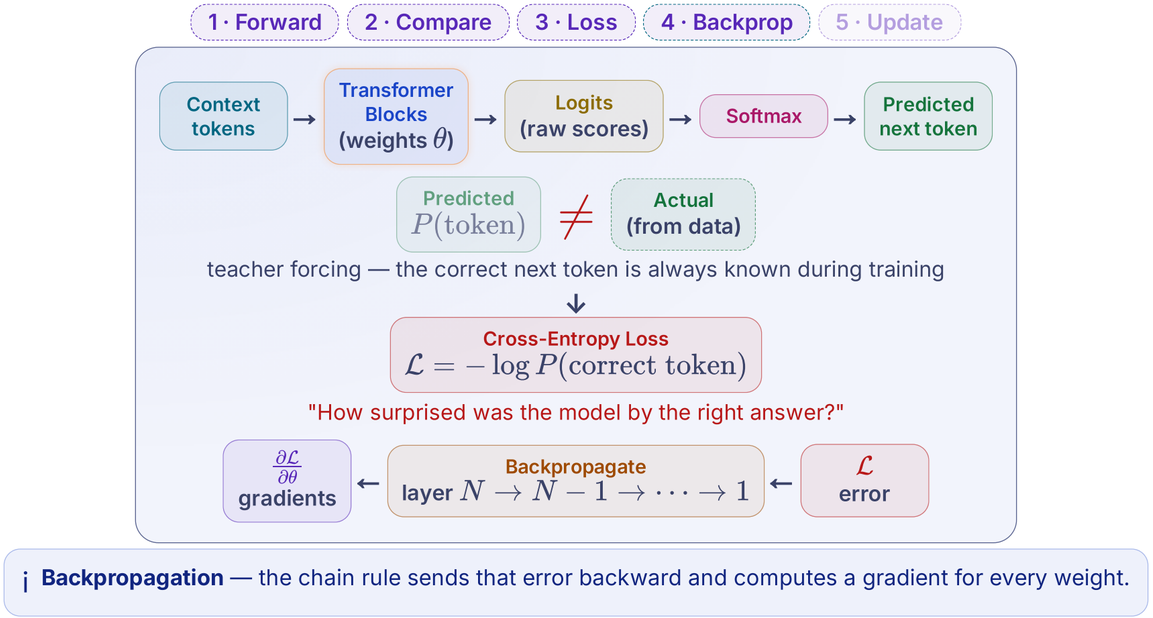


¡ **Cross-entropy loss** — if the model gave low probability to the correct token, the loss is high: $\mathcal{L} = -\log P(\text{correct token})$.

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.

