

LLM Fundamentals II

Inside the Model

In Part I, the model was a black box.
Now we open it up.



Look inside

Follow tokens through the decoder stack.



Understand attention

See how context shapes each next-token prediction.



Build better intuition

Connect architecture to everyday development.

INPUT

The

future

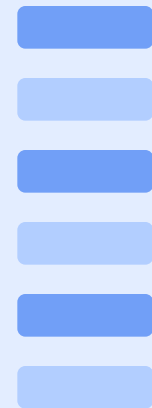
of

AI

is

DECODER-ONLY LANGUAGE MODEL

Token
embeddings



Transformer block × N

Norm → causal attention

+ residual

Norm → MLP

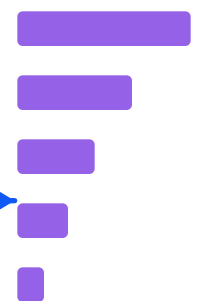
+ residual

RoPE: position in Q and K
Common pre-norm design

Final norm
+ vocab
projection



NEXT TOKEN
probabilities



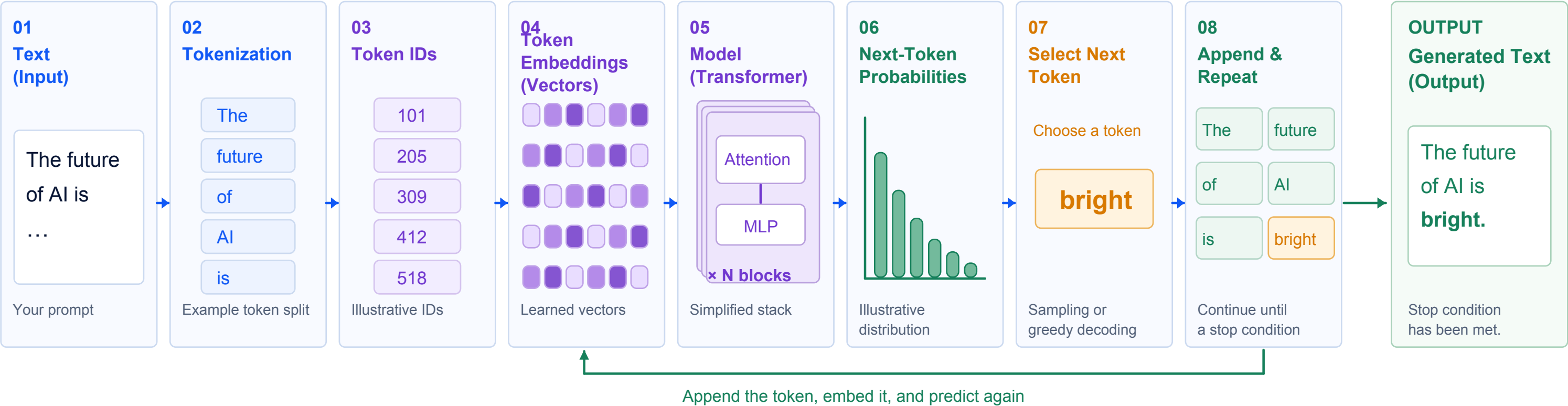
Illustrative
distribution

TODAY'S ROUTE

Tokens → representations → attention → context → next-token prediction

Recap: From Text to Prediction

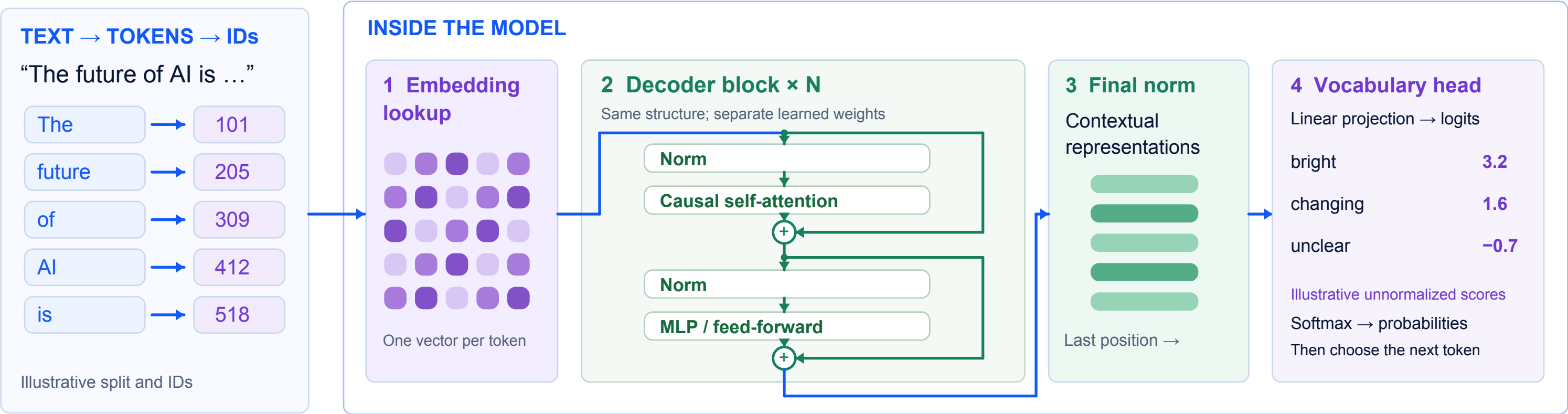
In Part I, we saw the end-to-end flow of how an LLM turns your text into the next token. Here's the big picture again.



