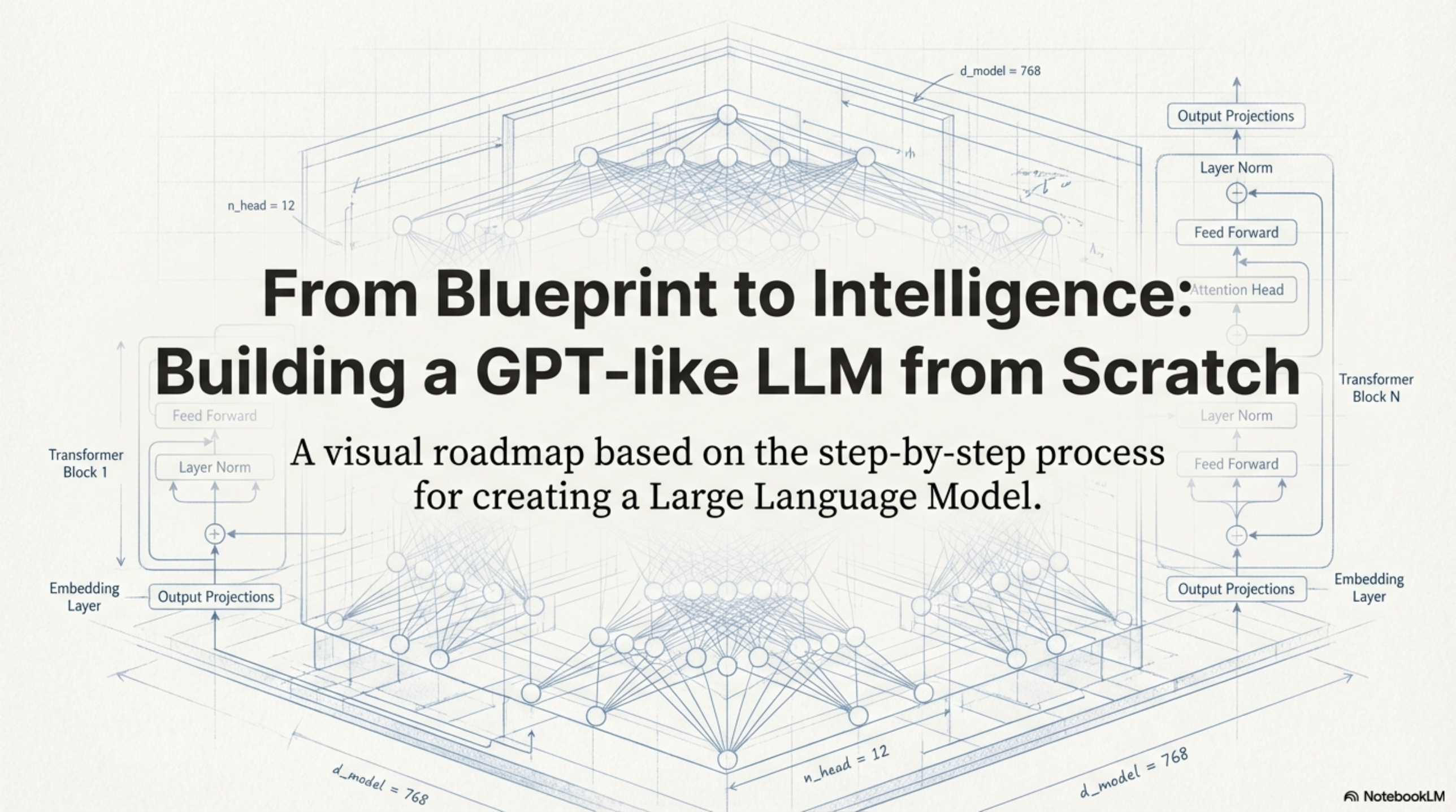
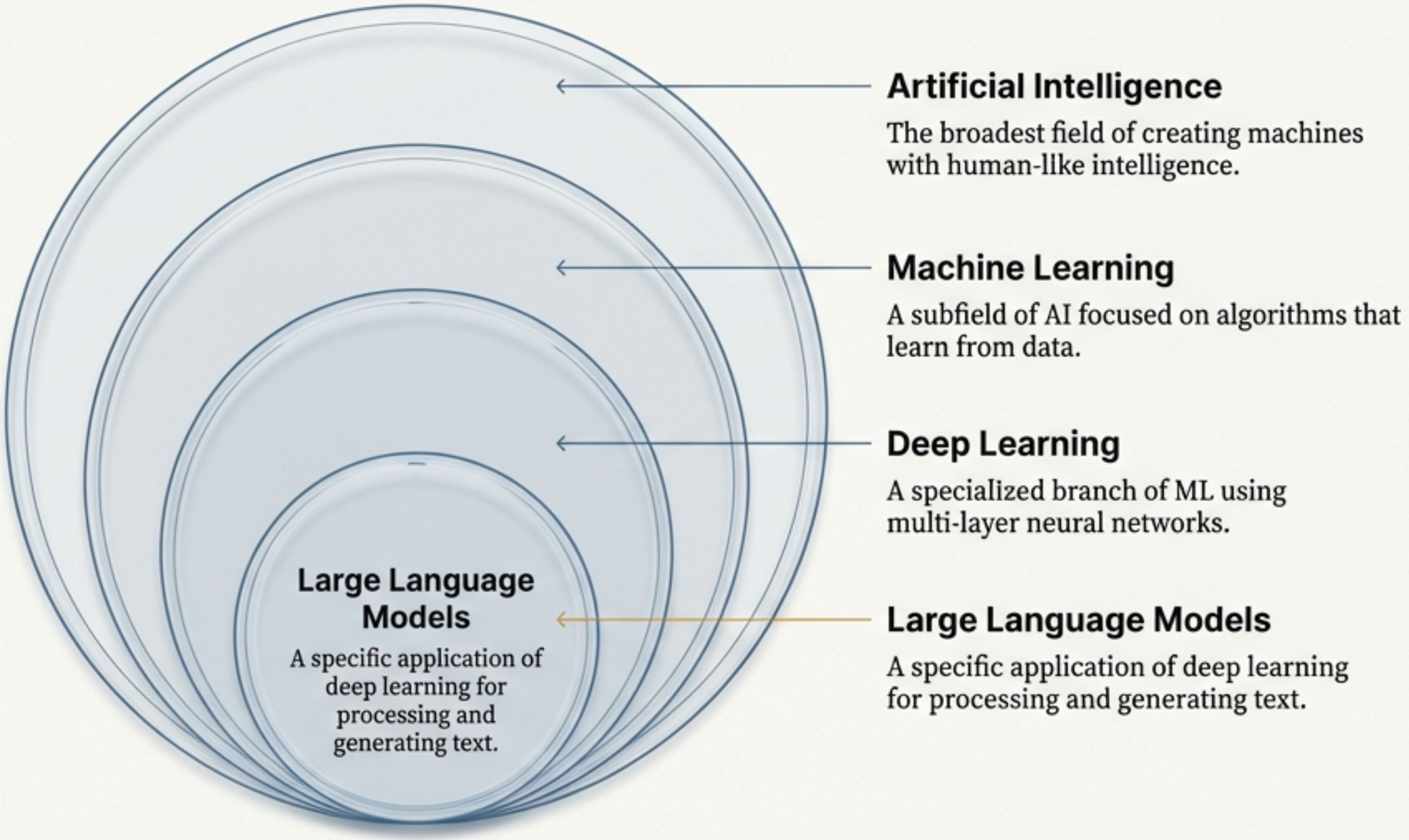