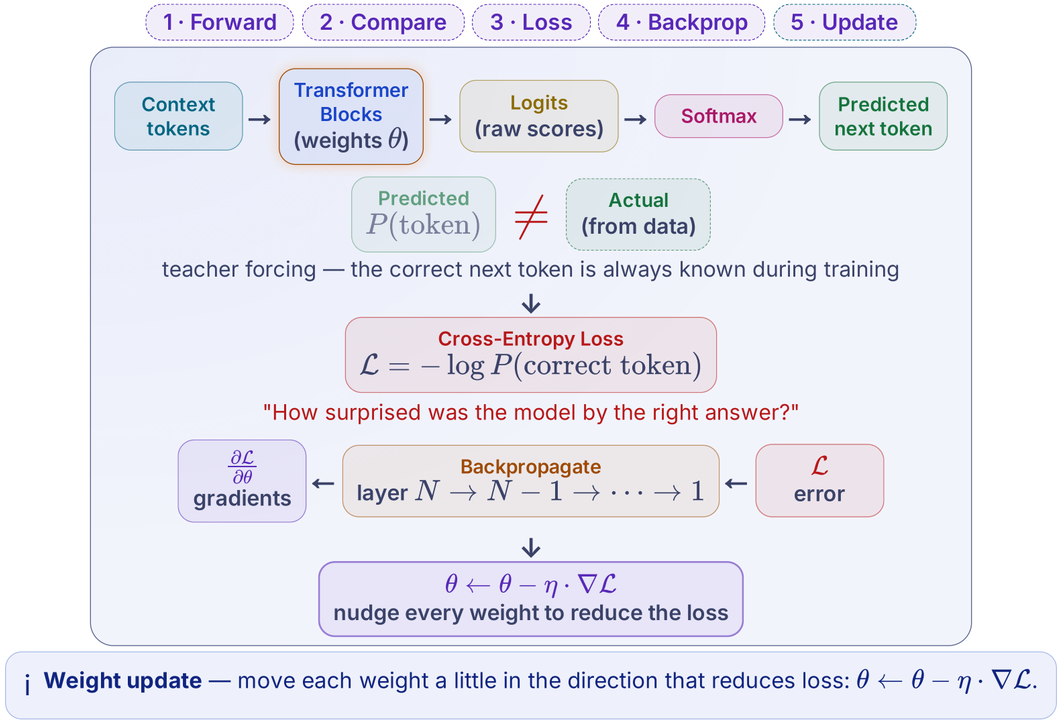


¡ **Backpropagation** — the chain rule sends that error backward and computes a gradient for every weight.

TRAINING LOOP

The training (learning) loop

Training starts with the **same forward pass** as inference, then adds ground truth, loss, backpropagation, and a weight update.



¡ **Weight update** — move each weight a little in the direction that reduces loss: $\theta \leftarrow \theta - \eta \cdot \nabla \mathcal{L}$.

OBJECTIVE

Teacher forcing: shift the sequence by one token

One training sequence becomes many prediction problems: each prefix is used to predict the token that comes next.

Given a token sequence $x_1, x_2, \ldots, x_T$, **teacher forcing** shifts it by one position:

Inputs (prefixes)

x_1 x_2 ... x_{T-1}

Targets (next tokens)

x_2 x_3 ... x_T

At position t, the model sees $x_{\leq t}$ and is scored on the true next token x_{t+1}.

→ **Training objective:** assign high probability to the true next token at every position in the sequence.

LOSS

The equivalent loss function

Same objective, written as a loss: negative log-likelihood, or cross-entropy.

$$\mathcal{L}(\theta) = -\frac{1}{T-1}\sum_{t=1}^{T-1} \log P_\theta(x_{t+1} \mid x_{\leq t})$$

Lower loss = the model assigns higher probability to tokens that actually occurred in the dataset

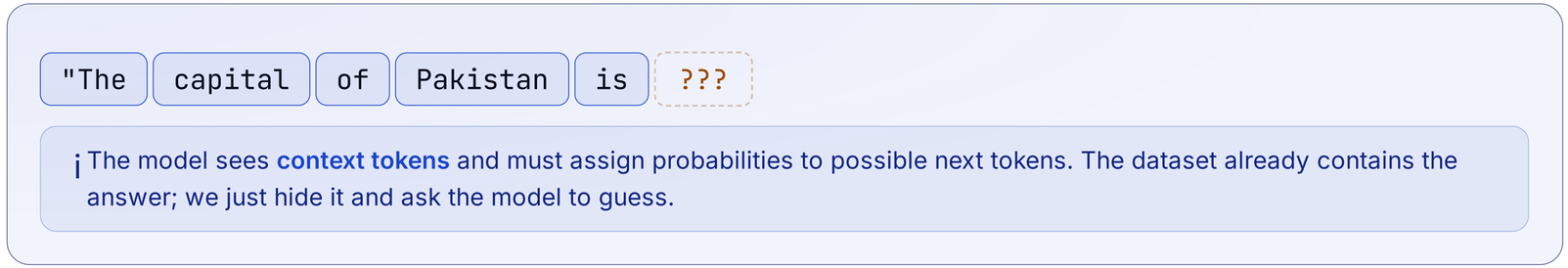
Key insight: this is **distribution matching**, not decision-making. The model is pushed to fit observed text, not to pursue truth or goals.

→ **Next:** one training iteration end-to-end — forward → loss → backward → update.

STEP 1 — THE TASK

Predict the next token

We'll trace one concrete training example from input context to weight update.



¡ The model sees **context tokens** and must assign probabilities to possible next tokens. The dataset already contains the answer; we just hide it and ask the model to guess.