Key Takeaway: LLMs don't work on text directly – they operate on numbers (vectors) and predict the most likely next token, one step at a time.

What Is Inside the Black Box?

A decoder-only Transformer turns token IDs into contextual vectors, then next-token scores (logits).
Shown here: a common sequential pre-norm design, as used in Llama-style models.



Rotary Position Embedding (RoPE) rotates query and key vectors according to token position.
This makes attention sensitive to relative position. The causal mask blocks future tokens.

KEY IDEA Attention mixes context; MLPs transform features; the vocabulary head scores next tokens.

Touvron et al. · LLaMA (2023) Hugging Face · Llama architecture

Why Context Matters

The same token can have different contextual representations and next-token predictions.
In a causal decoder, each position uses itself and earlier tokens — never future tokens.

A FINANCIAL CONTEXT

Earlier context

To deposit cash, I visited the bank

Causal attention + MLPs across layers

Contextual representation of "bank"

Illustrative vector

Possible next tokens: branch to and

"bank" refers to a financial institution.

Examples, not measured rankings

B RIVERSIDE CONTEXT

Earlier context

Along the river, we sat on the bank

Causal attention + MLPs across layers

Contextual representation of "bank"

Illustrative vector

Possible next tokens: and of near

"bank" refers to the side of a river.

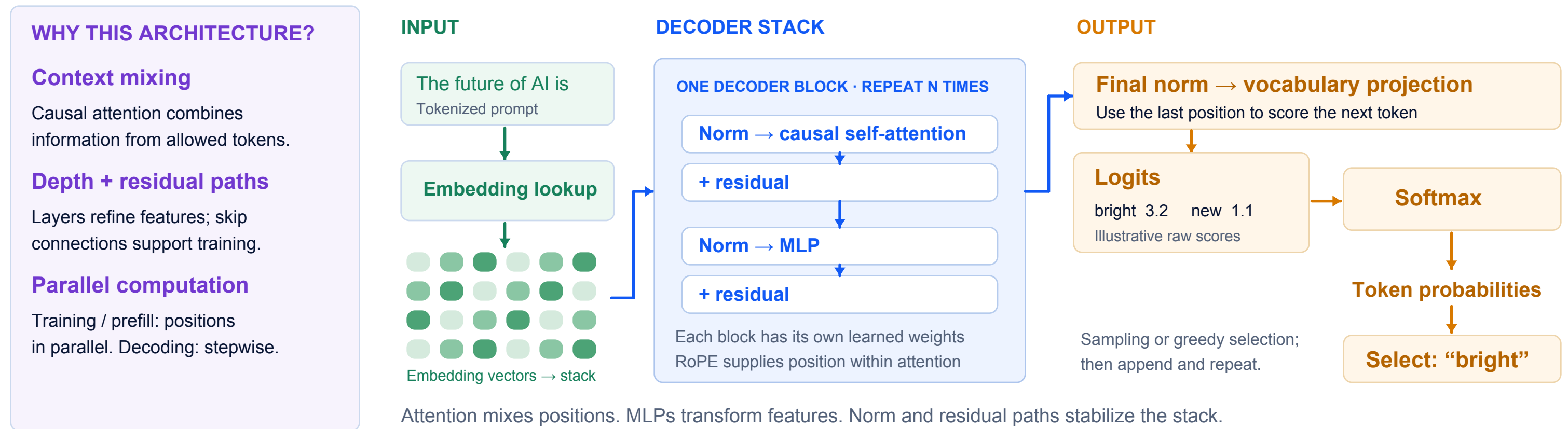
Examples, not measured rankings

KEY IDEA A token’s identity stays the same; its representation changes with the preceding context.

