Natural Language Processing (NLP)

Causal Language Modelling

Large Language Models

By:

Dr. Zohair Ahmed

Probability of the Whole Sentence (The Dream)

- Imagine a sentence: **"I love NLP"**
- Question: What is the probability that a random person says exactly this sentence?
- We write it as: $\boldsymbol{P}(\textbf{\textit{I love NLP}})$
- This is the **joint probability** of all four words together.
- But computers don't know this number directly, it's too hard to learn millions of full sentences.
- So we use a simple trick → break it into baby steps using the **chain rule**.

Chain Rule of Probability (The Magic Trick)

- Chain rule says:
- **Probability of everything = multiply probabilities step-by-step, using what already happened**
- In words:
- $P(I\ love\ NLP) = \underbrace{P(I)}_{1st\ word} \times \underbrace{P(love \mid I)}_{after\ seeing\ "I"} \times \underbrace{P(NLP \mid I\ love)}_{after\ seeing\ "I\ love"}$
- That’s it! Just multiply small conditional probabilities.

Chain Rule of Probability (The Magic Trick)

Piece	Meaning in real life	Easy Example
$\mathbf{P}(\mathbf{I})$	Probability that sentence starts with "I"	Quite high in English (many sentences start with I)
$\mathbf{P}(\textbf{love} \mid \textbf{I})$	After saying "I", probability next word is "love"	Very high ("I love..." is common)
$\mathbf{P}(\textbf{NLP} \mid \textbf{I love})$	After "I love", probability next word is "NLP"	High only if you are in this class

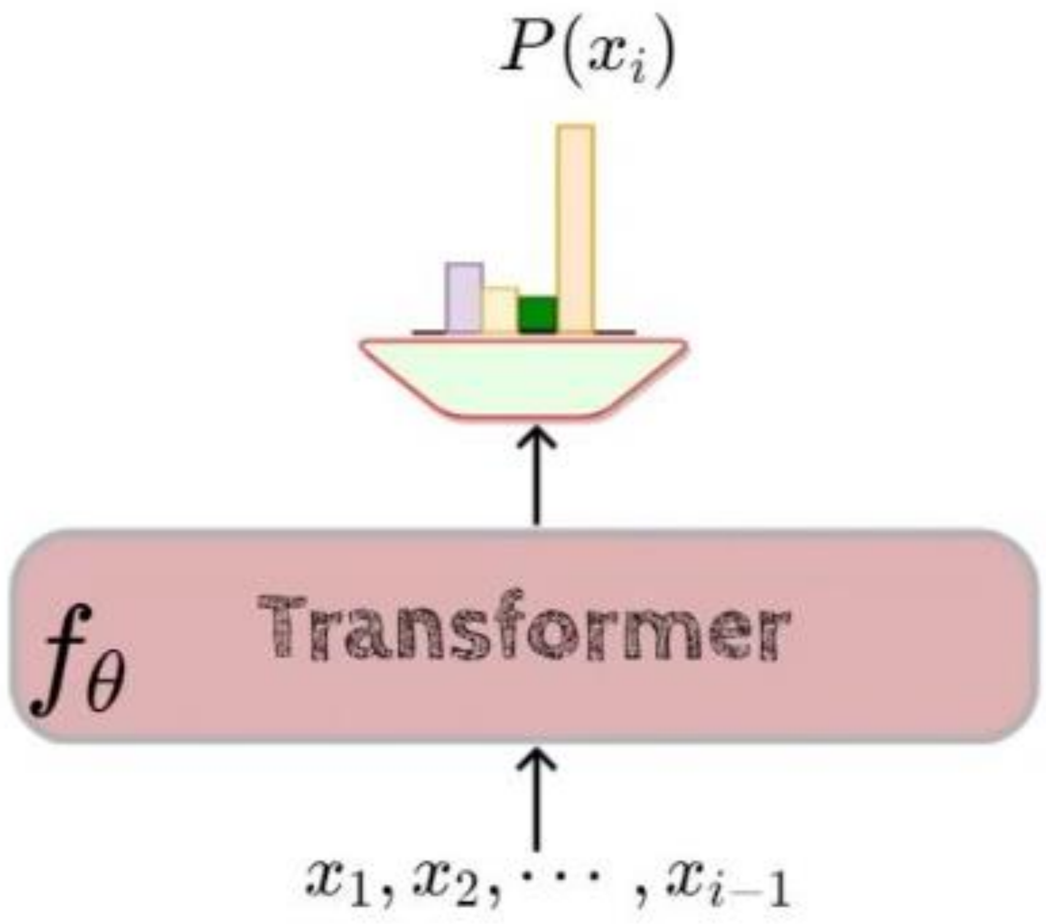
Marginal Distribution = The First One

- The very first probability $P(I)$has **no condition** nothing came before it.
- We call this a **marginal probability** (or marginal distribution).
- All the others are **conditional probabilities**.
- **Chain rule** turns one big impossible probability into many small easy ones.
- The very first one (with no history) is called the **marginal distribution**."