STEP 2 — FORWARD PASS

The model outputs a probability distribution

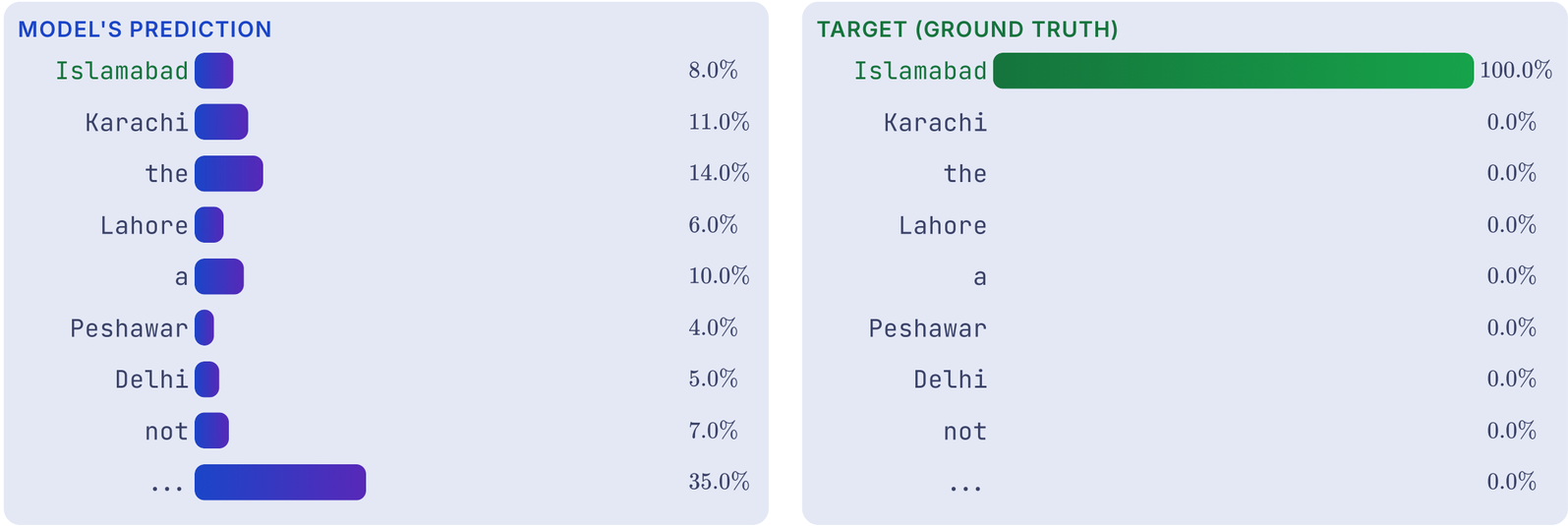
The prefix runs through the Transformer and becomes a **probability distribution over the vocabulary**.

¡ The final layer emits **logits**, and **softmax** turns them into probabilities that sum to 1. Early in training, this still looks a lot like guessing.

STEP 3 — THE TARGET

What the model *should* have predicted

We know the correct next token from the training data. The **target distribution** puts 100% probability on the correct answer.

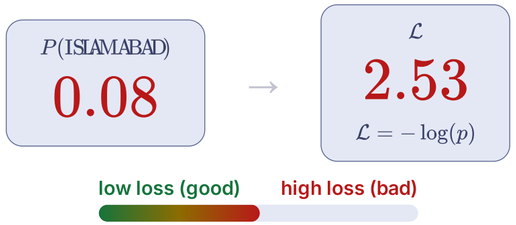


! The model only gave 8% to **"Islamabad"**, but the target says it should be 100%. That gap is what the **loss function** measures.

Cross-entropy loss: how wrong is the model?

The loss measures the gap between prediction and target. For next-token prediction, we use **cross-entropy loss**.

$$\mathcal{L} = -\log(P_{\text{model}}(\text{Islamabad})) = -\log(0.08) = 2.53$$



¡ **Intuition:** if the model gave 100% to "Islamabad", $\mathcal{L} = -\log(1) = 0$ (perfect). If only 1%, $\mathcal{L} = -\log(0.01) \approx 4.6$ (terrible). The loss captures "how surprised the model is by the right answer."