The Transformer – The Core Architecture

Many modern LLMs use a decoder-only Transformer: repeated blocks refine contextual representations.

A common pre-norm design is shown below. Exact block, position and attention choices vary by model.

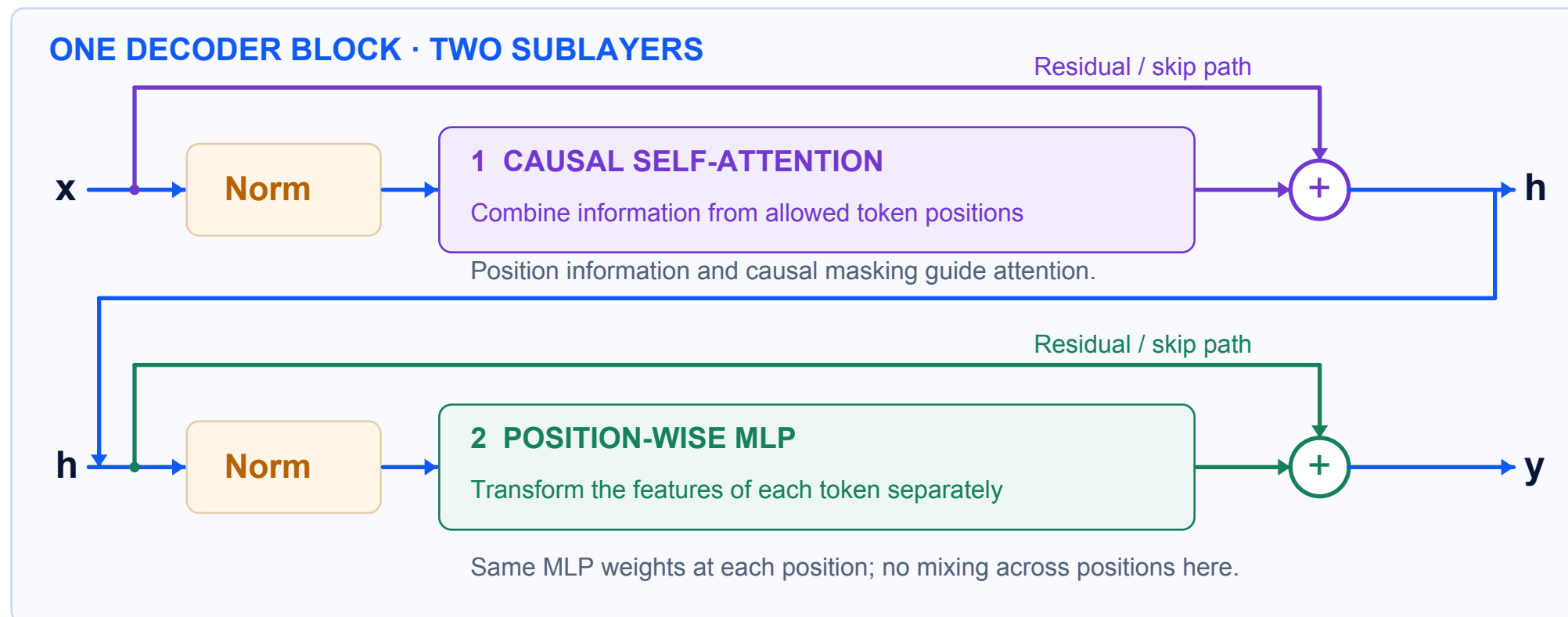


KEY IDEA Repeated decoder blocks build context; the output head turns it into next-token scores.

The Transformer Block

One decoder block contains two sublayers: attention mixes context; the MLP transforms token features.

Sequential pre-norm design: normalize before each sublayer, then add its update to the residual stream.



WHAT EACH PART DOES

Attention

Each position uses itself and earlier positions to combine contextual features.

Norm + residual

Normalize the sublayer input; add its output to the unchanged residual path.

MLP / feed-forward

Transforms features independently at each position.

Norm = normalization. MLP = multi-layer perceptron. Exact normalization and activation choices vary by model.

