Transformer Decoder — Step 1

Output Embedding & Positional Encoding

Understanding how the decoder starts generating words, one token at a time

Where We Are in the Transformer

Encoder

Already processed the input sentence

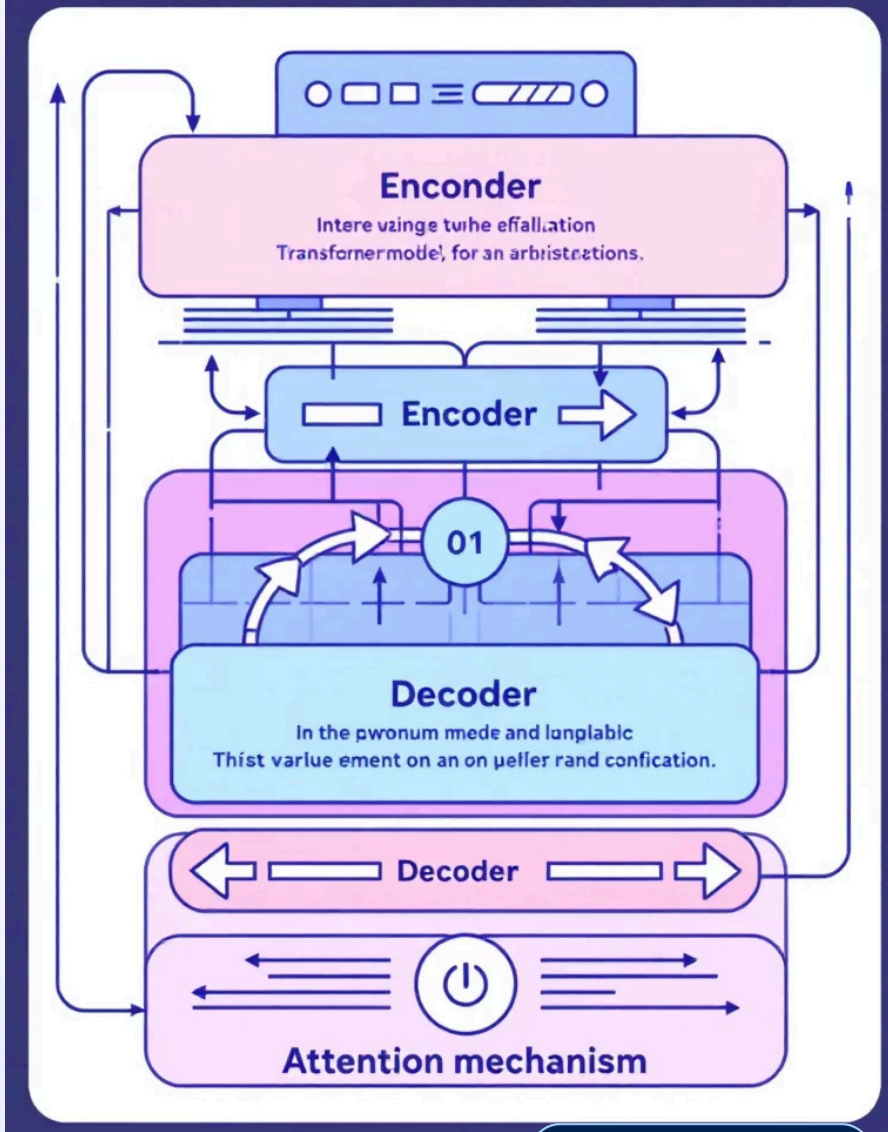
"The cat sat on the mat"

Decoder

Now starts generating the output

"Le chat s'est assis sur le tapis"

The decoder works **one token at a time**, building the output sequence step by step.



Output Embedding

The decoder converts each word (or token) into a vector of numbers — a dense numeric representation that captures semantic meaning.

During **training**, we use the target sentence (like "Le chat..."). During **inference**, we use the generated words so far.

Each embedding vector typically contains 512 or more dimensions, allowing the model to encode rich linguistic information.

Example Embeddings

"Le" → [0.12, -0.08, 0.55, ...]
"chat" → [0.23, -0.19, 0.71, ...]
"s'est"→ [0.31, -0.42, 0.18, ...]

Positional Encoding

The Transformer architecture processes all tokens in parallel, which means it **doesn't inherently know word order**. This is where positional encoding becomes crucial.