Backpropagation computes gradients

The loss signal flows *backward* through the network. For each parameter, we compute how changing it would affect the loss.

$\mathcal{L} = 2.53$ | $\frac{\partial \mathcal{L}}{\partial \theta}$ compute gradients | $\theta \leftarrow \theta - \eta \nabla \mathcal{L}$ update params

◀◀◀◀ gradients flow backward

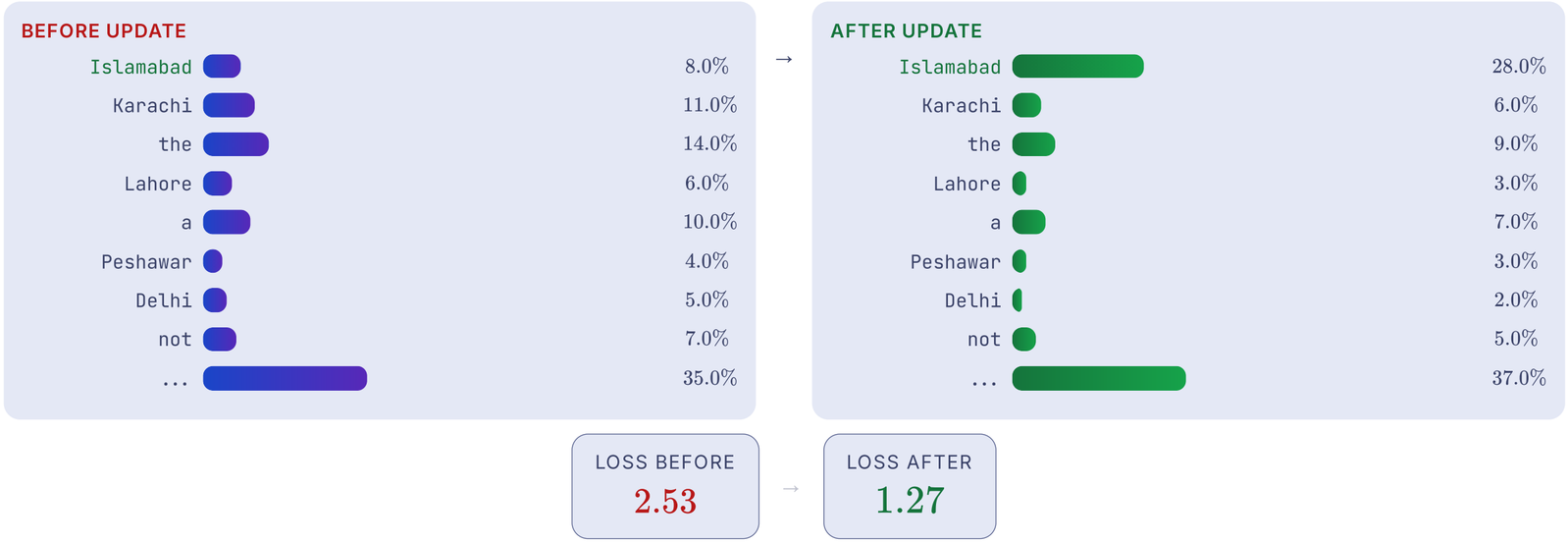
$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

¡ η is the learning rate. The **gradient** tells each parameter which way to move and by how much.

STEP 6 — AFTER UPDATE

The model improves!

After the update, the model assigns *more* probability to the correct token. Repeat this across many examples, and the model learns.



✓ After one step, "Islamabad" rises from 8% to 28%, and the loss drops from 2.53 to 1.27. Scale that up, and the model gradually learns.

KEY INSIGHT

What the loss never names

After one full update, the core constraint is visible: training directly rewards only better **next-token prediction**. Anything richer has to be *useful for prediction*, not separately requested.