PRE-NORM

$$h = x + \text{Attention}(\text{Norm}(x))$$

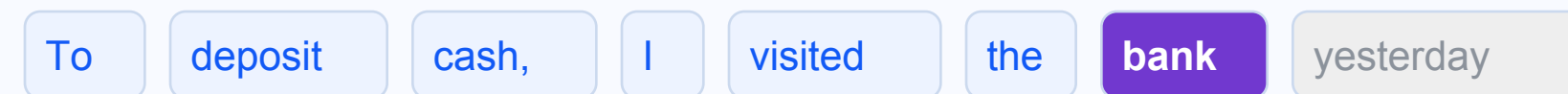
$$y = h + \text{MLP}(\text{Norm}(h))$$

Attention – The Core Idea

For each token position, attention gathers a weighted mix of information from accessible positions.

The weights depend on the input, the layer and the attention head. They are computed anew for each query.

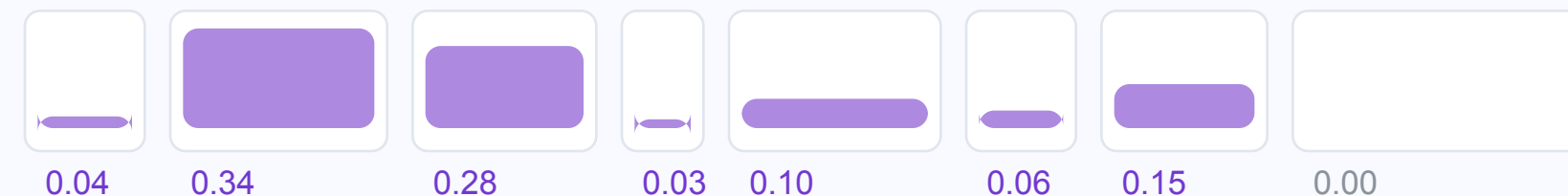
EXAMPLE · PROCESSING “bank”



Earlier tokens + the current token: allowed

Future: masked

Attention weights for the query at “bank”



High weight → a larger contribution from that position’s value vector.

Illustrative weights for one head; not measured model outputs. Weights sum to 1.

HOW THE MIX IS COMPUTED

1 Compare

The current query is compared with the keys of accessible positions.

2 Normalize

Mask future positions, then use softmax to turn the scores into weights.

3 Mix

Multiply each value vector by its weight and sum them into one output vector.

Queries, keys and values are learned projections of token representations. The next slides unpack them.

KEY IDEA

Attention mixes value vectors — it does not select the next token or prove which words explain an answer.

Self-Attention in Practice

Slide 7 showed one query at “bank”. Now apply the same operation at every token position.
Each row has its own query, allowed keys and attention weights. All positions share the learned projections.

WHO CAN ATTEND TO WHOM?

Query ↓ / Key →	To	deposit	cash,	I	visited	the	bank	yesterday
To		×	×	×	×	×	×	×
deposit			×	×	×	×	×	×
cash,				×	×	×	×	×
I					×	×	×	×
visited						×	×	×
the							×	×
bank	.04	.34	.28	.03	.10	.06	.15	0
yesterday								

Blue: allowed · Gray: masked · Purple row: the illustrative weights from slide 7

1 One query per position

“bank” can use itself and earlier tokens.
“yesterday” can also use “bank”.
“bank” cannot look ahead to “yesterday”.

2 One contextual output per position

Each row mixes the accessible value vectors.
The purple row sums to 1; the other rows have their own weights, not shown here.

3 Parallel computation, causal access

Training and prompt prefill compute rows in parallel. The mask still prevents future access.

