Week 2

LLM Overview

Understanding the Architecture and Training of Large Language Models

Agenda

- Transformer Architecture and Attention Mechanism
- Next Token Prediction and Hallucination
- LLM Pretraining
- Supervised Fine-Tuning (SFT)
- Alignment Techniques: DPO and PPO
- Data Requirements, Costs, and Challenges
- Test-Time Scaling (O1, O3)
- Hands-On Project Introduction

Transformer Architecture

Multi-Head Self-Attention

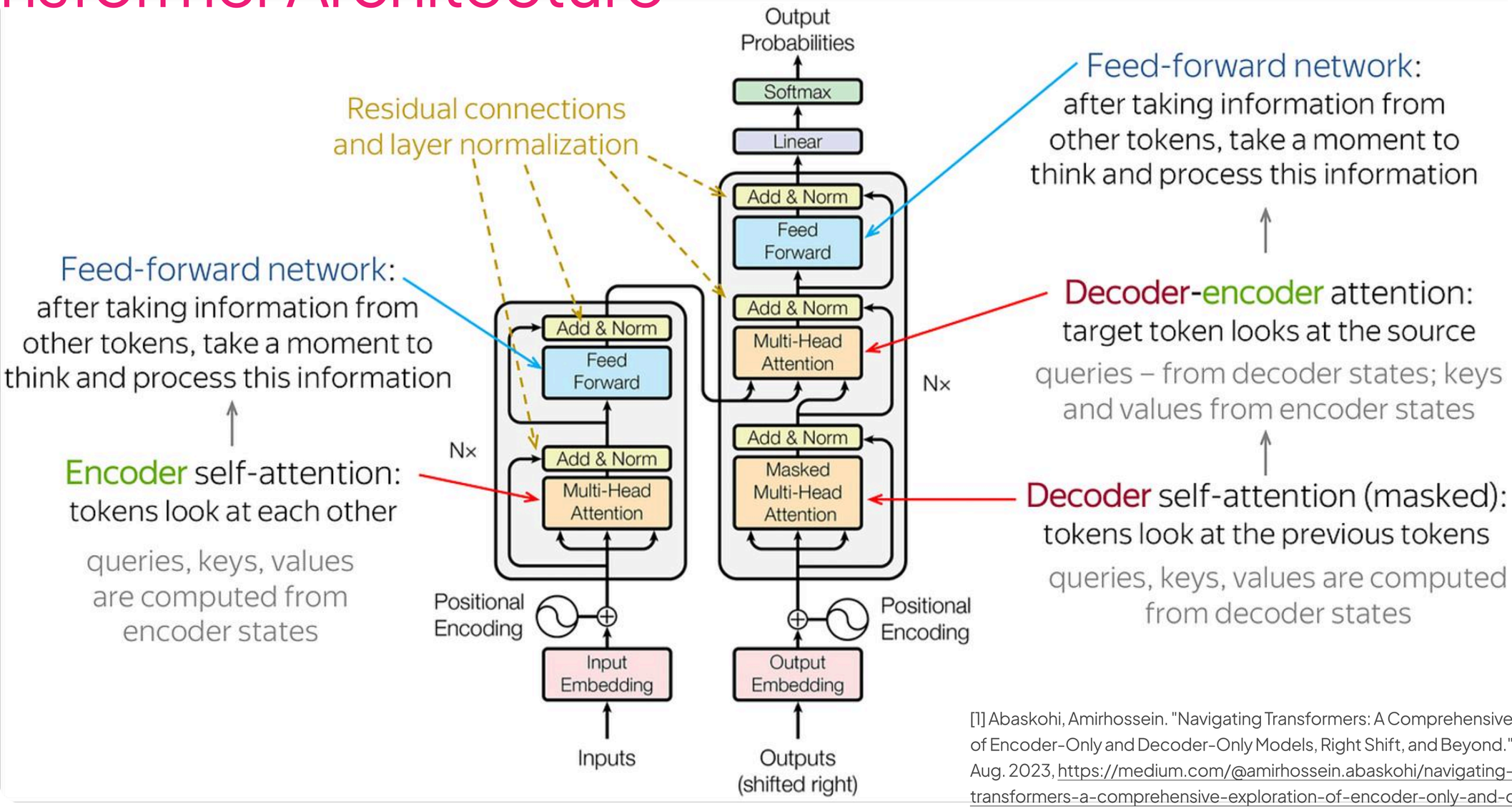
Feedforward Neural Networks

Key Components

Positional Encoding

Layer Normalization

Transformer Architecture



[1]

[1] Abaskohi, Amirhossein. "Navigating Transformers: A Comprehensive Exploration of Encoder-Only and Decoder-Only Models, Right Shift, and Beyond." Medium, 16 Aug. 2023, https://medium.com/@amirhossein.abaskohi/navigating-transformers-a-comprehensive-exploration-of-encoder-only-and-decoder-only-models-right-a0b46bdf6abe.

Encoder-Decoder Structure

Overview:

- The Transformer architecture revolutionized natural language processing by eliminating recurrence and convolutions in favor of attention mechanisms. [1]
- The architecture comprises two main components:
- The **Encoder** and the **Decoder**, both built from stacks of identical layers.

Input Processing:

- Each input token is first converted into a vector using an embedding layer.
- Positional encoding is added to these embeddings to retain the order of the sequence, as the model lacks inherent sequence-awareness.