# From Blueprint to Intelligence: Building a GPT-like LLM from Scratch

A visual roadmap based on the step-by-step process for creating a Large Language Model.

# An LLM is a Deep Learning Model Trained on Massive Text Datasets

Large Language Models (LLMs) are deep neural networks designed to understand and generate human-like text. The 'large' refers to both the model's size (billions of parameters) and the immense datasets they are trained on. Their core architecture, the Transformer, allows them to process language with a nuanced understanding of context.

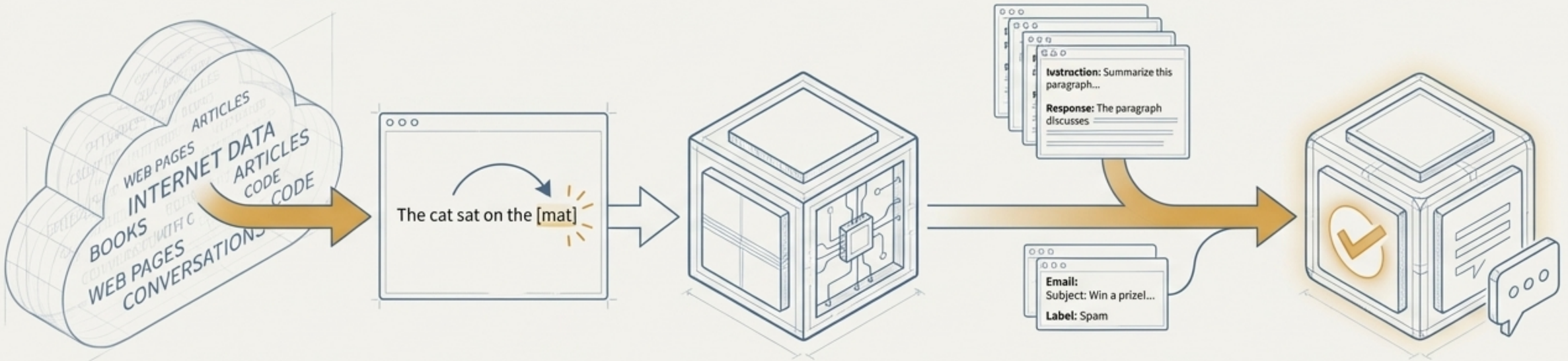



---

LLMs are a specialized subset of AI, built on the foundations of machine learning and deep learning. They learn patterns from data without explicit programming.

# The LLM Lifecycle: Broad Pretraining, Followed by Specific Finetuning

Building and deploying a capable LLM is a two-stage process.




## PRETRAINING PROCESS

**Goal:** To build a general understanding of language.

**Process:** The model is trained on a vast, diverse corpus of unlabeled text. The task is simple: next-word prediction.

## BASE LLM

**Outcome:** A "base" or "foundation" model with broad knowledge and text completion abilities.

## FINETUNING PROCESS

**Goal:** To specialize the model for specific tasks.

**Process:** The pretrained model is further trained on a smaller, labeled dataset tailored to a specific domain or task.

## FINETUNED LLM

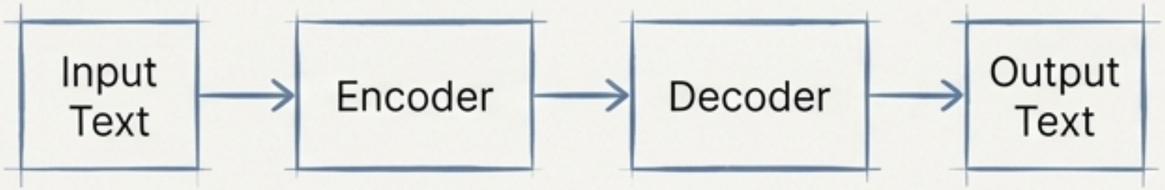
**Examples:**
**Classification:** Finetuning on emails labeled as spam/not-spam.

**Instruction-Following:** Finetuning on instruction-response pairs to create a helpful assistant.

# Our Blueprint: The Decoder-Only GPT Architecture

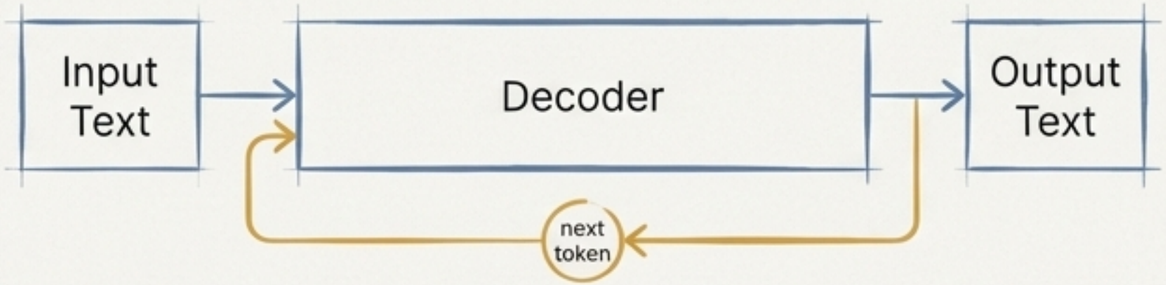
Most modern LLMs are based on the Transformer architecture, originally designed for machine translation with an Encoder-Decoder structure. GPT (Generative Pretrained Transformer) models simplify this by using only the Decoder component.

## The Original Transformer




The encoder creates a numerical representation of the input text. The decoder uses this representation to generate the output sequence. Suited for sequence-to-sequence tasks like translation.