Position	Name	Has condition?	Example
1st word	Marginal distribution	No	$P(\text{I})$
2nd word	Conditional	Yes	$P(\text{love} \mid \text{I})$
3rd word	Conditional	Yes	$P(\text{NLP} \mid \text{I love})$

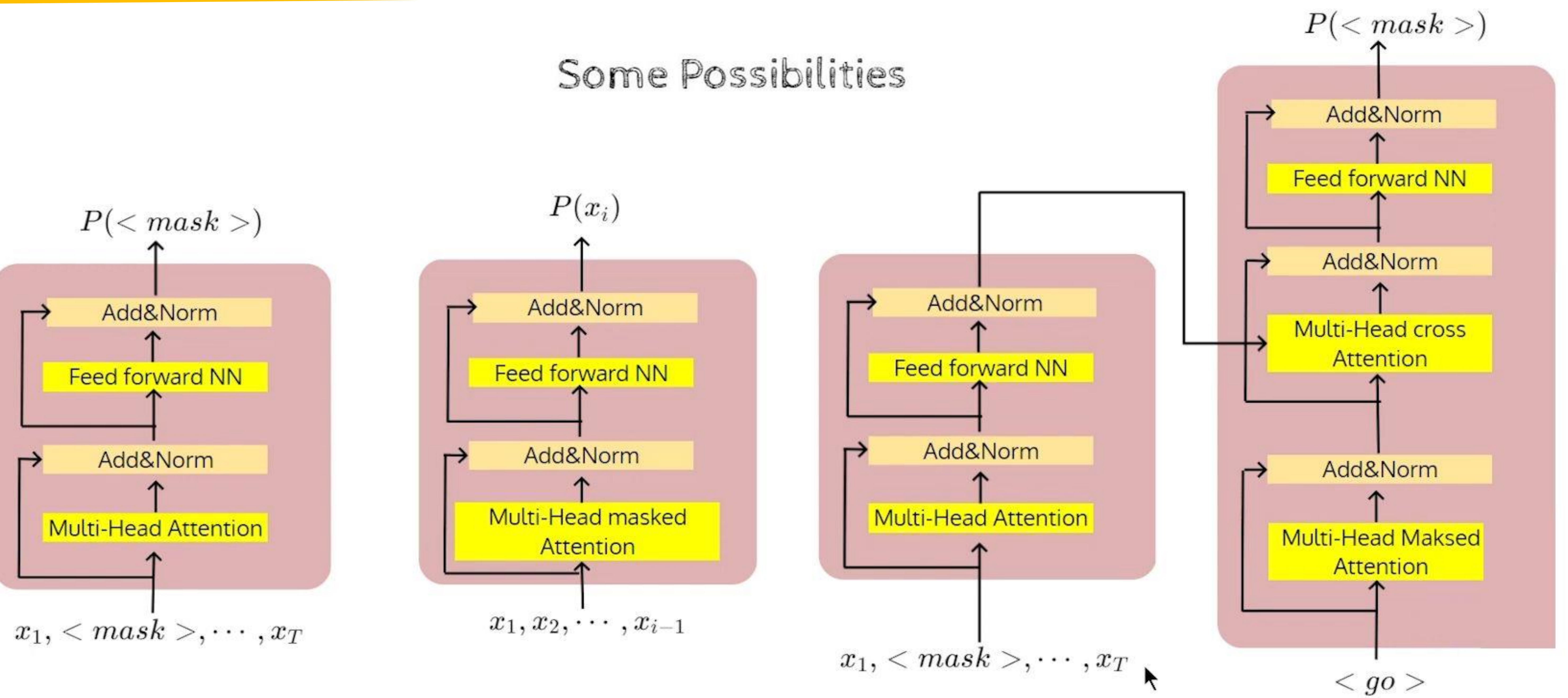
Causal Language Modeling

- Predict the probability **distribution over the vocabulary** for the next token in a sequence.
- Can Transformer be a function to predict **distribution over the vocabulary?**
- This follows the chain rule of probability: $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P\left(x_i, |, x_1 \ldots x_{i-1}\right)$,where $P(x_1)$is the marginal distribution.
- Building on machine translation Transformers (encoder-decoder architecture).
- Exploring Transformers as a function to estimate these distributions.
- Ensure the model only accesses past and present tokens (causality), not future ones, to simulate real-world generation.
- Causal LM is foundational for tasks like text generation, chatbots, and code completion.
- It differs from masked LM (e.g., BERT) by being unidirectional.