Layer Composition:

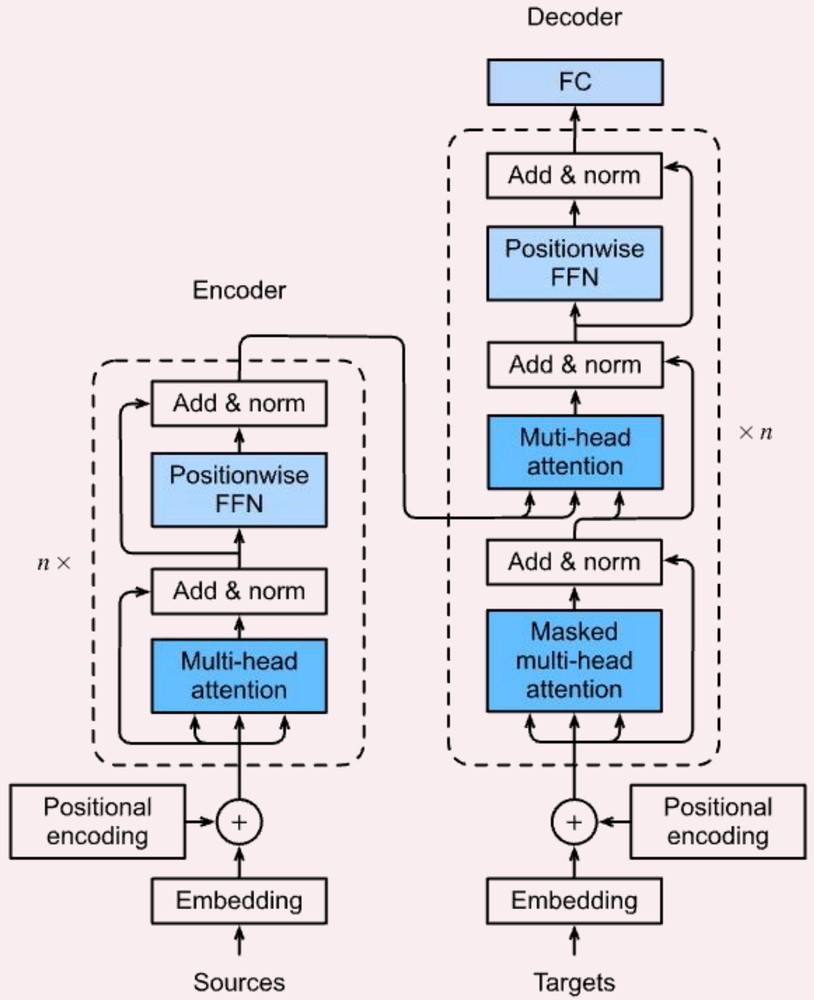
Each encoder layer consists of:

1. **Multi-Head Self-Attention Mechanism**: Allows the model to focus on different parts of the input sequence simultaneously, capturing various relationships.
2. **Feed-Forward Neural Network (FFN):** Applies non-linear transformations to each position separately and identically.

- Residual connections and layer normalization are employed after each sub-layer to facilitate training and convergence

[1] Zhang, Aston, et al. "11.7. The Transformer Architecture." Dive into Deep Learning, 2023, https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html.

Encoder



[1]

Input Processing:

- Similar to the encoder, the decoder starts with embedding the target sequence tokens and adding positional encodings.

Layer Composition:

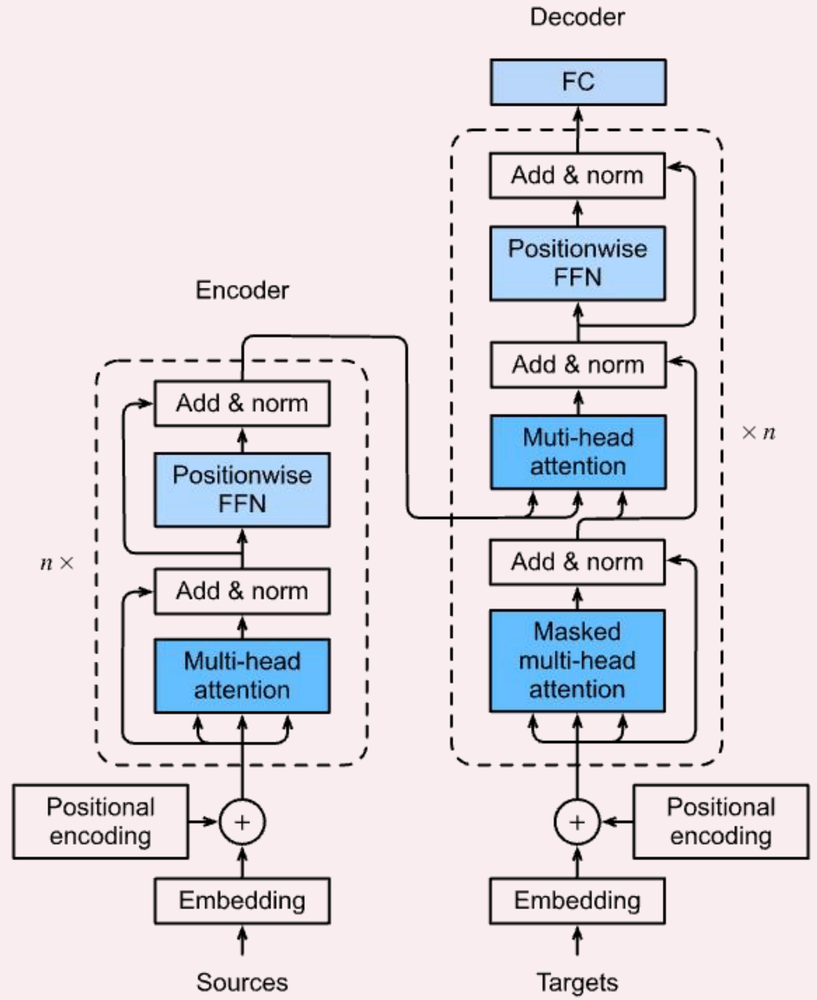
Each decoder layer consists of:

1. **Masked Multi-Head Self-Attention Mechanism:** Prevents positions from attending to subsequent positions, ensuring the model predicts the next token without peeking ahead.
2. **Encoder-Decoder Attention Mechanism:** Allows the decoder to focus on relevant parts of the input sequence by attending to the encoder's output.
3. **Feed-Forward Neural Network (FFN):** Similar to the encoder's FFN, applied position-wise.

Again, residual connections and layer normalization are applied after each sub-layer.

[1] Zhang, Aston, et al. "11.7. The Transformer Architecture." Dive into Deep Learning, 2023, https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html.

Decoder



[1]

Encoder-Decoder Key Features:

- **Parallelization:** Unlike RNNs, Transformers process all tokens simultaneously, enabling efficient training and inference. [1]
- **Long-Range Dependency Modeling:** The attention mechanisms allow the model to capture relationships between distant tokens effectively.
- **Flexibility:** The architecture can be adapted for various tasks by modifying the encoder, decoder, or both.