## The GPT Architecture

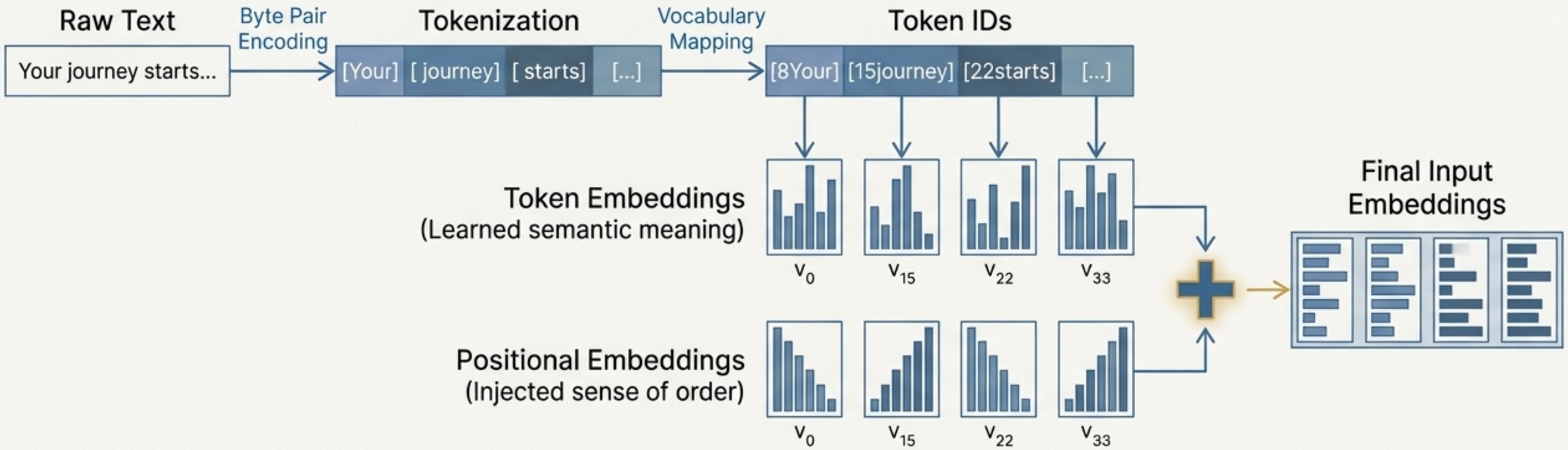



The decoder-only structure is inherently autoregressive, generating text one token at a time by predicting the next token based on all previous ones. This unidirectional design makes it ideal for generative tasks.

**We will be building a GPT-like model, focusing exclusively on the powerful decoder part of the original Transformer.**

# Step 1: Encoding Language and Position into Vectors

An LLM cannot process raw text. The first step is to convert a sequence of words into a sequence of numerical vectors that encode both meaning and order.




The final input to the model is a matrix where each row is a vector that combines a token's meaning with its position in the sequence.

# The Core Engine: Self-Attention for Context-Awareness

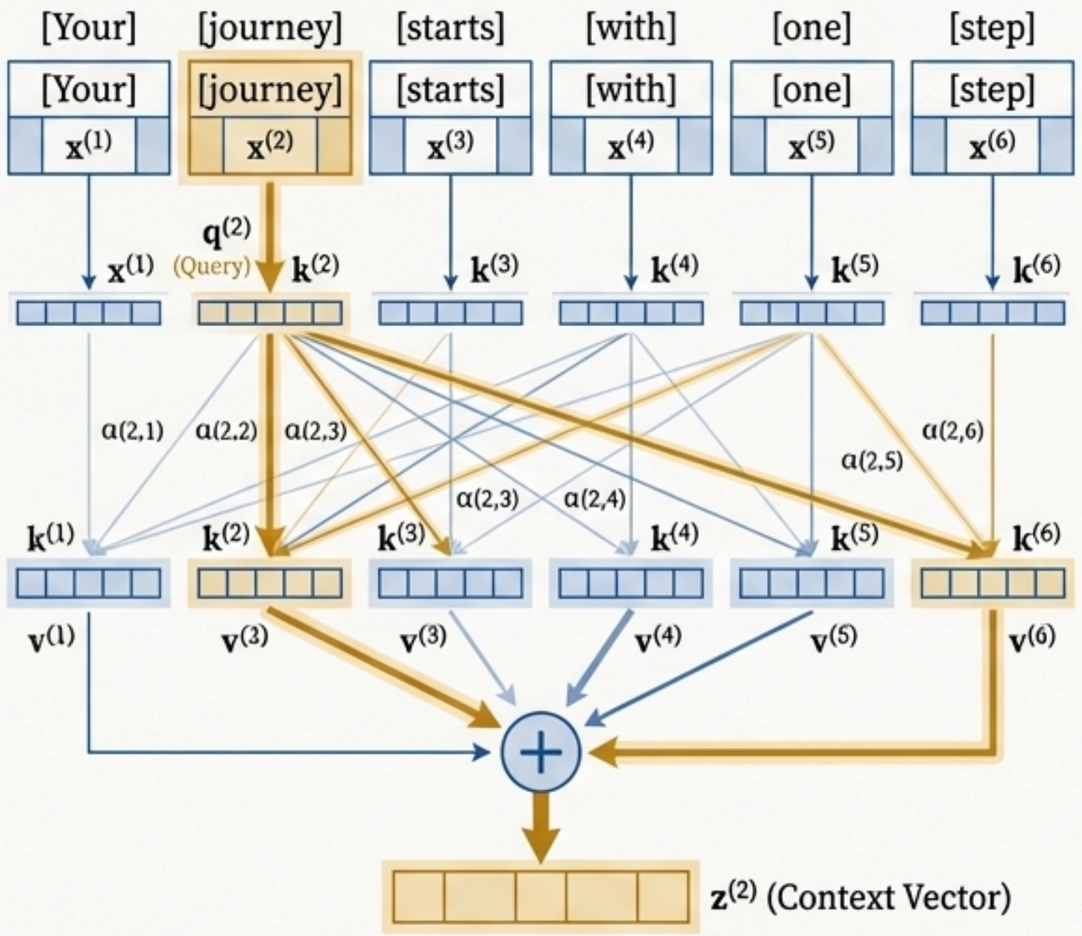
## The Problem & Solution

### The Problem: Contextual Ambiguity

A standard token embedding for "bank" is the same in "river bank" and "investment bank." How can the model differentiate?

