Large Language Models

LLM

Large Language Models are a subset of neural networks designed specifically for processing and generating human language which is trained on massive and diverse data.

- Trained on massive data (billions of parameters)
- General purpose (previous ML models are trained task specific)
    - No feature engineering needed.
- Contextual Understanding

Differences between Traditional ML and LLM

- Specialization and Flexibility
- Scale and Complexity
- Training Data and Process
- Performance and Generalization

Transformer Architecture

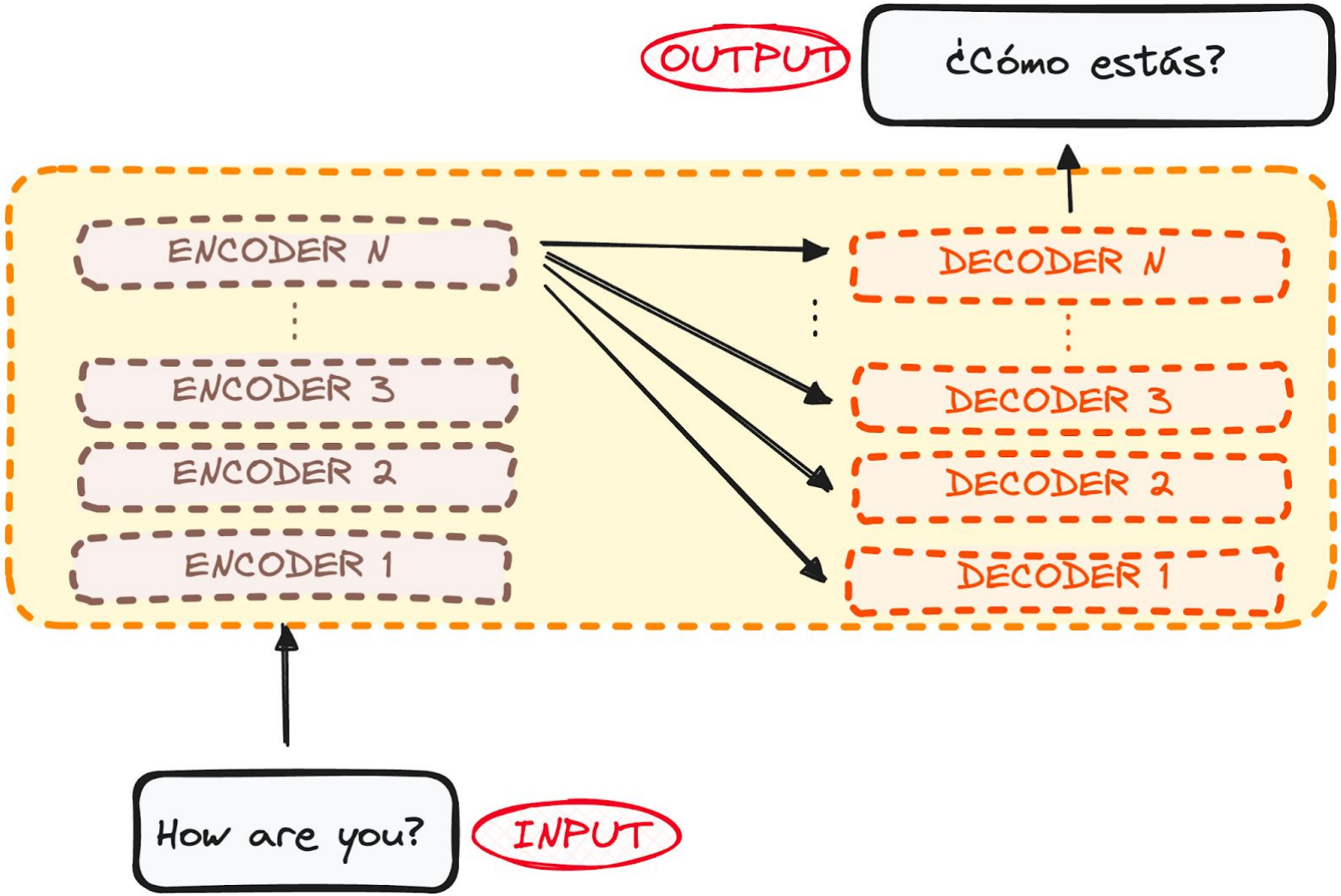
“Attention is all you need” by Google in 2017

A transformer is a type of artificial intelligence model that learns to understand and generate human-like text by analyzing patterns in large amounts of text data.