× Truth × Reasoning × Helpfulness × Safety × Goals × Consistency

$$\mathcal{L} = -\log P_\theta(x_{t+1} \mid x_{\leq t})$$

that is the whole signal

¡ **The narrow bottleneck:** everything the model learns has to pass through lowering next-token error.

? **So why do facts, code, and reasoning show up at all?** Next we zoom out from one update to the broader pressures created by repeating this objective at scale.

KEY IDEA

Why a local objective can create reasoning-like behavior

Local objective, global pressure: the model is graded one token at a time, but getting that token right often requires a coherent picture of the whole prefix.

At each position: raise the probability of the true next token

Reveal the hidden pressures

ENTITIES
Track who is who and how references connect across the passage.

CAUSAL CUES
Infer what caused what, and which outcomes are still possible.

CONSTRAINTS
Rule out continuations that violate timing, access, or prior facts.

MULTI-STEP STRUCTURE
Maintain the shape of a plan, proof, argument, or story.

KEY IDEA

Why a local objective can create reasoning-like behavior

Local objective, global pressure: the model is graded one token at a time, but getting that token right often requires a coherent picture of the whole prefix.

At each position: raise the probability of the true next token

Reveal the hidden pressures

ENTITIES
Track who is who and how references connect across the passage.

CAUSAL CUES
Infer what caused what, and which outcomes are still possible.

CONSTRAINTS
Rule out continuations that violate timing, access, or prior facts.

MULTI-STEP STRUCTURE
Maintain the shape of a plan, proof, argument, or story.

¡ **"Reasoning-like" behavior can be the cheapest way to reduce prediction error,** even when the loss never asks for reasoning explicitly.

INTERACTIVE EXAMPLE

Next-token prediction in a Pakistani drama

The answer is implied, not stated. To predict the next token here, the model has to combine timing, access, and entity tracking.

```
Karachi, one evening...
10:30 Shahbaz Ahmed is found dead in his office. Cause of death: poison.
Three people matter in the story: Farah (secretary), Salman (business partner), and Rukhsana (cleaner).
  9:50 Farah has already left the office.
  10:00 Salman leaves for a meeting.
  10:20 Rukhsana makes tea for Shahbaz.
  Forensics: the poison is taken between 10:20 and 10:30.
  CCTV: after 10:00, only one person enters the pantry.
Police review the evidence and conclude the killer was ____ ___ ----------
```

Farah | Salman | Rukhsana | Reveal answer

? Pick the suspect whose timeline fits every clue.

INTERACTIVE EXAMPLE

Next-token prediction in a Pakistani drama

The answer is implied, not stated. To predict the next token here, the model has to combine timing, access, and entity tracking.

```
Karachi, one evening...
10:30 Shahbaz Ahmed is found dead in his office. Cause of death: poison.
Three people matter in the story: Farah (secretary), Salman (business partner), and Rukhsana (cleaner).
  9:50 Farah has already left the office.
  10:00 Salman leaves for a meeting.
  10:20 Rukhsana makes tea for Shahbaz.
  Forensics: the poison is taken between 10:20 and 10:30.
  CCTV: after 10:00, only one person enters the pantry.
Police review the evidence and conclude the killer was  Rukhsana
```

Farah | Salman | Rukhsana | Reveal answer

✓ Best next-token continuation here: **Rukhsana**, based on timing + access constraints.

→ **No line explicitly states the answer.** Correct next-token prediction requires integrating entity tracking, timing, and access constraints.

TAKEAWAY

One next token can depend on the whole story

The label is still local, but the evidence is distributed across many lines. That is why next-token training can reward richer internal state.

What training still asks for

$$\hat{x}_{t+1} \sim P_\theta(x_{t+1} \mid x_{\leq t})$$

Given a prefix, output a distribution over the next token. During training, we reward higher probability on the true continuation.

To predict correctly, it must track

1. **Time window:** when the poison could have been taken (10:20–10:30)
2. **Access:** who could have reached the pantry during that window (departures + CCTV)
3. **Consistency:** reconcile all lines into one coherent timeline

✓ **Key point:** local supervision can reward rich internal state. Here, the model is pushed to build a coherent case file: suspects, timeline, and constraints.