01

Add Position Information

We add a positional encoding vector to each word embedding

02

Preserve Sequence Order

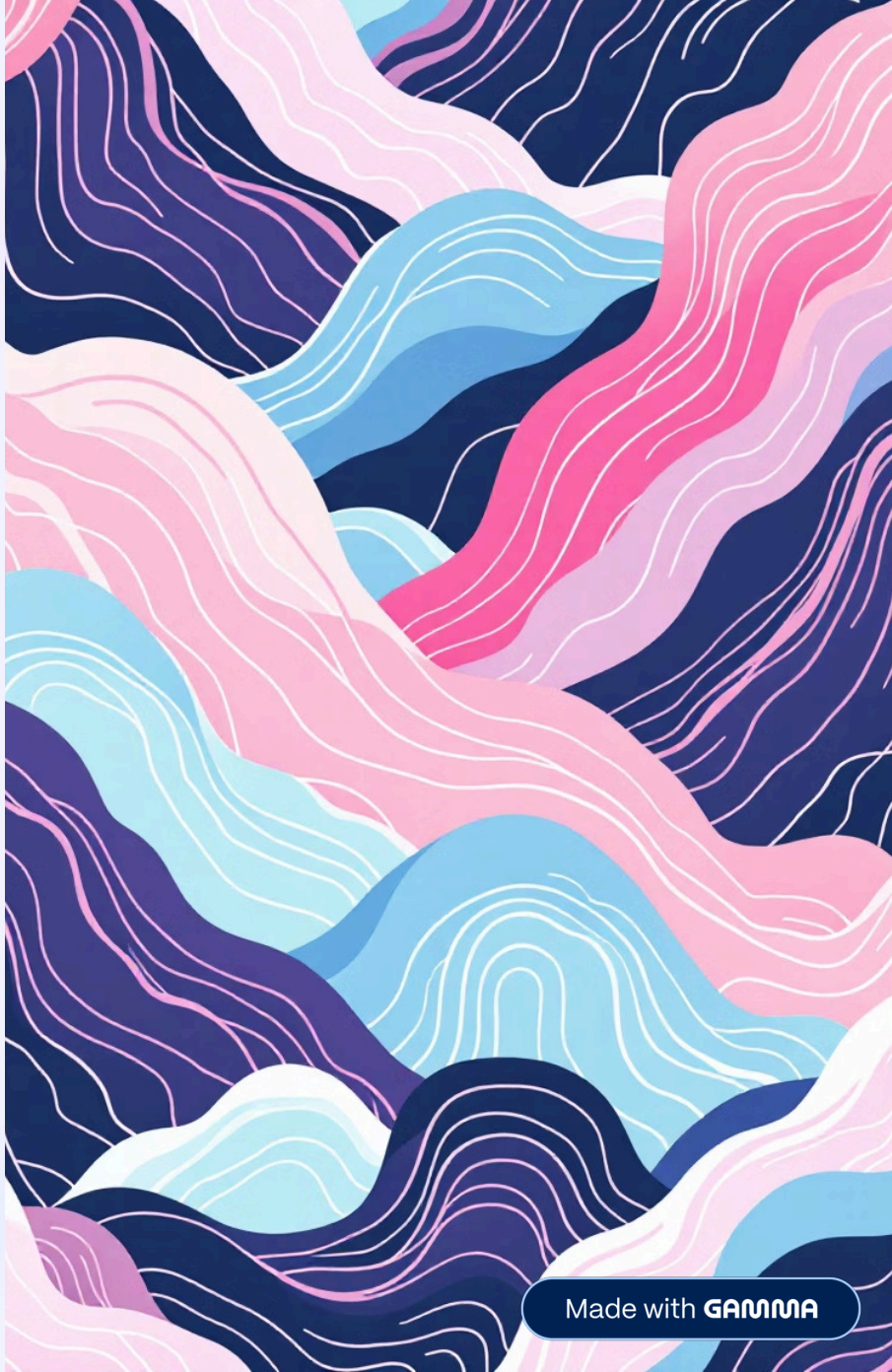
Helps decoder understand that "Le" is position 1, "chat" is position 2, "assis" is position 3