Transformer Variants for Language Modeling

Some Possibilities



Using only the encoder of the transformer (encoder only models)

Using only the decoder of the transformer (decoder only models)

Transformer Variants for Language Modeling

- **Three Possibilities for LM Architectures**:
- **Encoder-Only**: Bidirectional context; suitable for understanding tasks but not generation (e.g., BERT-style).
- **Decoder-Only**: Unidirectional **(causal);** focuses on **auto-regressive** prediction (e.g., GPT series).
- **Encoder-Decoder**: Combines both for seq2seq tasks like translation, but adaptable for conditional generation.
- **Focus of This Lecture**: Decoder-only models (informally known as such in the community).
- Derived from the decoder part of the vanilla Transformer used in translation.
- **Why Decoder-Only?**: Efficient for generation tasks; no need for a separate encoder when the input is the sequence itself.
- Decoder-only models scale well with parameters (e.g., billions in modern LLMs) and are trained on massive text corpora for next-token prediction.

Anatomy of the Vanilla Transformer Decoder

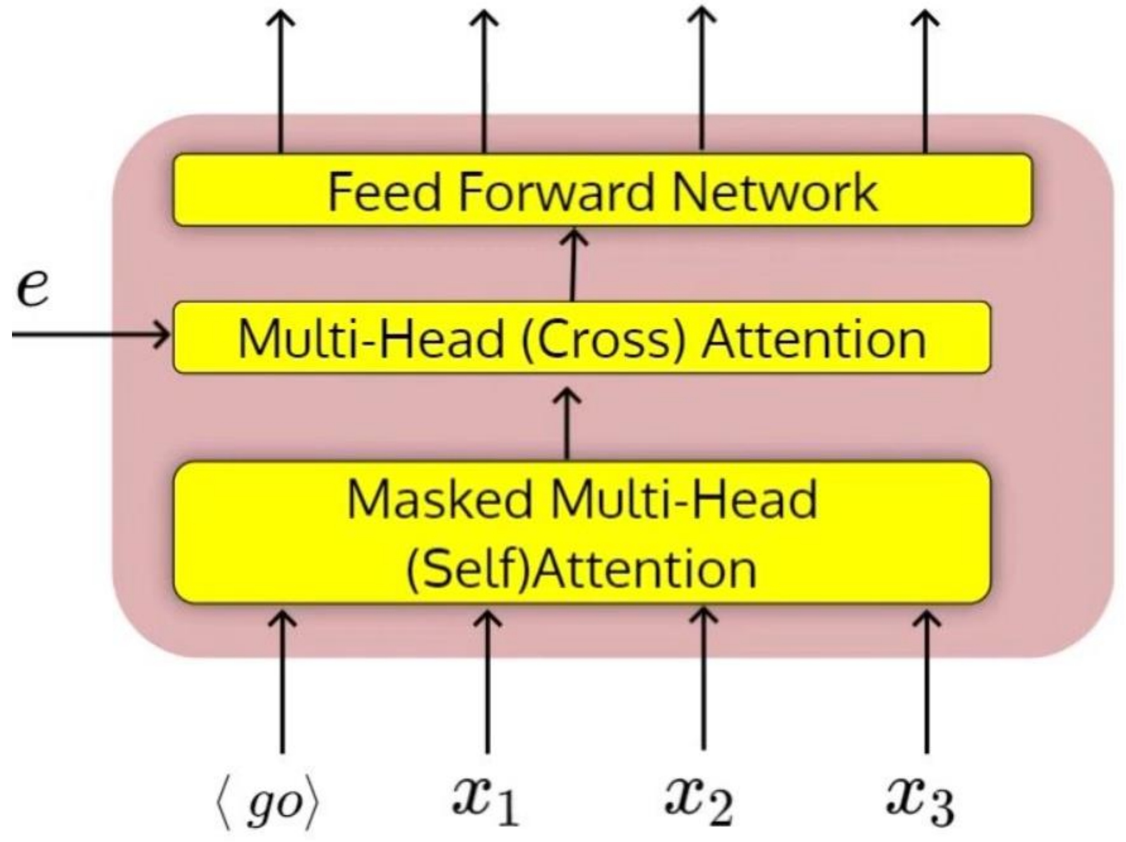
- **Components from Translation Model**:
- **Masked Multi-Head Self-Attention**: Attends to previous tokens in the decoder sequence.
- **Cross-Attention**: Attends to encoder outputs (e.g., source sentence in translation).
- **Feed-Forward Network (FFN)**: Position-wise dense layers for non-linear transformations.
- Layer normalization and residual connections between each sub-layer.
- **Input Sequence**: A sequence of tokens (words or subwords) embedded into vectors.
- Positional encodings added to preserve order (e.g., sinusoidal or learned).
- **Task Adaptation for LM**: Input is a growing sequence; predict next token iteratively.
- Starts from a special token (e.g., <BOS> or "go") for the first prediction.
- Each layer is stacked attention heads allow parallel computation of different relationships.