### The Solution: Self-Attention

A mechanism that allows each token to look at all other tokens in the input sequence to compute a new, context-aware representation, called a **context vector.**




*To compute the context vector for a given token, its* **Query** *vector is compared against the* **Key** *vectors of all other tokens. The similarity scores (attention weights) determine how much of each token's* **Value** *vector contributes to the final context vector.*

## The Key, Query, and Value Analogy

### The QKV Analogy

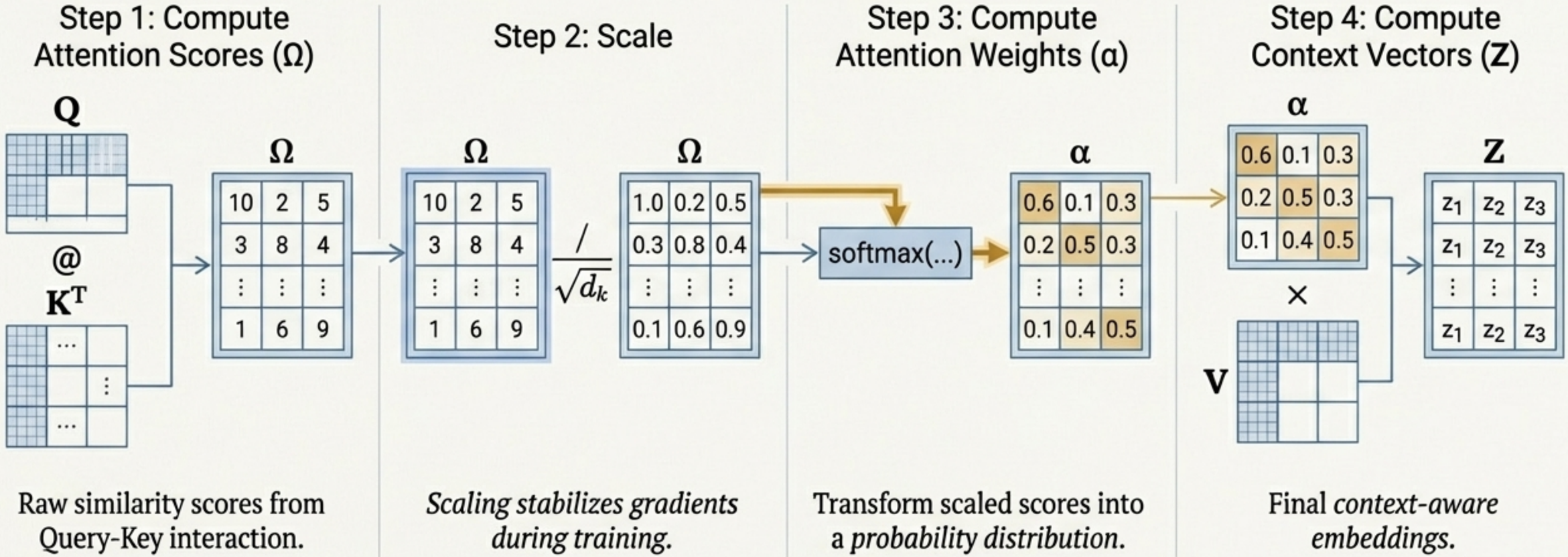
**Query (Q):** What I am looking for. (Represents the current token's focus).

**Key (K):** What I contain. (Acts as a label for other tokens to match against).

**Value (V):** What I actually am. (The real content of a token).

# The Math of Self-Attention: Scaled Dot-Product

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q} \mathbin{@} \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbin{@} \mathbf{V}$$

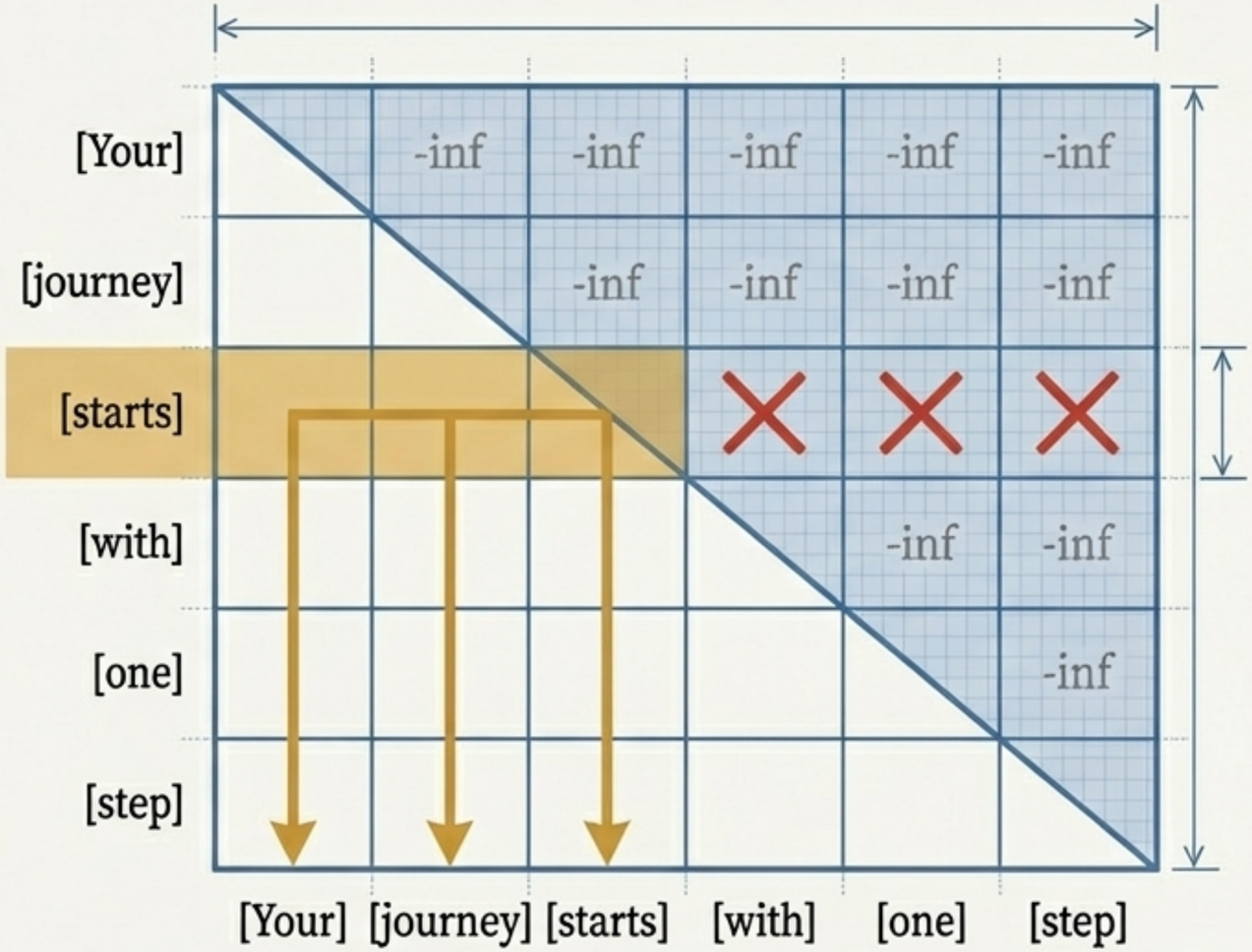


# Causal Attention: Preventing the Model from Seeing the Future

The Need for a Mask: For a generative task like next-word prediction, the model must be **autoregressive**. When predicting the token at position i, it should only have access to information from tokens at positions 1 to i, not i+1 or beyond.