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

```
The Eiffel Tower is located in ??? type?
```

```
After the rain stopped, the children ran outside to ??? type?
```

```
She said "I'm not angry," but the tone of her voice suggested she was actually quite ??? type?
```

```
If all roses are flowers, and all flowers need water, then all roses need ??? type?
```

```
The sum of 127 and 385 is ??? type?
```

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ??? type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris type?

After the rain stopped, the children ran outside to ??? type?

She said "I'm not angry," but the tone of her voice suggested she was actually quite ??? type?

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to ??? type?

She said "I'm not angry," but the tone of her voice suggested she was actually quite ??? type?

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ??? type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play type?

She said "I'm not angry," but the tone of her voice suggested she was actually quite ??? type?

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite ??? type?

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ??? type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset type?

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need ??? type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water type?

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???   type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water logic

The sum of 127 and 385 is ??? type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???
```

type?

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water logic

The sum of 127 and 385 is 512 type?

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water logic

The sum of 127 and 385 is 512 arithmetic

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + ???   type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water logic

The sum of 127 and 385 is 512 arithmetic

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + fibonacci(n-2)  type?
```

INTERACTIVE

ACTIVITY

You Are the Language Model

Try each blank first. Then reveal what kind of knowledge it depends on. “Just predict the next token” quietly demands many different capabilities.

The Eiffel Tower is located in Paris world knowledge

After the rain stopped, the children ran outside to play common sense

She said "I'm not angry," but the tone of her voice suggested she was actually quite upset pragmatics

If all roses are flowers, and all flowers need water, then all roses need water logic

The sum of 127 and 385 is 512 arithmetic

```
def fibonacci(n):
  if n ⩽ 1: return n
  return fibonacci(n-1) + fibonacci(n-2)
```

code structure

SCALE

Pretraining at scale repeats the same objective billions of times

Same loop, huge exposure. This is where the model first picks up language fluency, broad statistical regularities, and background knowledge useful for continuation.

$$\min_{\theta} \; -\log P_\theta(x_{t+1} \mid x_{\le t})$$

Context tokens → Predict next token → Compute loss → Update weights ↺

2T-5T
usable training tokens

1k+
GPUs in parallel

weeks
continuous optimization

✓ **What emerges first:** fluent continuation: syntax, grammar, coherence, and broad background knowledge. Specialization and instruction-following come later.

DATA MIXTURE

Capability follows the data mixture

Once the model has language foundations, the next lever is the distribution you continue training on. Different mixtures reward different strengths.

Code-heavy mixture

Example: 80% code + technical text, 20% generic data

1. **Strength:** better symbolic precision, decomposition, and code completion.
2. **Tradeoff:** less stylistic range in narrative or poetic writing.

Literary-heavy mixture

Example: 80% literary/prose text, 20% generic data

1. **Strength:** richer tone control, narrative fluency, and stylistic variation.
2. **Tradeoff:** less reliability on coding and math-heavy symbolic tasks.

! The model is still learning from token statistics, not from “intent.” Change the distribution, and you change what patterns it gets rewarded for.

✓ **Takeaway:** use broad pretraining for foundations, then use the data mixture to steer specialization.

INTERACTIVE

It's still just a text completer

Pretraining made the model a powerful completer, not automatically a helpful assistant. So what continuation does it find likely here?

What continuation does the base model expect?

```
What is the capital of France?
```

Show likely continuation

INTERACTIVE

It's still just a text completer

Pretraining made the model a powerful completer, not automatically a helpful assistant. So what continuation does it find likely here?

What continuation does the base model expect?

```
What is the capital of France?
```

Show likely continuation

```
What is the capital of France?
What is the capital of Germany?
What is the capital of Italy?
What is the capital of Spain?
What is the capital of ...
```

It continued a familiar pattern instead of answering. Quiz lists are common in training data, so the model keeps the list going. Helpfulness is not the default objective.

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

Math — get the answer

```
347 + 128
```

Show fix

Code — get the implementation

```
fibonacci function
```

Show fix

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

```
User: What is the capital of France?
Assistant:
```

→ "The capital of France is Paris."