Ensuring Causality - Masking in Self-Attention

- During training, the full sequence is available, but the model must not "cheat" by seeing future tokens.
- In inference, future tokens are unknown; user provides a prompt (e.g., "I am going to") and expects completion.
- **Solution: Causal Masking**:
- In self-attention: Compute queries (Q), keys (K), values (V) as usual.
- Attention scores: scores $= \frac{QK^T}{\sqrt{d_k}}$.
- **Apply mask:** Add a lower-triangular matrix with $-\infty$ above the diagonal before softmax.
- Mask matrix example (for sequence length 4): $\begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$

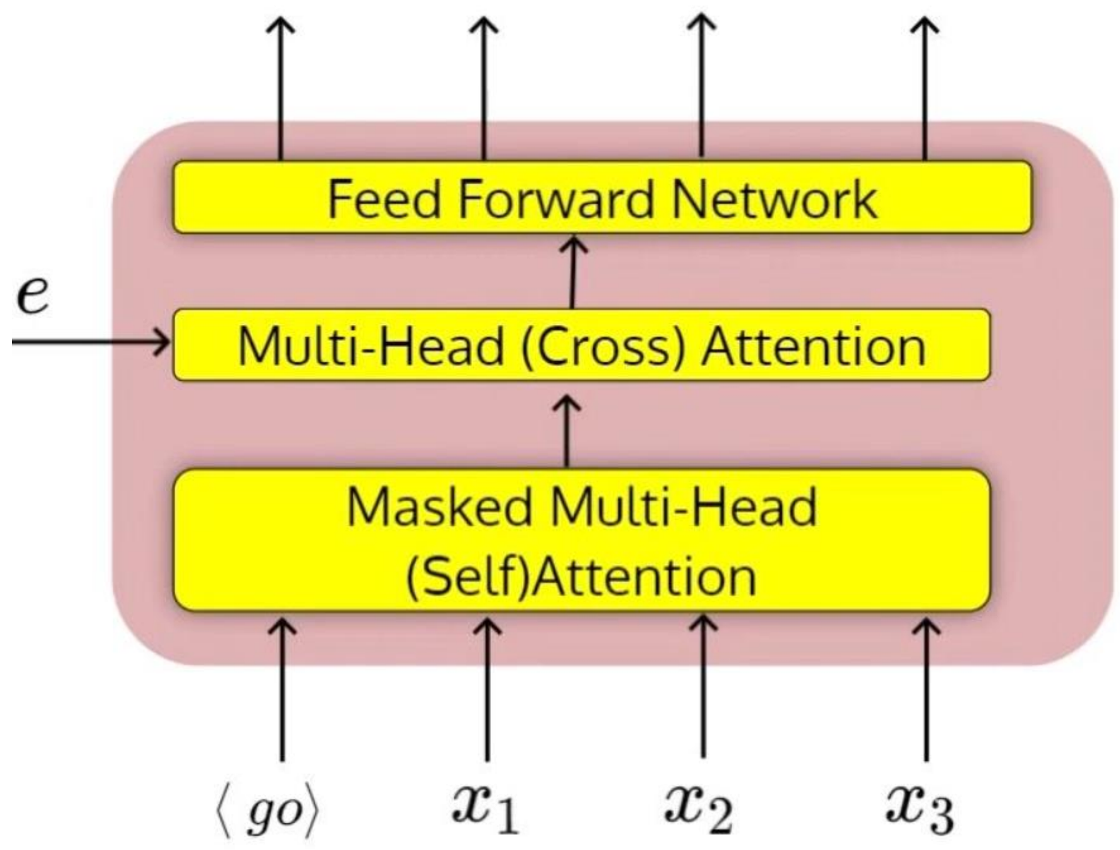
Post-softmax: Future weights become 0.