[1] Hugging Face. Transformer-based Encoder-Decoder Models. Hugging Face Blog

Applications

MACHINE TRANSLATION
(E.G., ENGLISH TO GERMAN)

TEXT SUMMARIZATION

QUESTION ANSWERING

TEXT GENERATION

Attention Mechanism

- An attention mechanism is a component in deep learning models that enables the network to focus on specific parts of the input data, assigning different levels of importance to different elements. This approach enhances the model's ability to capture relevant information, especially in tasks involving sequences or spatial data. Here’s some attention mechanism

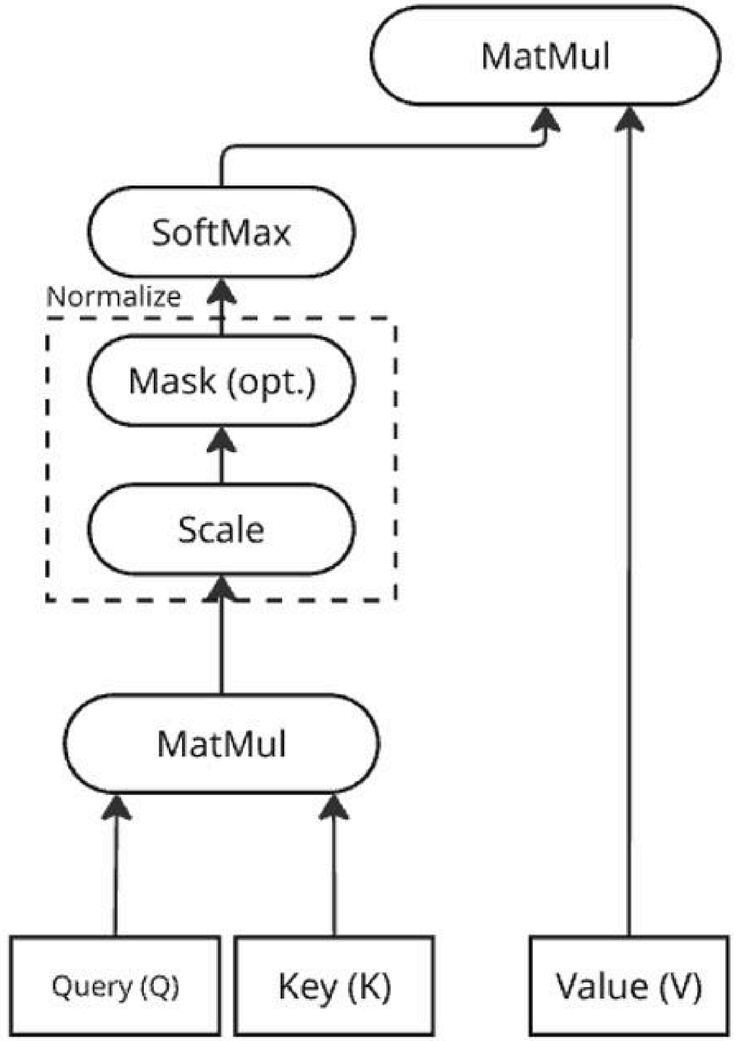
- Type of the attention cover in this lecture

1. Self-Attention
2. Multi-Head Attention

Self Attention / Scaled Dot Product attention

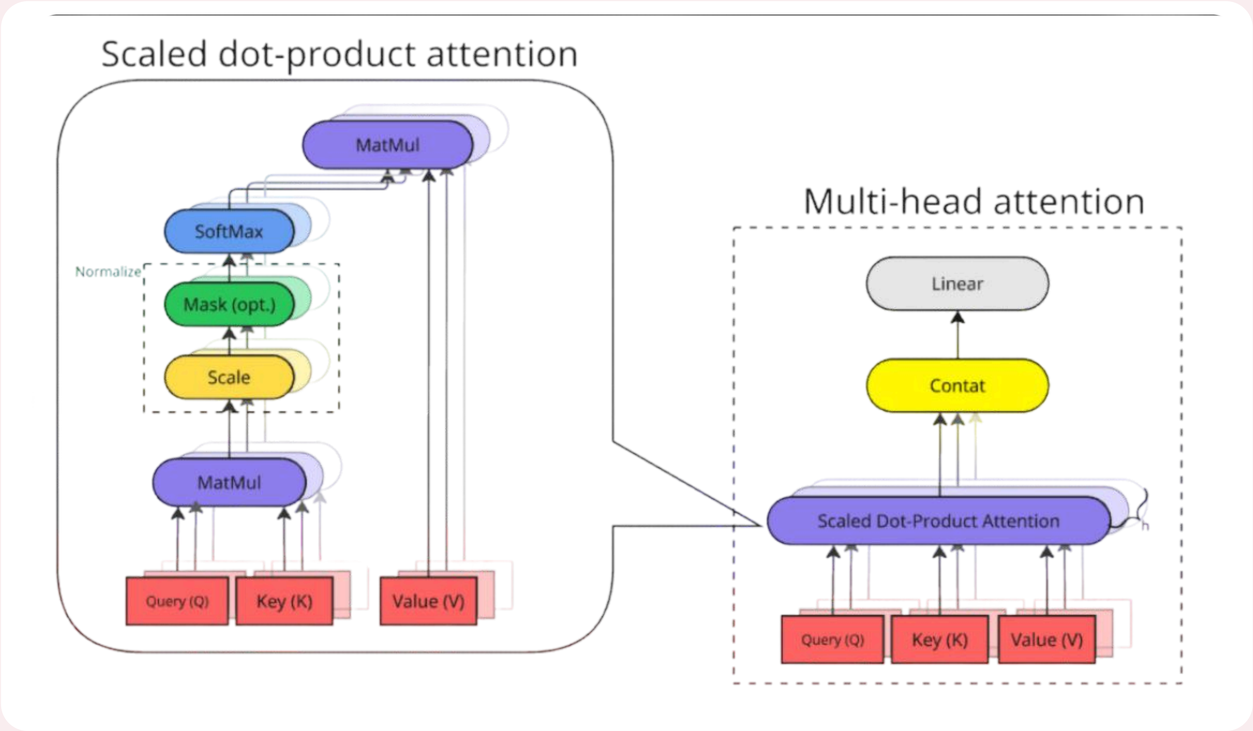
- The scaled dot-product attention is a mechanism that Transformers use to weigh the significance of different parts of the input data. It computes attention scores based on the **query (Q), key (K), and value (V)** matrices derived from the input. The attention function is computed as a dot product of Q and K, scaled by the square root of the key's dimensionality, followed by a softmax function to obtain the weights on the values.