The Mechanism: We apply a **"causal mask"** to the attention scores matrix before the softmax step. This mask sets all values in the upper triangle of the matrix (representing connections to future tokens) to negative infinity (-inf).

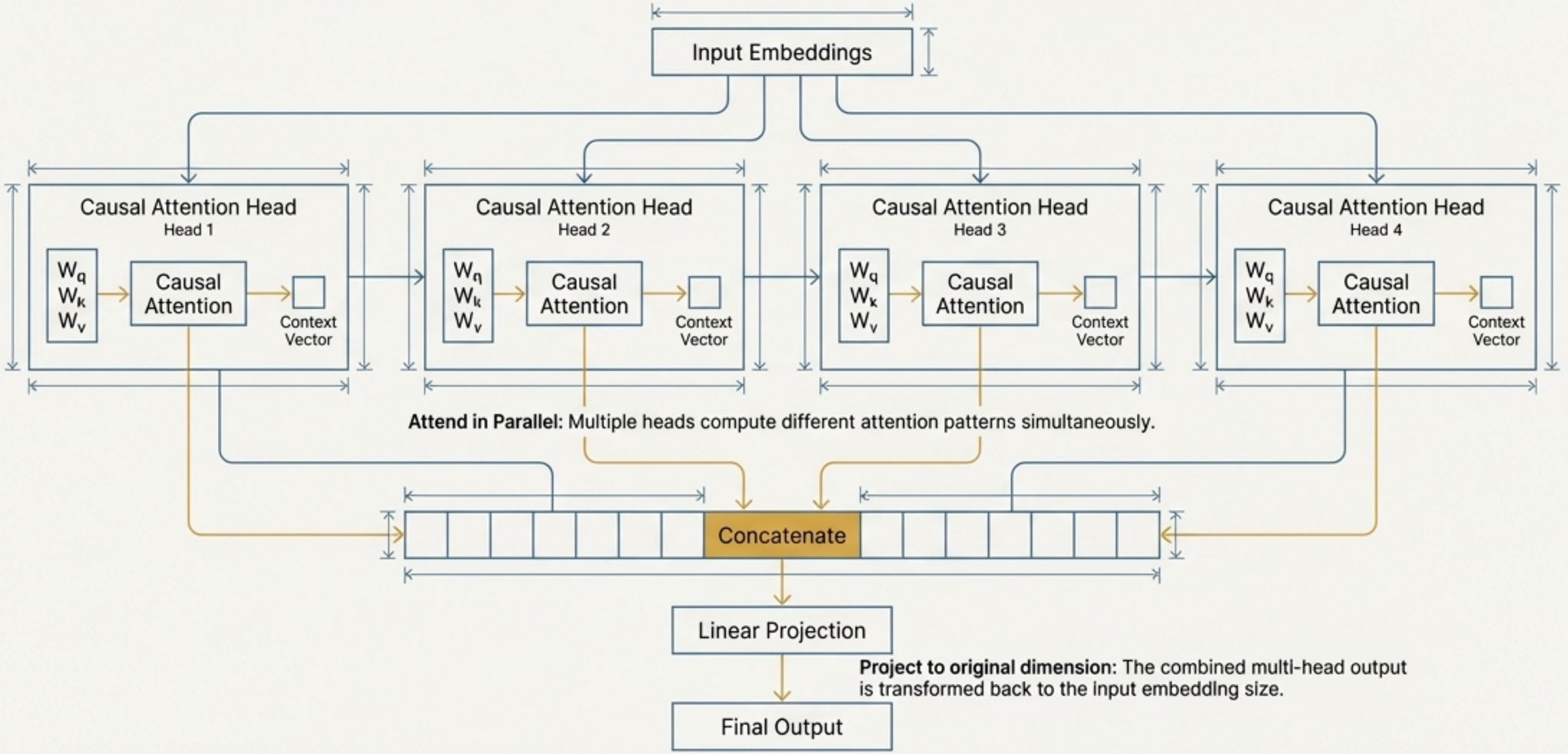
The Effect: When the softmax function is applied, $e^{(-inf)}$ becomes 0. This ensures that each token can only assign attention weights to itself and preceding tokens.