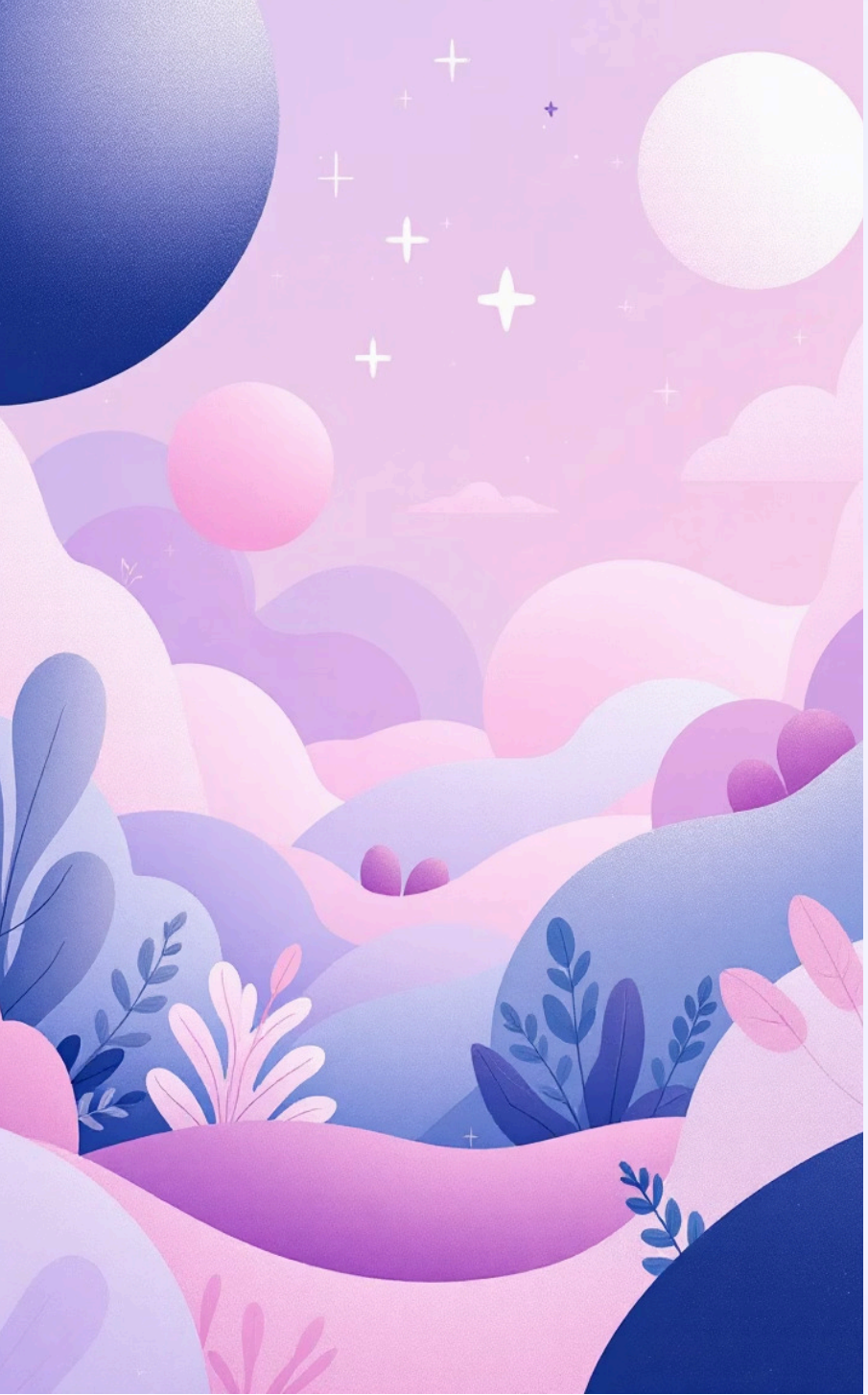
03

Create Final Input

The result combines both meaning and position

$$\text{Input to decoder} = \text{Embedding} + \text{Position vector}$$





Why Add Instead of Concatenate?

Same Dimensions

Both embeddings and positional encodings are the same size (typically 512 dimensions)

Fixed Size

Adding keeps the dimension fixed, preventing the model from growing too large

Blended Information

Each number in the vector now mixes **word meaning** with **word position**

This elegant solution maintains computational efficiency whilst encoding both semantic and positional information in a single, unified representation.

Example — Building Decoder Input

Let's see how embeddings and positional encodings combine to create the actual decoder input:

Position	Token	Embedding	Positional Encoding	Final Input
1	Le	E_1	P_1	$E_1 + P_1$
2	chat	E_2	P_2	$E_2 + P_2$
3	s'est	E_3	P_3	$E_3 + P_3$

> These combined vectors become the inputs to the decoder block, ready for the attention mechanism to process.

Transition — Now the Decoder Generates Output

Embeddings Ready ✓

Word meanings captured in dense vectors

Position Info Added ✓

Sequence order encoded in the input

Next Step

The decoder uses them to **start predicting the next word**

Let's explore how this prediction process differs between training and inference modes.

During Training (Teacher Forcing)

During training, the decoder receives real previous words as input. This technique, called **teacher forcing**, helps the model learn efficiently.

The model learns to predict the **next** word based on the ground truth sequence, allowing it to adjust its weights accurately.

Training Example

Step	Decoder Input	Model Predicts
1	<START>	"Le"
2	<START> Le	"chat"
3	<START> Le chat	"s'est"

🧠 The model sees the correct next word and adjusts its internal weights to improve future predictions, learning language patterns through supervised learning.

During Inference (Generation Time)

During inference, the decoder **doesn't know the correct answer**. It must generate text independently.

The process starts with only the `<START>` token, predicts one word at a time, and feeds each prediction back in as input for the next step.

This continues until the model predicts the `<END>` token or reaches a maximum length.

Inference Example

Step	Decoder Input	Model Predicts
1	<START>	"Le"
2	<START> Le	"chat"
3	<START> Le chat	"s'est"

> 🌀 This autoregressive generation continues iteratively, with each new word becoming part of the input for predicting the subsequent word.

Summary

Concept	Training	Inference
Input	Real target words	Model's own predictions
Guidance	Uses ground truth	No ground truth
Purpose	Learn language patterns	Generate output text

Training Phase

The decoder learns using real words, building knowledge of language patterns

Inference Phase

The decoder generates words one by one, applying learnt patterns to create new text

This two-stage process enables transformers to both learn from data and generate coherent, contextually appropriate sequences during deployment.