Application: Widely used in Natural Language Processing (NLP) tasks to understand context by relating words in a sentence to each other.



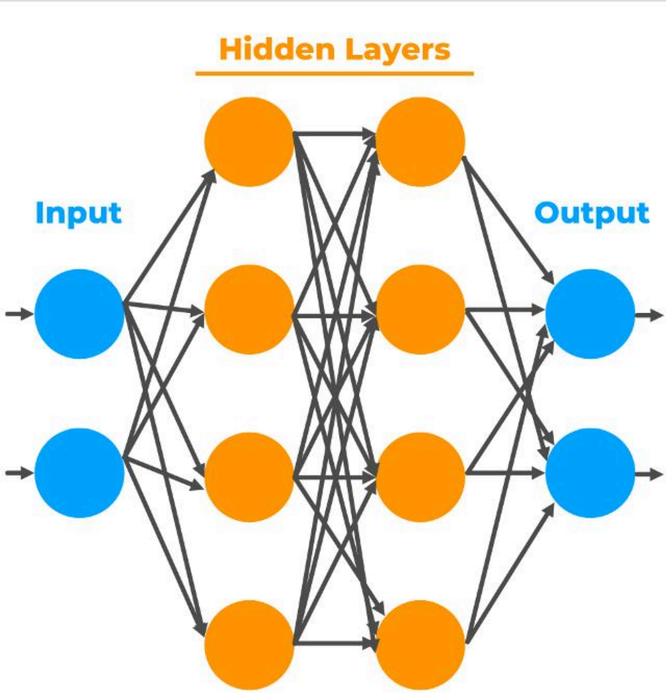
Multi-Head Attention

The **Multi-Head Attention** mechanism allows the model to focus on different parts of the input sequence simultaneously. By employing multiple attention "heads," the model can capture various types of relationships and dependencies within the data.



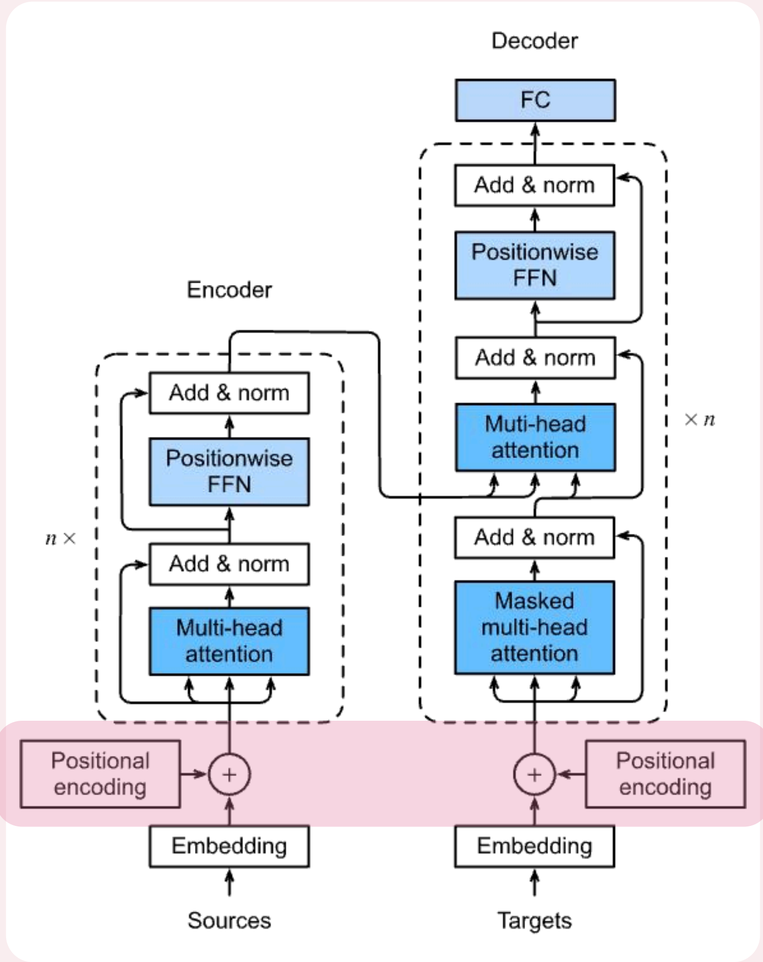
- Multiple “heads” are created, each attending to the input sequence through its own Q, K, and V matrices.
- Each head learns to focus on different aspects of the data, like long-range dependencies, syntactic relationships, or local word interactions.
- The outputs from each head are then concatenated and projected to a final representation, capturing the multifaceted nature of the input.

Feedforward Neural Networks



- A feedforward neural network is a type of neural network where connections between the nodes do not form a cycle. It processes inputs in one direction—forward—from input to output.
  - **No recursion or feedback loops**
  - **Core component of the Transformer architecture**
  - **Used after attention layers to apply transformations to each token independently**
- Common Types of FFNNs
  - Standard Dense FFNN
  - Position-wise FFN (used in Transformers)
  - Residual FFN Blocks

Positional Encoding in Transformers



Why Positional Encoding?