*The token "starts" can only attend to itself and previous tokens.*

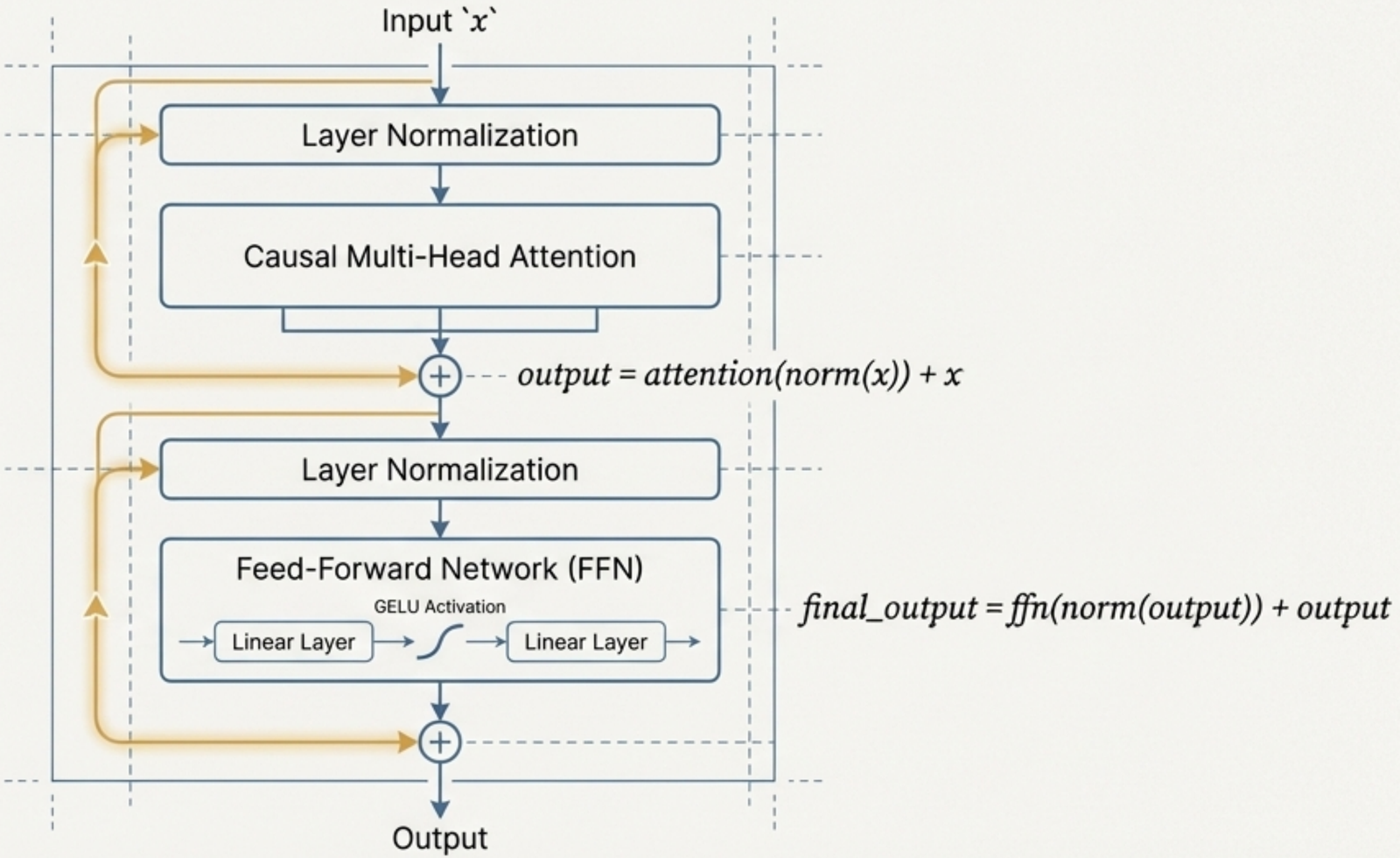
# Scaling Up to Multi-Head Attention

A single self-attention mechanism might learn to focus on one type of relationship (e.g., subject-verb).
Multi-Head Attention allows the model to learn different relationships in parallel.
It's like having multiple 'experts' analyzing the input from different perspectives.



# Assembling the Transformer Block

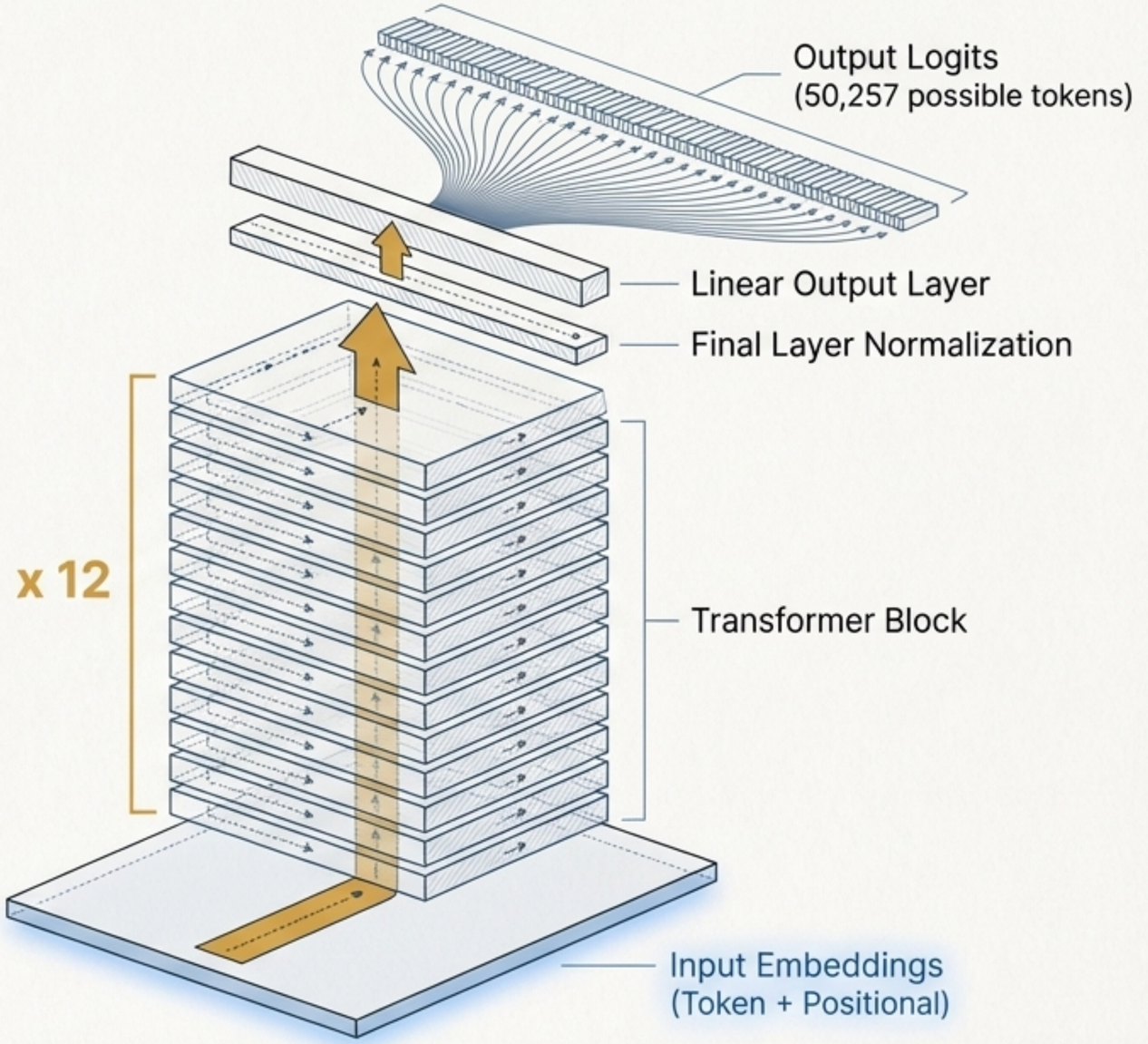
The Transformer Block is the core, repeatable unit of a GPT model. It combines the attention mechanism with a feed-forward network to process and refine the input sequence.