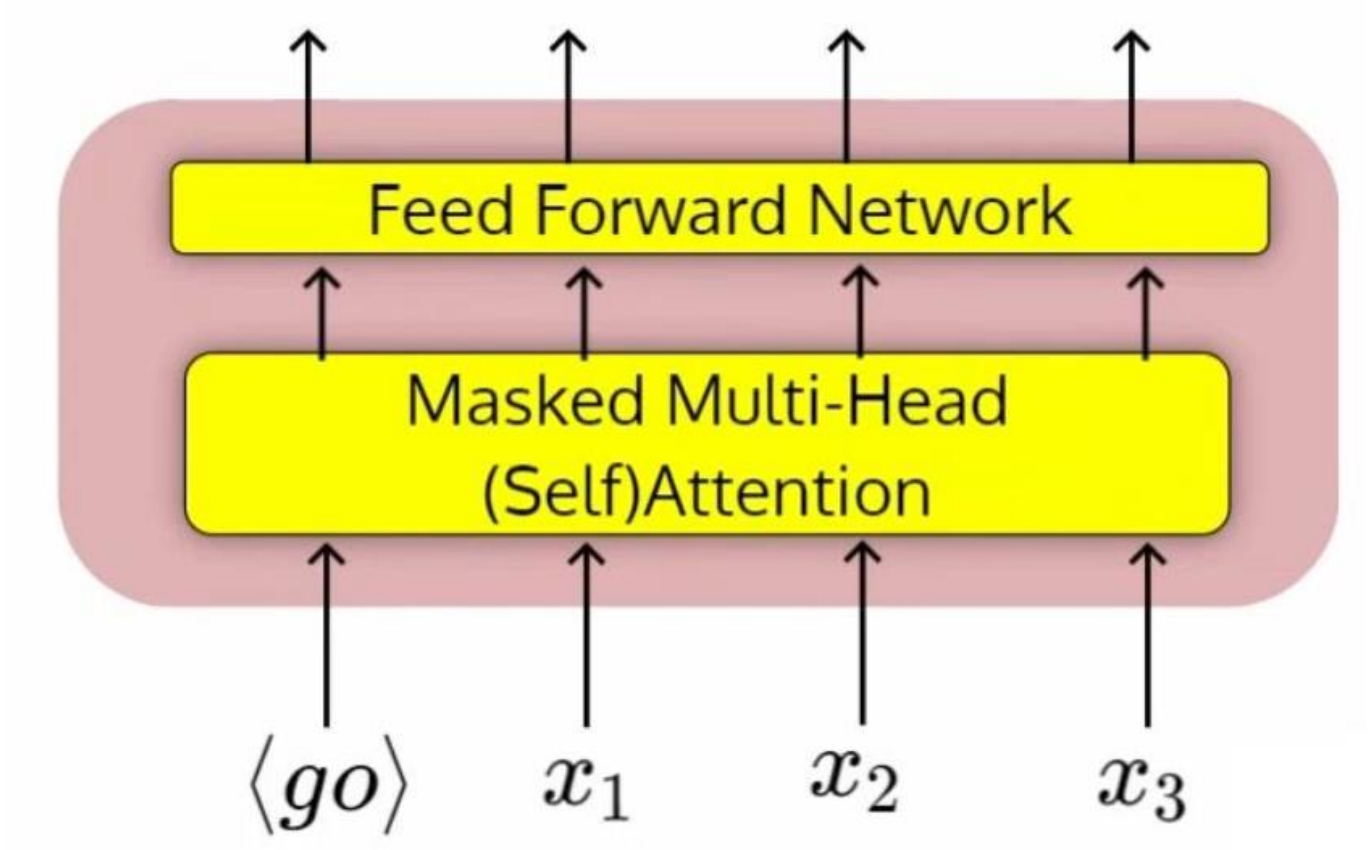
- The Masked Head Attention is Required
- Masking prevents information leakage; essential for auto-regressive behavior.
- Without it, the model would overfit to bidirectional context, failing at generation.



Ensuring Causality - Masking in Self-Attention

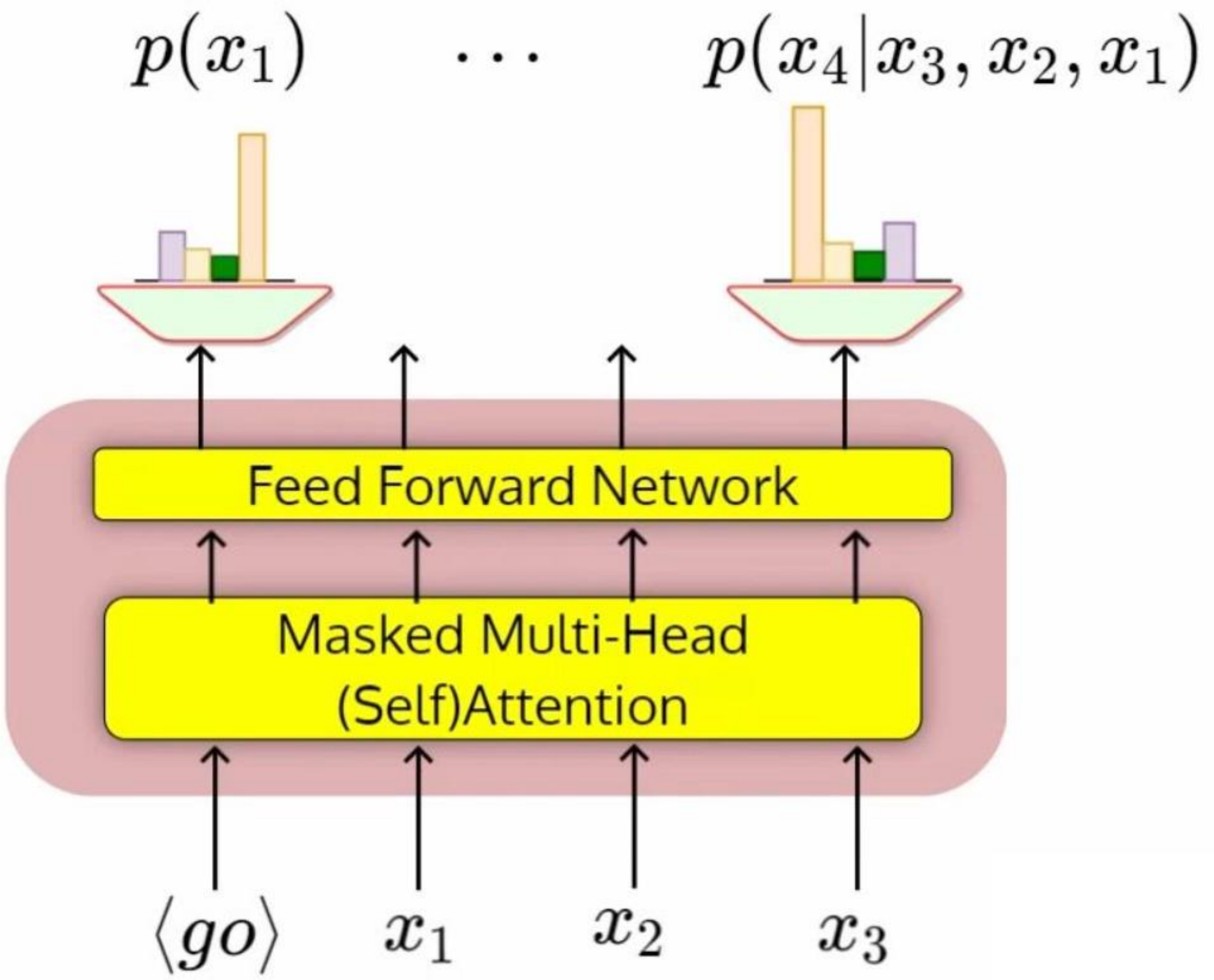
- Does Cross Attention is Required?