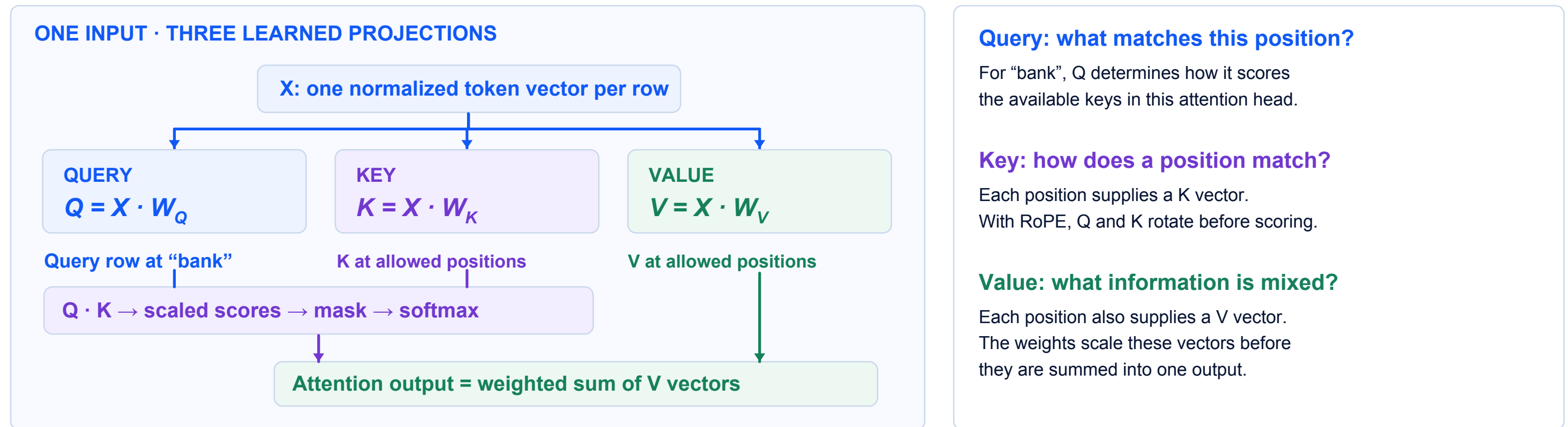
“Self” means queries, keys and values come from the same sequence. Diagram: one head in one layer.

KEY IDEA Every position builds its own contextual representation — without seeing future tokens.

Queries, Keys and Values

Three learned linear projections produce queries, keys and values from the same input representations.

Continue the “bank” example: its query is compared with accessible keys, then the resulting weights mix values.



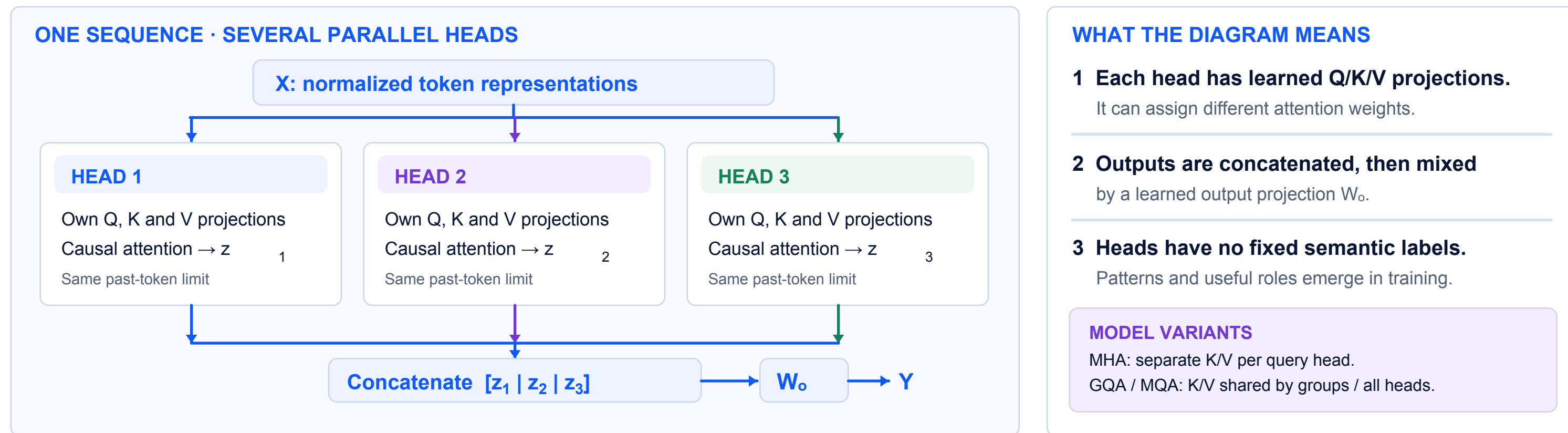
One head shown. X contains one row per token; W matrices are learned projections shared across positions.

KEY IDEA Queries and keys determine the weights. Values supply the information being combined.

Multi-Head Attention

Several attention heads process the same sequence in parallel, using different learned projections.

Their outputs are joined, projected and passed to the residual addition in the decoder block.



Three heads are shown only for clarity. Head counts and K/V sharing depend on the model configuration.

KEY IDEA Multiple learned views of the same context are combined into one attention output.

Position Matters

The same words in a different order can describe a different event.
Many decoder models use rotary position embeddings (RoPE) to make attention position-sensitive.