# Building a Deep LLM by Stacking Transformer Blocks

The power of a GPT model comes from its depth. We create this depth by stacking the Transformer Blocks one after another. The output of one block becomes the input for the next.

| Example: GPT-2 (124M parameters) | |
|---|---|
| Vocabulary Size: Source Serif Pro | 50,257 |
| Context Length: Source Serif Pro | 1024 tokens |
| Embedding Dimension: Source Serif Pro | 768 |
| Number of Layers (Blocks): Source Serif Pro | 12 |
| Number of Attention Heads: Source Serif Pro | 12 |



# Awakening Stage 1: Pretraining with Next-Word Prediction

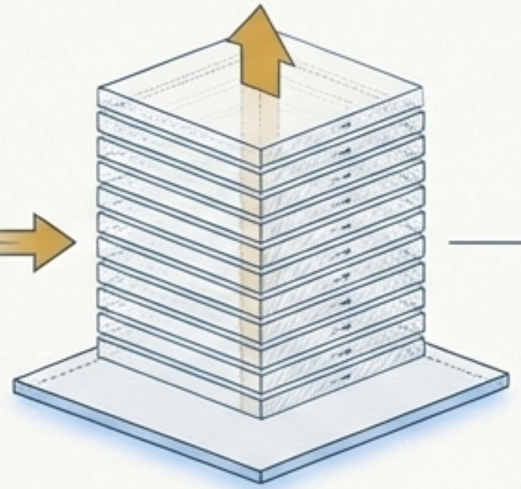
**Goal:**

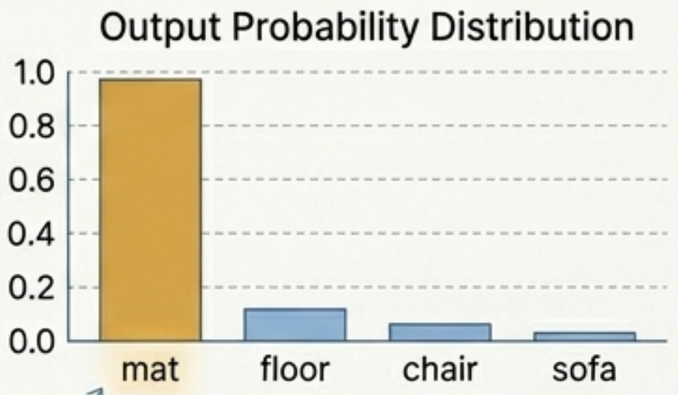
To teach the model the statistical patterns of human language—grammar, facts, reasoning, and context.

**The Task: Self-Supervised Learning:**

The model is trained on a massive, unlabeled text corpus. The learning task is incredibly simple: given a sequence of tokens, predict the very next token.

The cat sat on the






*For the input "The cat sat on the", the text itself provides the target label: "mat".*

**The Scale**

This stage is computationally massive. GPT-3's pretraining dataset included sources like CommonCrawl (410 billion tokens) and Wikipedia (3 billion tokens). The total training compute was estimated at $4.6 million.

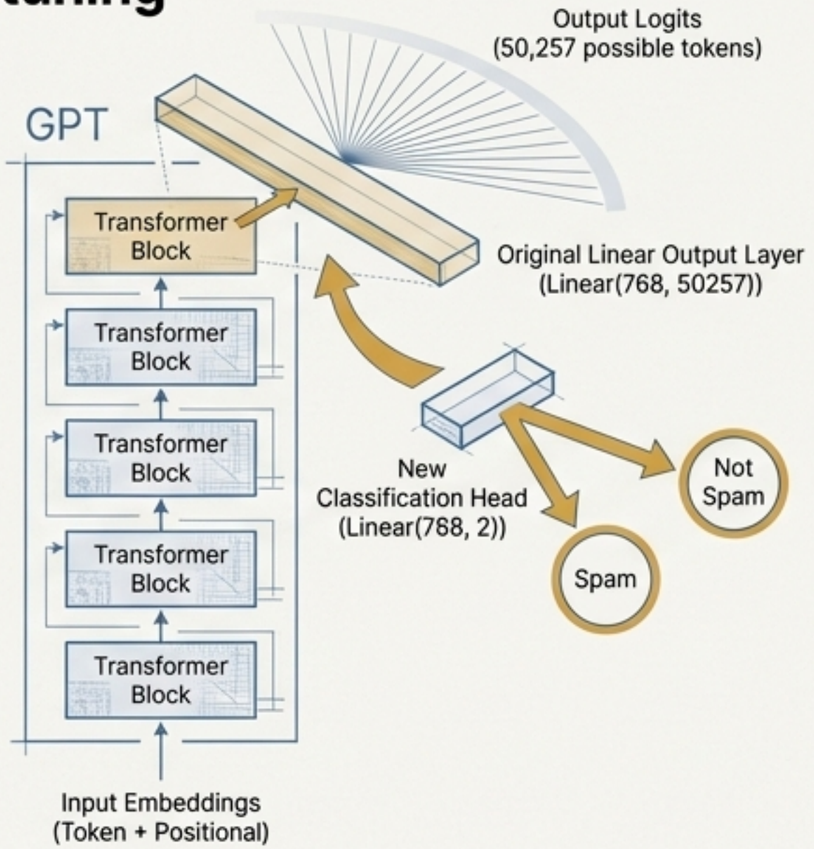
# Awakening Stage 2: Finetuning for Specialized Tasks

After pretraining, the general-purpose "base model" is adapted to specific applications using smaller, task-oriented datasets.

## Classification Finetuning

**Goal:**
To train the model to categorize text.

**Method:**
The final output layer is replaced with a new, smaller classification head. Only this new head and a few top layers of the model are trained on a labeled dataset.




## Instruction Finetuning

**Goal:**
To train the model to be a helpful assistant that follows commands.

**Method:**
The model is trained on a dataset of instruction-response pairs, formatted with a specific prompt style. The model learns to generate the appropriate `### Response:` given an `### Instruction:`.

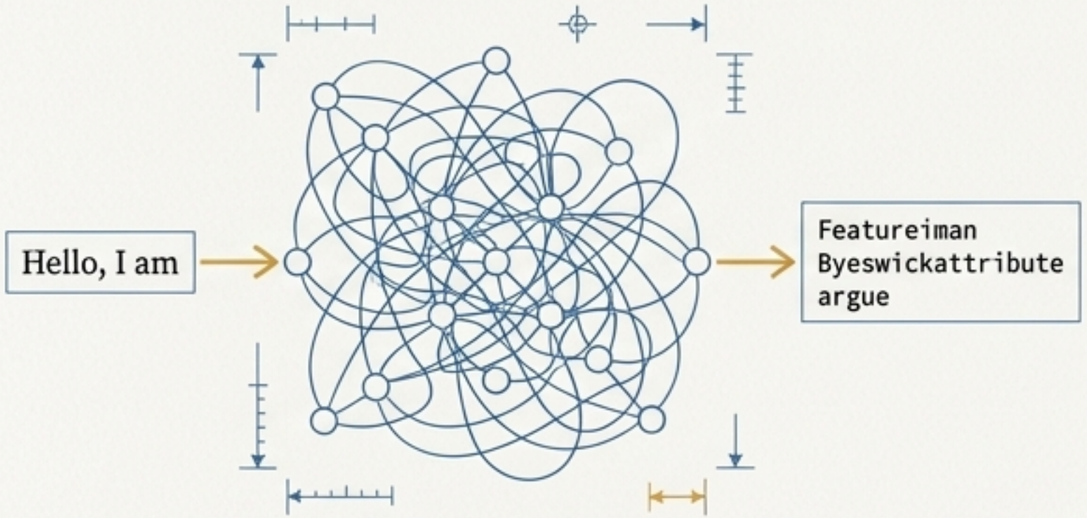
```
### Instruction:
What is an antonym of 'complicated'?

### Response:
An antonym of 'complicated' is 'simple'.
```

# The Transformation: From Random Weights to Coherent Generation

An untrained LLM, with its weights initialized randomly, produces incoherent gibberish. The training process organizes these weights, enabling the model to generate meaningful and contextually relevant text.