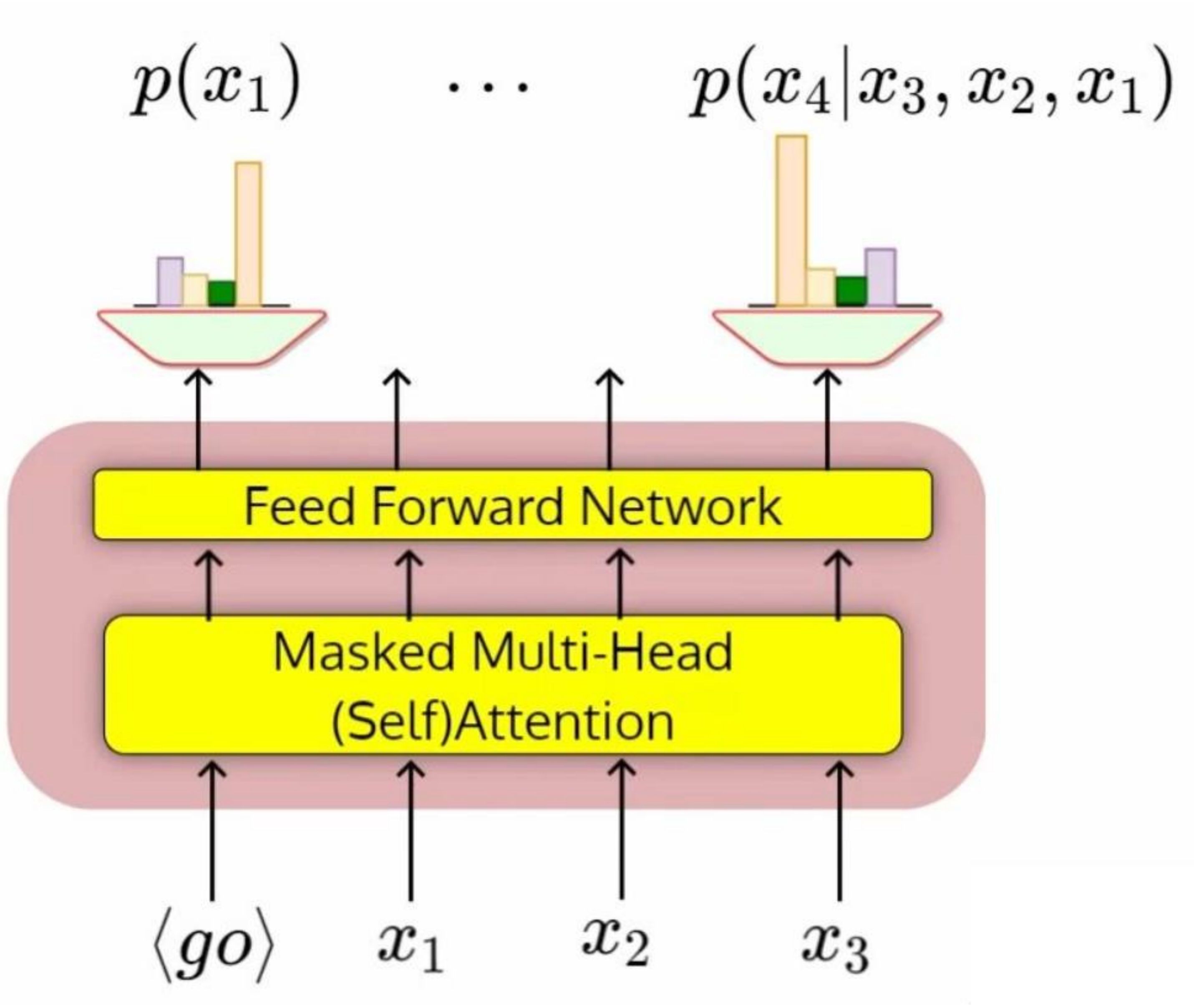
Auto-Regressive Generation Process

- Generate tokens one-by-one, feeding predictions back as input.
- **Start:** Input = special start token (e.g., <BOS>); predict $P(x_1)$.
- **Step 1:** Sample x_1 new input = [<BOS>, x_1 ;[predict $P(x_2 \mid x_1)$.
- **Continue:** For step k, input = [<BOS>, $x_1, \ldots, x_{k-1}$;[predict $P(x_k \mid x_1 \ldots x_{k-1})$.
- **During Training**: Use teacher-forcing – provide ground-truth prefixes, predict next token.



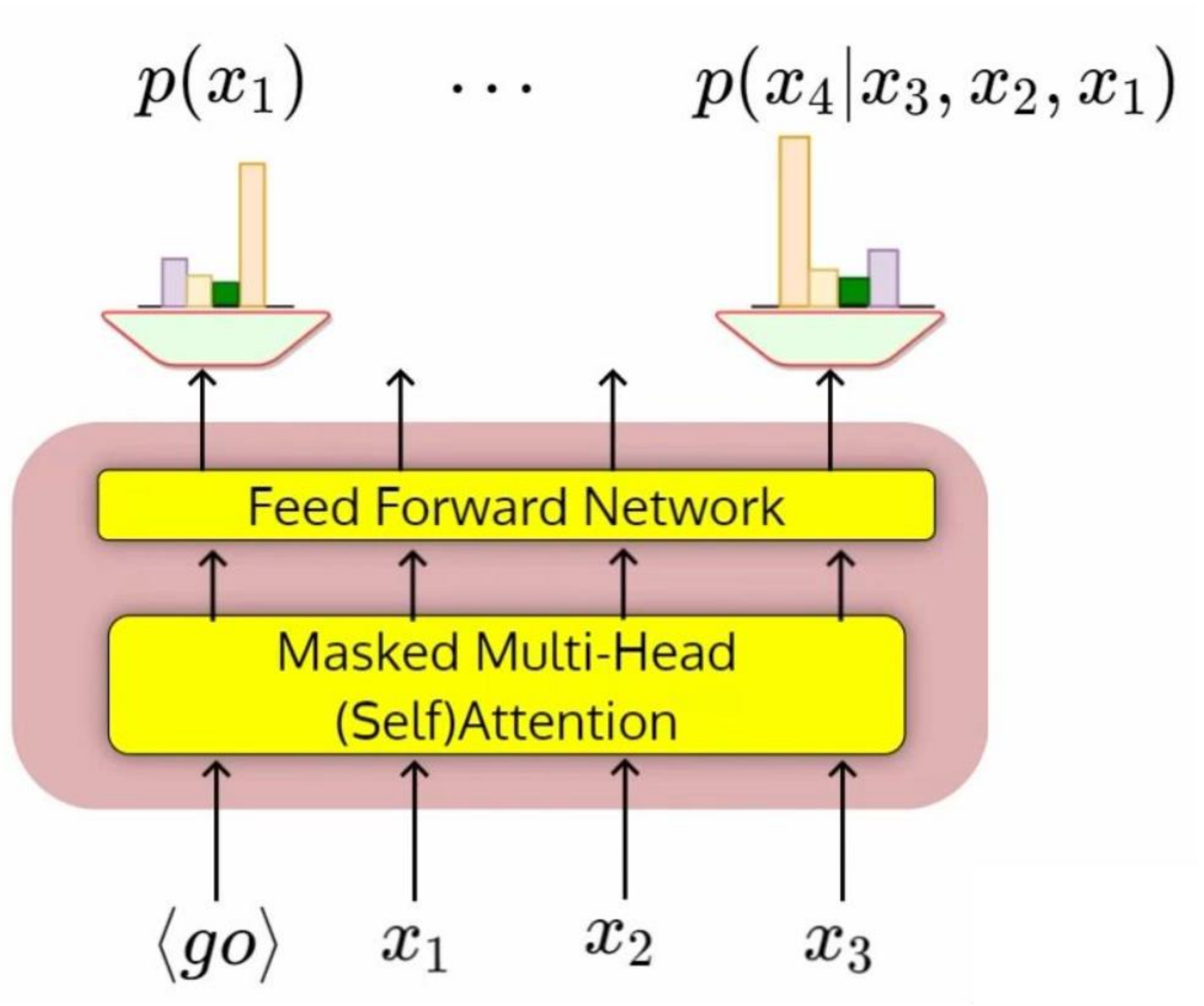
Auto-Regressive Generation Process

- This time the probabilities are determine by the parameter of the model.
- If, all the parameters in transformer like WQ, WK, WT and the parameters are in FFN are called Theta / (θ).
- The input are given like “go, I, am” in the form of Embedding, these are also part of Theta / (θ).
- After that Pass all the Embeddings and pass all into the Transformation and perform computations.
- So, we can say that like Softmax output is computed by the transformer using the parameters of the transformer.