SAME WORDS · DIFFERENT ROLES

POS	0	1	2	3	4	5
	The	dog	chased	the	cat	.
	Chaser: dog		Chased: cat			
	The	cat	chased	the	dog	.
	Chaser: cat		Chased: dog			

Word boxes are illustrative; a tokenizer may split words into smaller pieces.

RoPE · ROTARY POSITION EMBEDDING

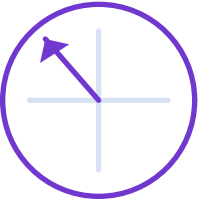
For one attention head: rotate Q and K by position.

QUERY at i



rotate by i

KEY at j

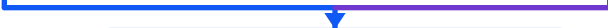


rotate by j

compare



rotated Q · rotated K



Relative distance influences the score.
The causal mask separately blocks future positions.

RoPE is one position method (used by Llama). Other models use learned positions or attention biases.

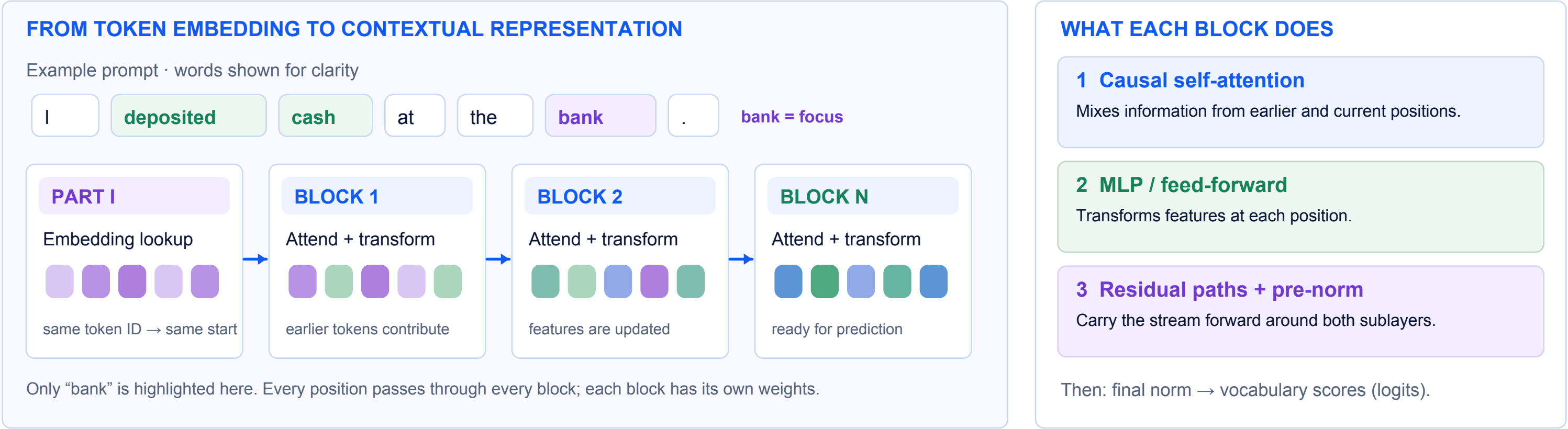
KEY IDEA

Position helps attention distinguish both token content and where it occurs.

[Su et al. · RoFormer / RoPE](#) [Hugging Face · Llama documentation](#)

Layer by Layer

Part I gave us token embeddings: learned starting vectors for token IDs.
Decoder blocks repeatedly update the sequence, making each position sensitive to its available context.



Layer roles are not fixed: there is no universal “grammar layer” or “facts layer”.