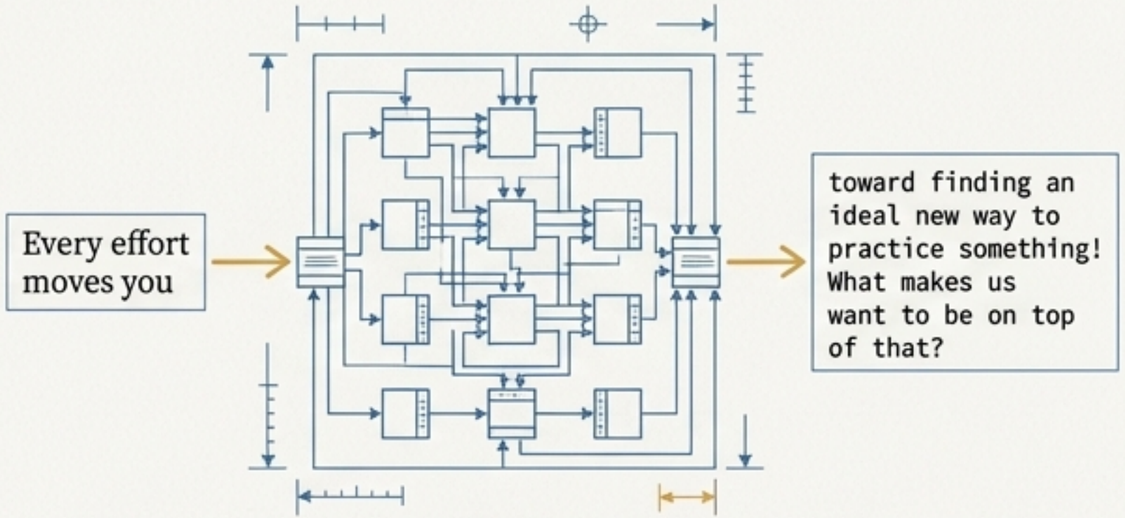
## Before Training: Untrained Model Output




Random weights

The output is nonsensical, reflecting the initial random state of the model's parameters.

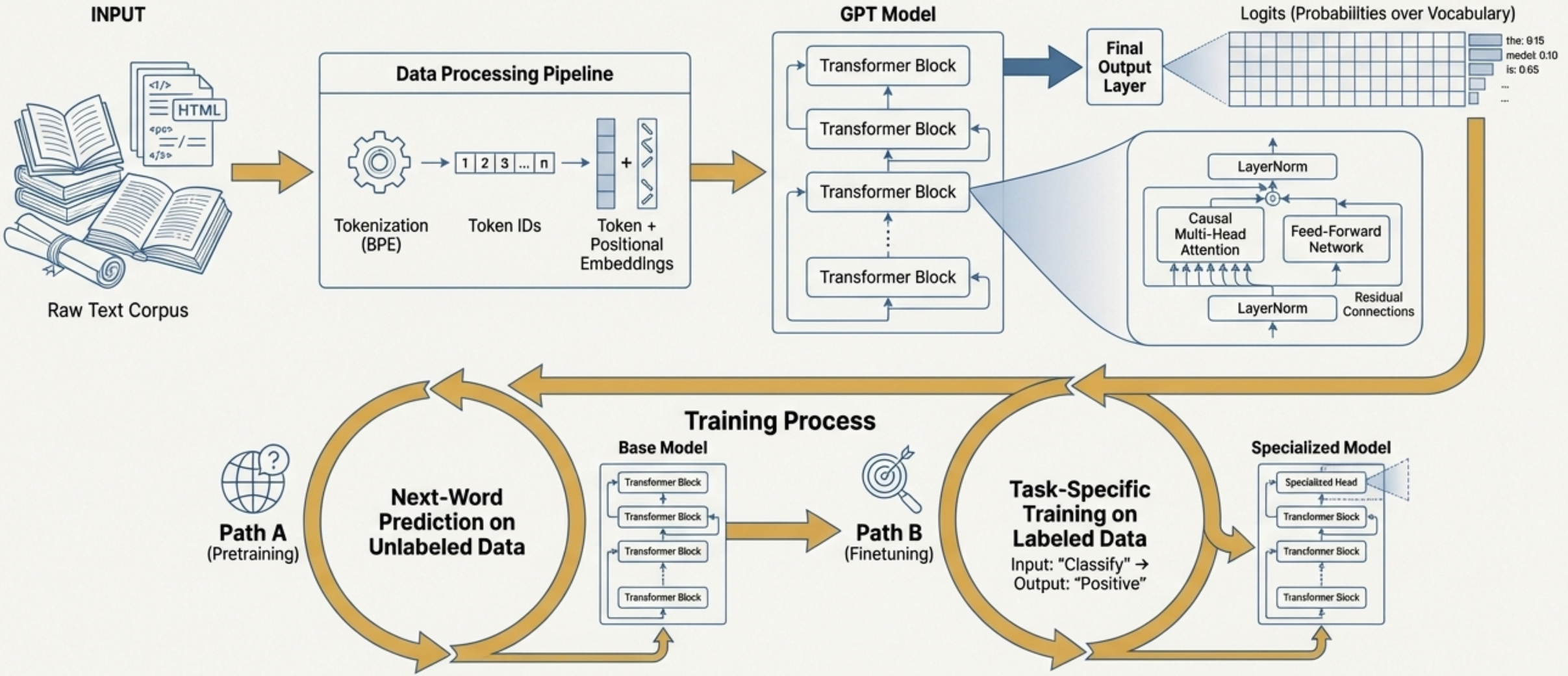
## After Training: Pretrained Model Output




Trained weights

The model now generates grammatically correct, coherent, and contextually plausible text, demonstrating the knowledge gained during pretraining.

# The Complete Blueprint: From Raw Text to a Trained LLM




Every component, from data encoding to the stacked transformer architecture, plays a crucial role in transforming a simple next-word prediction task into a model capable of complex language understanding and generation.