Training Objective - Maximizing Likelihood

- Therefore, the **objective** is to **maximize** the **likelihood** $L(\theta)$
- **Example**: For sequence:
- If my training sample is “I am going home today”.
- When I Pass <go> the P(“I”) should me maximize.
- When I Pass <I> the P(“am”) should me maximize.
- Maximize $P(\text{"am"} \mid \text{"I"})$,then $P(\text{"going"} \mid \text{"I am"})$,etc.



Training Objective - Maximizing Likelihood

- **Likelihood Function**: $\mathcal{L}(\theta) = \prod_{i=1}^{n} P\,(x_i, |, x_1, \ldots x_{i-1}\theta)$.
  - Log-likelihood: $\log \mathcal{L}\,(\theta) = \sum_{i=1}^{n} \log P\,(x_i, |, x_1, \ldots x_{i-1}\theta))$ for stability).
- **Optimization**: Maximize via gradient descent (backpropagation).
  - Loss: Negative log-likelihood
  - Adjust θ so correct next token gets high probability.
- **Iterative Process**: Over epochs, predictions align with data; uses optimizers.
- Trained on massive datasets (e.g., web text).

Causal Language Modeling (CLM)

- **Meaning:** The **task/objective**: Predict the next token given only previous tokens.
- Formal goal: Maximize $P(x_{t+1}, \|, x_1 \ldots x_t)$
- **When is it used?:** Training and inference (the task itself)
- **Key Point to Remember:** This is the task you are solving (like “classification” or “translation”).

Auto-Regressive (AR)

- **Meaning:** The modeling approach / architecture that solves CLM.
- Uses causal (masked) attention so future tokens are invisible.
- Generates one token at a time, feeding its own output back.
- **When is it used?:** Both training and inference (how the model is built)
- Very close to CLM, but technically the method, not the task
- **Key Point to Remember:** Almost everyone uses “auto-regressive” and “causal LM” interchangeably today. 99% of the time they mean the same thing in practice.

Teacher Forcing

- **Meaning:** A training trick/technique used while training auto-regressive models.
- During training, instead of feeding the model's own (often wrong) prediction, we feed the correct ground-truth token as the next input.
- Only during training (never at inference)
- **When is it used?:** Both training and inference (how the model is built)
- **Key Point to Remember:** Without teacher forcing, training GPT-style models would be extremely slow and unstable.

Remember

- **Causal Language Modeling** = the task
- **Auto-regressive** = the architecture that does this task (causal attention + left-to-right generation)
- **Teacher forcing** = the trick we use only during training so the auto-regressive model learns fast and stably
- **Auto-Regressive:** The **model design:** predicts next token using only past tokens (causal) → **(architecture)**
- **True Auto-Regressive: During inference**: model feeds its own predictions back as input. Only at test/generation time.

GPT

- GPT is a Transformer-based auto-regressive model trained with teacher forcing to solve the causal language modeling task.
- **GPT is not a new idea:**
- It's just CLM (task) + auto-regressive Transformer (design) + teacher forcing (training).

Layer	What It Is	Real Name
1. Task	Predict next word	**Causal Language Modeling (CLM)**
2. Architecture	Left-to-right, decoder-only, causal attention	**Auto-Regressive (AR)**
3. Training Trick	Feed correct words during training	**Teacher Forcing**

Model	Year	Size	Still GPT?
GPT-1	2018	117M	Yes
GPT-2	2019	1.5B	Yes
GPT-3	2020	175B	Yes
GPT-4 / GPT-4o	2023–2025	~1.8T	Yes (same recipe, just bigger + better data)

Did OpenAI copy or steal Google's Transformer idea?

- **NO:** OpenAI did NOT copy or steal Google's Transformer.
- They built on a public research paper that Google published openly.
- Imagine Google invents the **wheel** and publishes a paper: "Here's how to make a round wheel use it freely!"
- OpenAI says: "Cool! We'll use **one wheel** and make a **unicycle** (GPT)."
- Not stealing. **Improving and specializing.**
- **OpenAI did not steal they stood on Google's shoulders, just like Google stood on LSTM's shoulders."** This is **progress**, not theft.
- **Google gave the world the Transformer. OpenAI made it talk.** Both win. Science wins. You win (you get ChatGPT!).
- No drama. Just good research.

Fact	What Actually Happened
June 2017	**Google publishes** the paper **"Attention Is All You Need"** Introduces **Transformer**
Paper is 100% public	Anyone (OpenAI, Meta, you, me) can read, use, and build on it
2018–2019	**OpenAI releases GPT-1, GPT-2** Uses **decoder-only Transformer** from the paper
Legal?	YES: research papers are meant to be shared and improved
Stealing?	NO: this is how science works