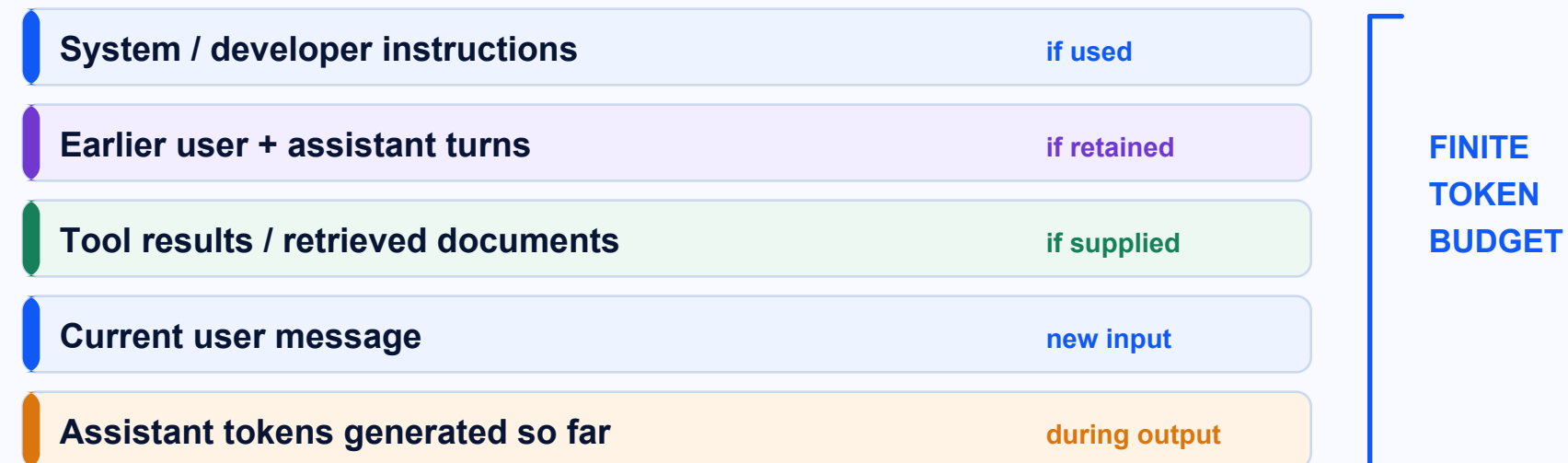
KEY IDEA Embeddings start the sequence; stacked blocks build context-sensitive states for prediction.

Context Window

The model uses the tokens available for this request, including relevant earlier messages when supplied.

The application decides what to send; chat roles and content are encoded in a model-specific token sequence.

WHAT MAY BE IN THE CURRENT CONTEXT?



Older items can be omitted or summarized when the window fills.
The exact limit depends on the model and serving configuration.

CONTEXT WINDOW ≠ LONG-TERM MEMORY

A window is working input for this request.
Prompting does not itself change model weights.
Saved facts return only if the app includes them.

FOR DEVELOPERS

- 1 Budget for output**
Input and generated tokens share the limit.
- 2 Select useful context**
Retrieve relevant text; summarize old turns.
- 3 Check what survives**
Test truncation and verify critical facts.

A larger window holds more tokens but does not guarantee that every detail will be used reliably.

KEY IDEA Your app must select, fit and refresh the information the model needs.

Training vs Inference

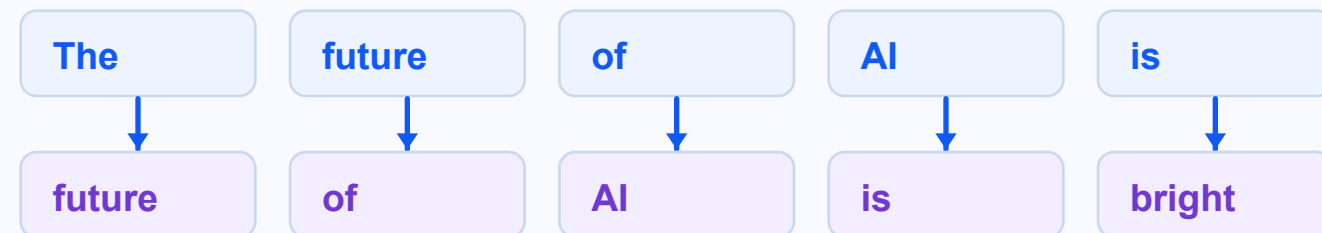
The same decoder model is used to learn from known sequences or to generate from a prompt.

Training compares predictions with known next tokens; inference extends a prompt with new tokens.