Math — get the answer

```
347 + 128
```

Show fix

Code — get the implementation

```
fibonacci function
```

Show fix

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

```
User: What is the capital of France?
Assistant:
```

→ "The capital of France is Paris."

Math — get the answer

```
347 + 128
```

Show fix

```
347 + 128 =
```

→ "475"

Code — get the implementation

```
fibonacci function
```

Show fix

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

```
User: What is the capital of France?
Assistant:
```

→ "The capital of France is Paris."

Math — get the answer

```
347 + 128
```

Show fix

```
347 + 128 =
```

→ "475"

Code — get the implementation

```
fibonacci function
```

Show fix

```
def fibonacci(n):
    """Return the nth Fibonacci
number."""
```

→ completes the function body

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

```
User: What is the capital of France?
Assistant:
```

→ "The capital of France is Paris."

Math — get the answer

```
347 + 128
```

Show fix

```
347 + 128 =
```

→ "475"

Code — get the implementation

```
fibonacci function
```

Show fix

```
def fibonacci(n):
    """Return the nth Fibonacci
number."""
```

→ completes the function body

✓ **You are not changing the model.** You are only changing the context so a reply, answer, or function body becomes the most likely continuation.

INTERACTIVE

Can we fix this without changing the model?

The weights are frozen. All you can change is the **prompt**. Can formatting alone steer next-token completion into useful behavior?

Prompt shaping can steer behavior

Same model, same weights. Only context formatting changes what continuation becomes likely.

Chat — get a helpful reply

```
What is the capital of France?
```

Show fix

```
User: What is the capital of France?
Assistant:
```

→ "The capital of France is Paris."

Math — get the answer

```
347 + 128
```

Show fix

```
347 + 128 =
```

→ "475"

Code — get the implementation

```
fibonacci function
```

Show fix

```
def fibonacci(n):
    """Return the nth Fibonacci
number."""
```

→ completes the function body

✓ **You are not changing the model.** You are only changing the context so a reply, answer, or function body becomes the most likely continuation.

! **But this is brittle.** Prompt tricks steer behavior; they do not make helpfulness the default. That is why post-training exists.

POST-TRAINING

Why post-training exists

Prompt tricks can steer a base model, but they are fragile. Post-training updates the weights so assistant-style behavior becomes more consistent, robust, and policy-constrained.

¡ **Prompting:** shape context at inference time. **Post-training:** shape parameters during training.

STEP 5

Supervised fine-tuning (SFT)

1. Train on chat demonstrations: user turns to assistant replies.
2. Teach stable instruction-following and response structure.
3. Make assistant-style continuation the default.

STEP 6

Preference training + RLHF

1. Use human rankings to separate better from worse candidate replies.
2. Push behavior toward helpfulness, policy compliance, and safer refusals.
3. Improve robustness beyond specific prompts or formatting tricks.

Base completer + chat template → SFT → Preferences / RLHF → Aligned assistant

¡ **The model is still doing next-token generation.** Alignment changes which continuations it is trained to prefer.

LIVE DEMO

Common LLM failures, live

Even chat-tuned models can still fail on low-level tasks. Try a legacy chat model on letter counting, reversal, exact copying, sorting, and decimal comparison.

API KEY

Paste an OpenAI-compatible API key

MODEL

`gpt-3.5-turbo-0125`

TAP A PROBE TO AUTO-TYPE AND SEND

Strawberry letters | Mississippi letters | Reverse strawberry | Reverse encyclopedia | Decimal compare | Rank decimals | Repeat poem ×12 | Alphabetize words

Tap a probe or type your own prompt. The transcript stays here until you clear context.

Enter an API key, then send a probe.

Try counting, reversal, copying, sorting, comparison, or another pr

Send | Clear context

What these probes test

1. **Character control:** counting letters or reversing a word requires finer tracking than many token predictors manage well.
2. **Exactness:** copying a word N times or sorting a short list reveals how quickly outputs drift from strict constraints.
3. **Symbolic comparison:** decimals and ranking tasks expose the gap between pattern-matching and exact reasoning.

¡ **Context persists across turns.** Use `Clear context` whenever you want a fresh start.