- Transformers lack recurrence or convolution, so they need an explicit way to understand the **order** of input tokens
- Helps model capture **sequence order** and **token relationships**
- Added to input embeddings before feeding into the encoder

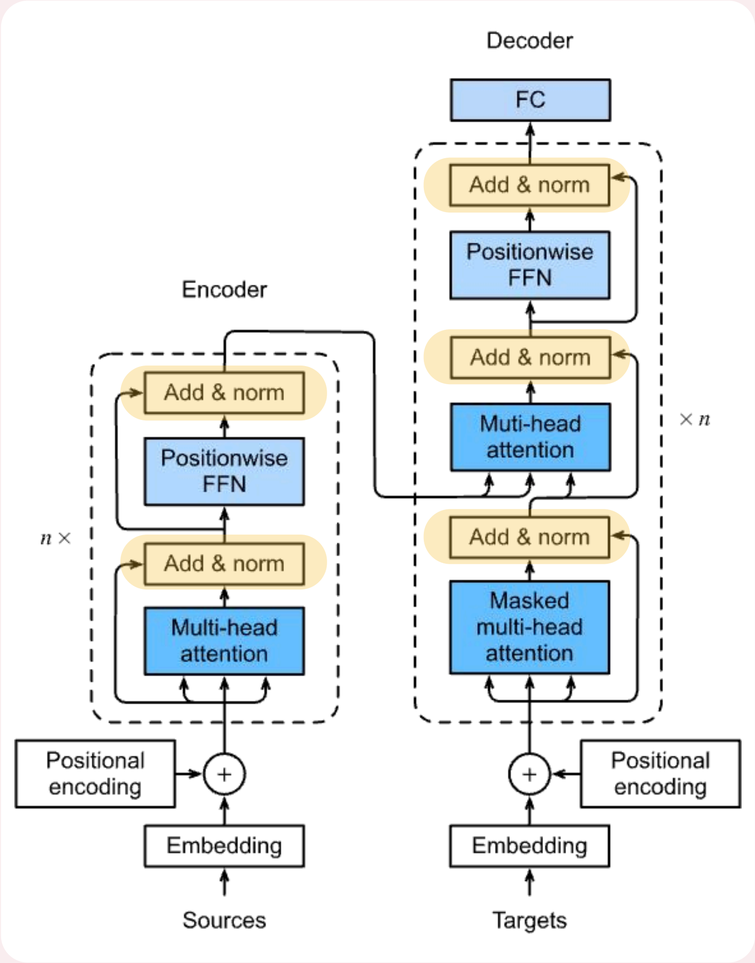
Two Main Types

1. Sinusoidal Positional Encoding (used in original Transformer)
2. Learned Positional Encoding

Positional encodings are added to token embeddings so the model can differentiate between:

- “The cat sat on the mat”
- “On the mat sat the cat”

Layer Normalization in Transformers



What is Layer Normalization?

Layer Normalization (LayerNorm) stabilizes and accelerates training by normalizing the **hidden states** across the **features of a single token**.

- Unlike BatchNorm (which normalizes across the batch), LayerNorm operates **independently per token**.
- Especially important in Transformers where token positions are processed in parallel.

Where It's Used in Transformers

- **Before or after** Multi-Head Attention and Feedforward layers (depending on pre-norm or post-norm architecture)
- Combined with **residual connections** to ensure gradient flow and model stability

Benefits

- Faster convergence during training
- Reduces internal covariate shift
- Enables deeper models with residual learning

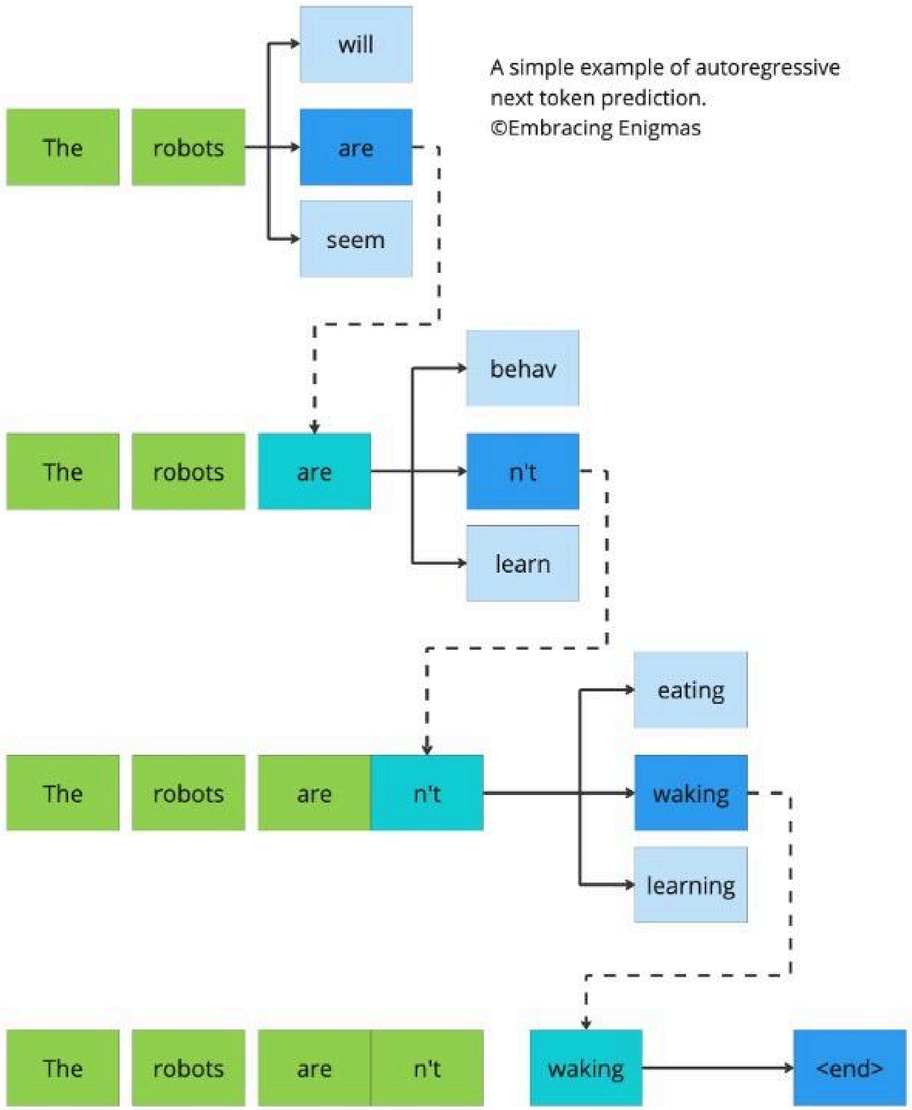
Next Token Prediction

Predict the next word in a sequence given the previous words.

Example:

- Input: ”The robots aren’t"
- Model Prediction: ”walking”

Training: Uses large corpora of text data to learn language patterns.