TRAINING · LEARN WEIGHTS

Known sequence: “The future of AI is bright”

INPUT POSITIONS



KNOWN NEXT-TOKEN TARGETS

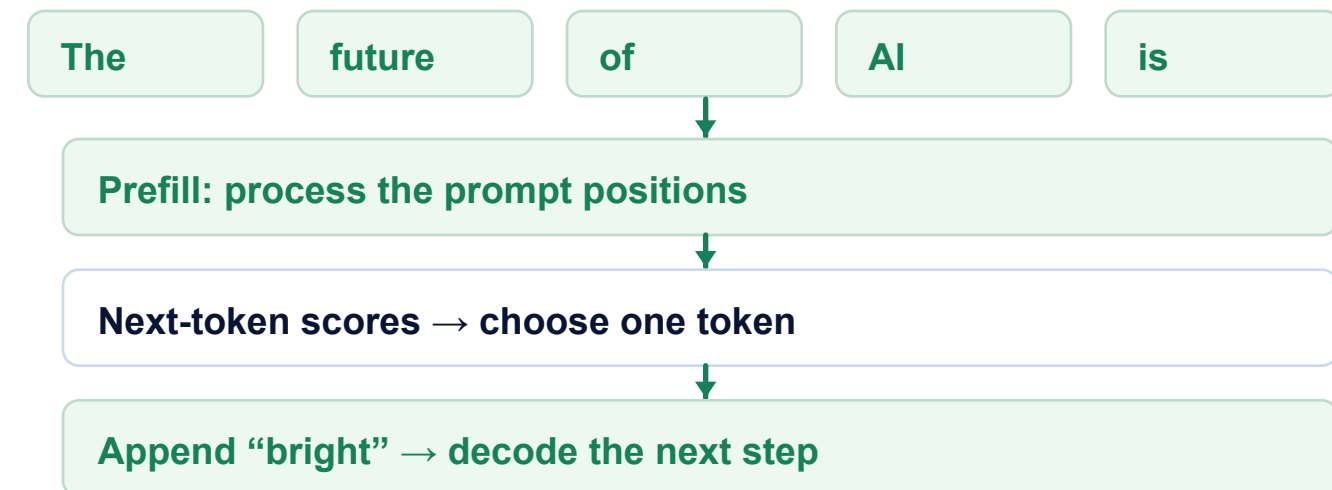
Causal forward pass → scores at all five positions

Compare with targets → loss → backprop → weight update

Each position sees its left prefix; many losses run in parallel.

INFERENCE · USE FIXED WEIGHTS

Prompt: “The future of AI is”



Output tokens arrive sequentially; weights stay fixed.

A KV cache can reuse earlier keys and values.

Word boxes simplify tokenization; “bright” is an example, not a guaranteed prediction.

KEY IDEA

Training updates weights from errors; inference generates with fixed weights, one new token at a time.

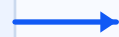
Why LLMs Can Be Wrong

A fluent answer can contain unsupported or incorrect claims.

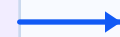
Next-token prediction produces text; factual accuracy needs separate evidence and checks.

WHERE ERRORS CAN ENTER

Question + context



Model scores tokens



Fluent answer

FLUENT ≠ VERIFIED

A claim needs support beyond how likely its words sound.

- 1 Learned patterns** Training text can include mistakes or misconceptions.
- 2 Missing evidence** The needed facts may be absent, old or poorly used.
- 3 Reasoning / generation** A calculation, inference or multi-step answer can fail.

Different tasks and models fail in different ways; test your own use case.

BUILD VERIFIABLE ANSWERS

1

Ground in evidence

Retrieve relevant, trusted source material.

2

Check citations

Match each important claim to the actual source.

3

Use tools

Validate calculations, IDs and structured facts.

4

Evaluate + abstain

Test hard cases; allow “I don’t know”.

A next-token probability is not the probability that a factual statement is true.

KEY IDEA

Treat generated claims as proposals: verify them against trusted evidence and tools.

Key Takeaways & What's Next

We have followed tokens from embeddings through decoder blocks to next-token scores.
Use this mental model—and its limits—when designing AI applications.

THE MODEL IN FOUR IDEAS

01 TOKENS → VECTORS

Part I: token IDs map to learned embeddings.
These are starting vectors for the prompt.

02 LAYERS BUILD CONTEXT

Causal attention and MLPs update vectors.
Norms and residual paths support the stack.

03 PREDICT & REPEAT

Final states produce next-token logits.
Choose a token, append it, then continue.

04 KNOW THE LIMITS

Context is finite; fluent text can be wrong.
Evidence and verification remain essential.

Position methods, head sharing and block details vary by model; inspect its configuration.

Decoder-only models share core ideas, but implementations are not identical.

KEY IDEA

Understand the mechanism; then design the context and checks around it.

WHAT'S NEXT FOR DEVELOPERS

1 Inspect a model

Layers, heads, position method, context limit.

2 Trace one prompt

Token IDs → logits → selected output tokens.

3 Build for reliability

Add relevant evidence, tools and validation.

Then evaluate on real tasks and failure cases.