Hallucination in LLMs

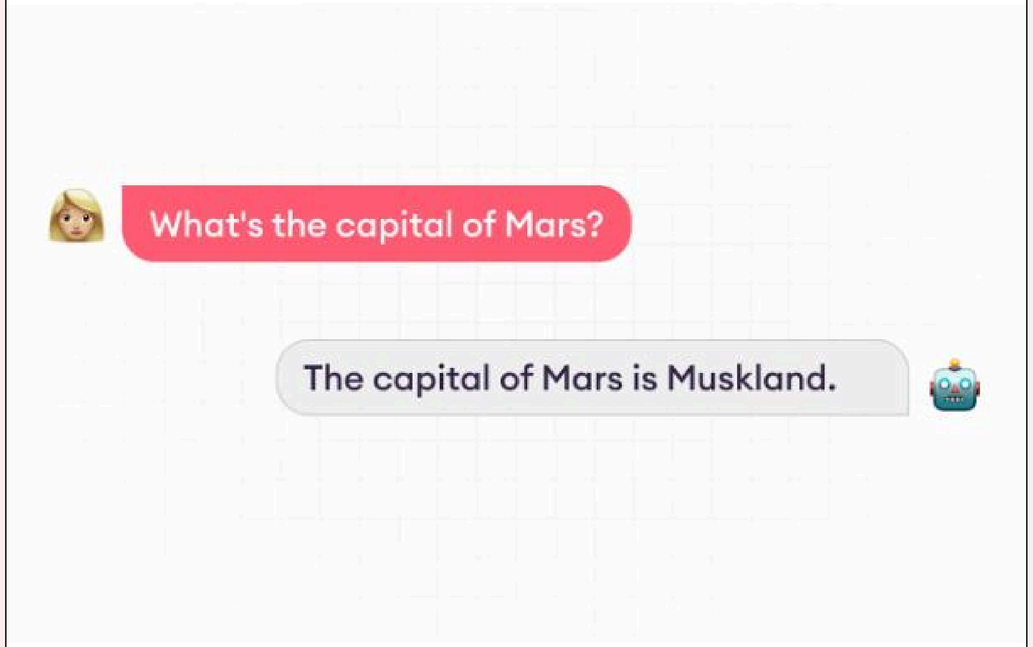
Definition: Hallucination occurs when an LLM generates output that is **plausible but factually incorrect or nonsensical**.

Why It Happens 🧠

- Lack of grounding in external factual data
- Over-generalization from training data
- Prompt ambiguity or under-specification
- Generation objectives not aligned with factuality

Example:

- Inventing non-existent facts
- Misattributing quotes or events



Reducing Hallucination in LLMs

1. **Better Prompt Design**
   - Be explicit in your instructions
   - Use few-shot or chain of thought (CoT) prompting to guide reasoning
2. **Use Retrieval-Augmented Generation (RAG)**
   - Augment LLMs with **external factual sources** (e.g., vector DBs)
   - Example: Use LangChain to retrieve context before generating
3. **Post-Processing Validation**
   - Use verifier models or agents to **fact-check outputs**
   - Cross-reference with external knowledge bases or search APIs
4. **Instruction-Tuned Models**
   - Use models like **GPT-4-turbo**, **Claude 3**, or **Mistral-Instruct**
   - These are better aligned to follow grounded instructions
5. **Guardrails and Tool Use**
   - Route complex or fact-sensitive queries through tools (search, calculators, APIs)
   - Integrate agents like **MCP** or **AutoGPT** to validate results dynamically

LLM Pretraining

What is LLM Pretraining?

🧠Objective

- Train LLMs to predict the next token in a sequence
- Learn general language understanding through unsupervised learning

🔧Core Characteristic:

- No labeled data required
- Massive scale: often 100B+ tokens
- Foundation for downstream finetuning

Example Task:

Input: "The Eiffel Tower is located in"

Target: "Paris"

Pretraining Pipeline Overview

General steps

1. Data Collection	2. Data Cleaning	3. Tokenization	4. Training Objective
Crawl and aggregate massive public text corpora	Remove low-quality, duplicated, or non-linguistic content	Use Byte Pair Encoding (BPE)or SentencePiece to tokenize text	Next Token Prediction/ Masked Language Modeling

Common Pretraining Datasets

Dataset	Description	Source
Common Crawl	Web-scale crawl data (~10TB+)	commoncrawl.org
BooksCorpus	11K+ books with diverse topics	GitHub: yknzhu/BookCorpus
Wikipedia	High-quality encyclopedic content	dumps.wikimedia.org
GitHub Code	Code repositories across many languages	GitHub Archive
ArXiv / PubMed	Scientific papers for domain LLMs	HuggingFace Datasets

Tokenization and Training Objectives

Tokenization

- Converts text to IDs using learned vocabulary
- BPE & SentencePiece commonly used for LLMs

Objective Examples:

- **Causal LM (e.g., GPT-3):** Predict the next token
- **Masked LM (e.g., BERT):** Predict randomly masked tokens

Example (Causal LM):

- Input: "The weather today is"
- Target: "sunny"

Key Challenges in Pretraining

Data Quality Issues

- Noisy web data → hallucinations or bias
- PII contamination → privacy risks
- Redundancy → overfitting
- Dataset Balance

Dataset Balance:

- Overrepresentation of English or certain domains
- Insufficient coverage of minority languages

Resource Constraints – Training GPT-3 (175B):

- 3000+ A100 GPUs
- Months of compute
- $5M–$10M cost

Pretraining Best Practices (Case Study) – LLaMA 4

Overview:

- **Models:** LLaMA 4 Scout and Maverick
- **Tokens:** ~40T (Scout), ~22T (Maverick)
- **Compute:** 7.38M GPU-hours on H100–80GB
- **Architecture:** Mixture-of-Experts (MoE) with early fusion for multimodality

Data & Filtering:

- **Sources:** Publicly available, licensed data, and Meta's own platforms
- **Techniques:** Deduplication with MinHash, language classification, and removal of low-quality content

Training Enhancements:

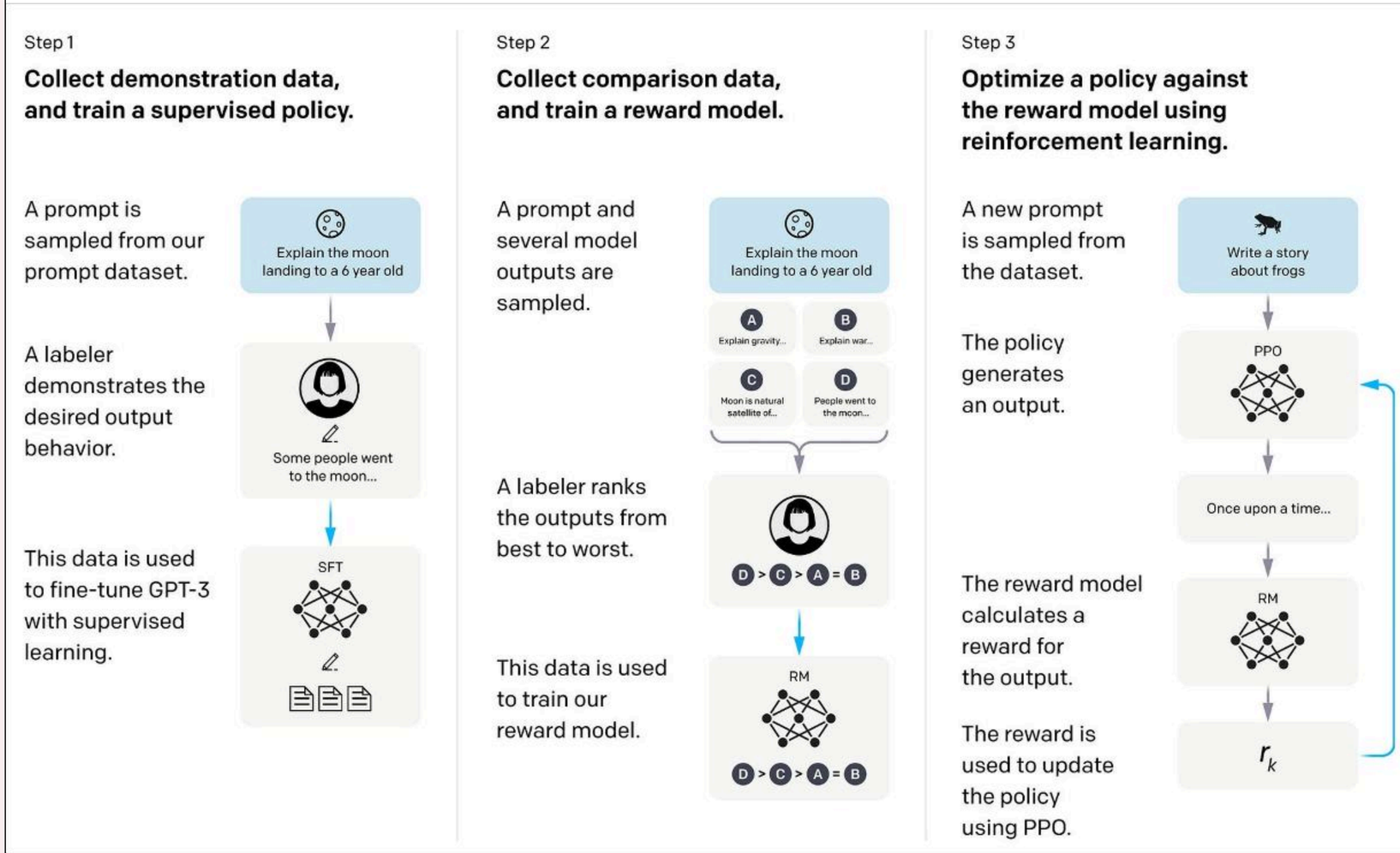
- **Precision:** FP8 for efficient computation
- **Context Window:** Up to 10 million tokens (Scout)
- **Post-training:** Lightweight SFT, online RL, and DPO for improved alignment

Multilingual & Multimodal:

- **Languages:** Pretrained on 200 languages
- **Modalities:** Supports text and image inputs

Meta AI. "Llama 4 Omni: Model Cards and Prompt Formats." Llama, 5 Apr. 2025, https://www.llama.com/docs/model-cards-and-prompt-formats/llama4_omni/.

Supervised Fine-Tuning (SFT)



Adapt the pretrained model to specific tasks using labeled data.

Methods:

- Full fine-tuning
- Parameter-efficient tuning (e.g., LoRA)

Applications:

- Question Answering
- Text Classification
- Summarization

WE WILL LEARN MORE SFT IN LESSON 5

Alignment Techniques

Ensure model outputs align with human values and intentions.

Methods:

DPO (Direct Preference Optimization):

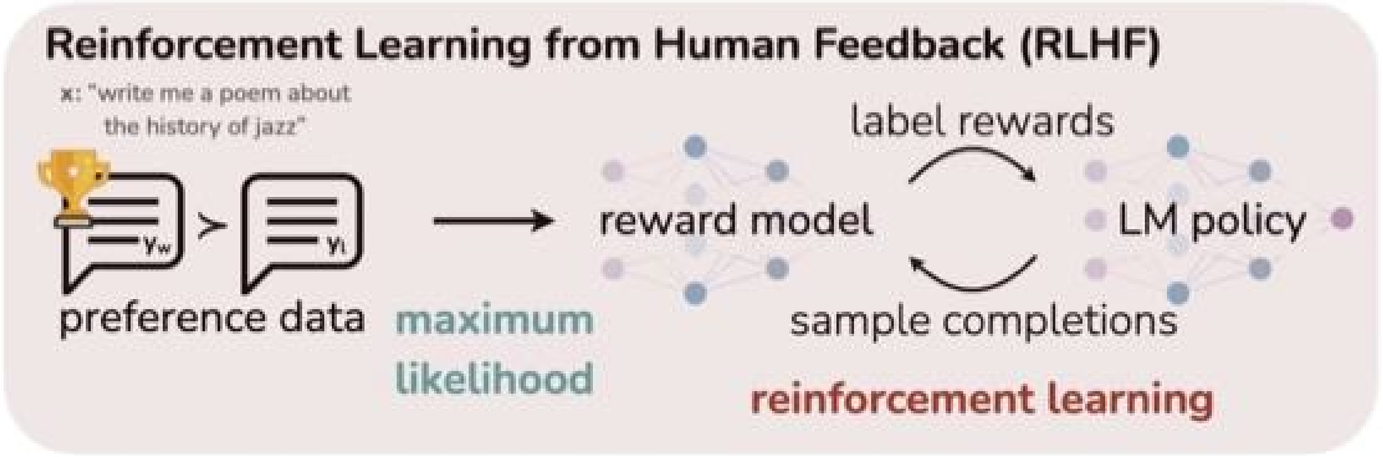
- Optimizes model outputs based on human preferences.

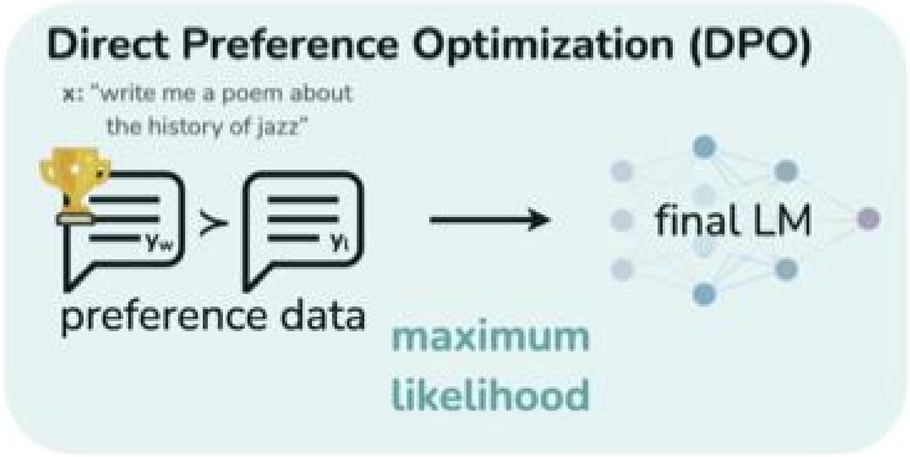
PPO (Proximal Policy Optimization):

- Reinforcement learning approach to fine-tune models.

Importance:

- Reduces harmful or biased outputs
- Improves user satisfaction





Emanuilov, Simeon. "Direct Preference Optimization (DPO) in Language Model Alignment." UnfoldAI, 23 Dec. 2024, https://unfoldai.com/dpo-llms/.

Direct Preference Optimization (DPO)

What is DPO?	A supervised learning method that aligns LLMs directly with human preferences. Simplifies the alignment process by eliminating the need for a separate reward model or complex reinforcement learning loops.
How It Works:	Utilizes datasets containing pairs of responses labeled as "preferred" and "less preferred." Optimizes the model to increase the likelihood of generating preferred responses over less preferred ones.
Benefits:	Simpler and more computationally efficient compared to traditional RLHF methods.Demonstrated effectiveness in tasks like summarization and dialogue generation.
Use Cases:	Training models like Zephyr and Intel's NeuralChat. Aligning LLM outputs with specific human preferences in various applications.

Proximal Policy Optimization (PPO)

What is PPO?	A reinforcement learning algorithm used to fine-tune LLMs based on human feedback. Forms the backbone of the Reinforcement Learning from Human Feedback (RLHF) approach.
How It Works:	Trains a reward model using human-labeled data indicating preferred responses. Fine-tunes the LLM by optimizing its policy to generate responses that maximize the reward signal, while ensuring stability through clipped updates.
Benefits:	Provides robust alignment with human preferences across diverse tasks. Balances exploration and exploitation, leading to stable and efficient training.
Use Cases:	Fine-tuning models like OpenAI's ChatGPT. Enhancing LLM performance in complex, multi-turn dialogue systems

Training Data

Data Requirements – Scale and Diversity

Why Data Matters

- LLMs learn by generalizing from massive datasets.
- **More data → better generalization** (to a point), especially for multilingual and multi-domain tasks.

Example:

- **GPT-4**: trained on an estimated **45+TB of text**, including books, code, social media, academic data.
- **LLaMA 4**: used **22T–40T tokens** with high diversity across 200+ languages.

Key Requirements:

- **Scale**: at least trillions of tokens to compete at frontier model level.
- **Diversity**: includes web pages, code, academic papers, user queries, social chat, etc.
- **Quality**: low-noise, well-formatted, human-like data improves alignment and reduces hallucination.

Training Costs – Real-World Examples

- Infrastructure Costs
- **Hardware**: Requires **A100/H100 GPU clusters** (or TPU equivalents)
- **Time**: Full pretraining can take **weeks to months**

Model	Params	Tokens	GPU Hours	Cost Estimate
LLaMA 2 70B	1.4T	2.6T	~1.5M	~$3M–$4M
GPT-3	175B	~300B	~3M	~$10M+
LLaMA 4	MoE 400B	22T–40T	7.4M+	~$15M+

Data Collection Challenges

Challenges in Gathering Training Data

Low-Quality Web Content	Duplicate Content	Imbalanced Domains	PII and Ethical Risks
Spam,clickbait,auto-generated junk Example: Common Crawl requires heavy filtering	Repeated tokens bloat training without adding signal Solution: **MinHash or Exact Deduplication**	Overrepresentation of English and tech leads to poor performance on underrepresented groups Requires curation of minority languages, informal speech, and diverse domains	Email addresses, SSNs, and user data often included in scraped corpora Must apply **regex filters + classifiers** to remove sensitive information

Practical Considerations

What This Means for You

- **Smaller Teams →** focus on data curation + fine-tuning, not full pretraining.
- Use public or open datasets (e.g., The Pile, Wikipedia dumps, arXiv, Common Crawl + filters)
- **Lightweight options:**
  - Start with open LLMs (Ollama, Mistral, LLaMA, Phi)
  - Train domain adapters (LoRA) with **instruction or SFT data**

Test-Time Scaling (O1, O3)

Optimizing model performance during inference by adjusting computational strategies without altering the model's weights.

01 Optimization:

- Utilizes mixed-precision training (e.g., FP16) to reduce memory usage and increase speed.
- Incorporates techniques like kernel fusion and hardware-specific quantization to boost performance.

03 Optimization:

- Employs advanced methods such as sparse attention mechanisms and adaptive computation to handle complex tasks efficiently.
- Integrates reflection mechanisms, allowing models to self-assess and refine their outputs during inference.

Benefits:

- Enhanced reasoning capabilities.
- Reduced latency and computational costs.

Thank you!