Recurrent Neural Network to Transformer architecture

- Processes input data sequentially
- inefficient when the elements are distant

- attention without regarding the distance
- boosted the performance(use of GPU)

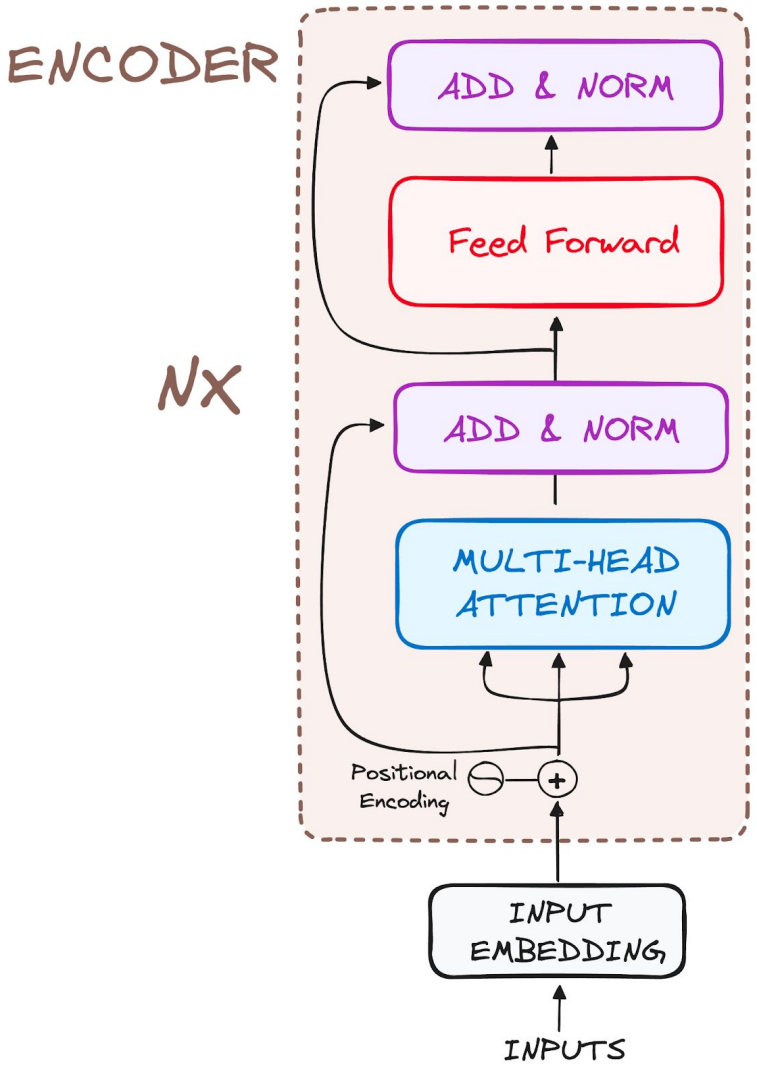

OUTPUT
¿Cómo estás?
ENCODER N
ENCODER 3
ENCODER 2
ENCODER 1
DECODER N
DECODER 3
DECODER 2
DECODER 1
How are you?
INPUT

The Encoder WorkFlow

The primary function of the encoder is to transform the input tokens

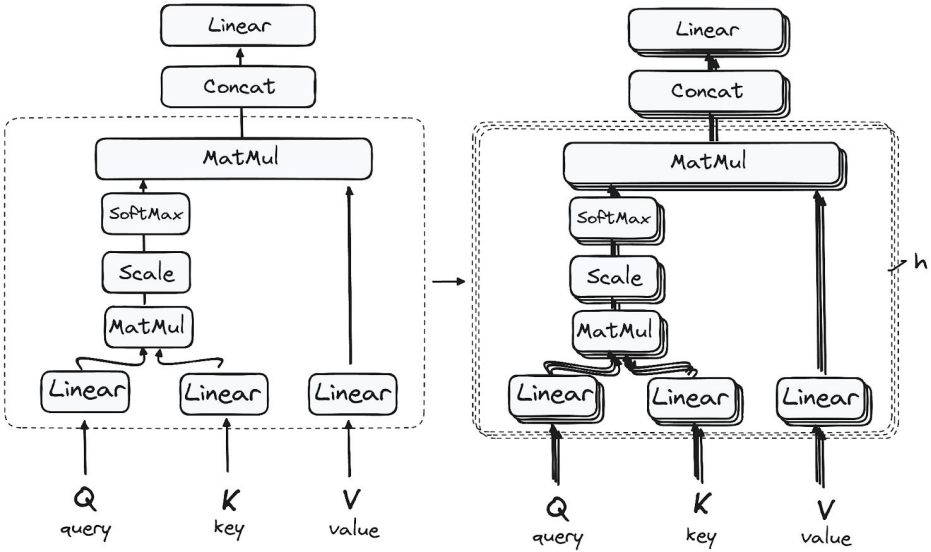
into contextualized representations.

- Input Embeddings
- Positional Encoding
  Stack of Encoder Layers
  - Multi headed self attention
  - **Normalization and Residual Connections**
  - **Feed-Forward Neural Network**



Multi headed self attention mechanism

- A query is a vector that represents a specific word or token from the input sequence in the attention mechanism.
- A key is also a vector in the attention mechanism, corresponding to each word or token in the input sequence.
- Each value is associated with a key and is used to construct the output of the attention layer. When a query and a key match well, which basically means that they have a high attention score, the corresponding value is emphasized in the output.



The Decoder WorkFlow

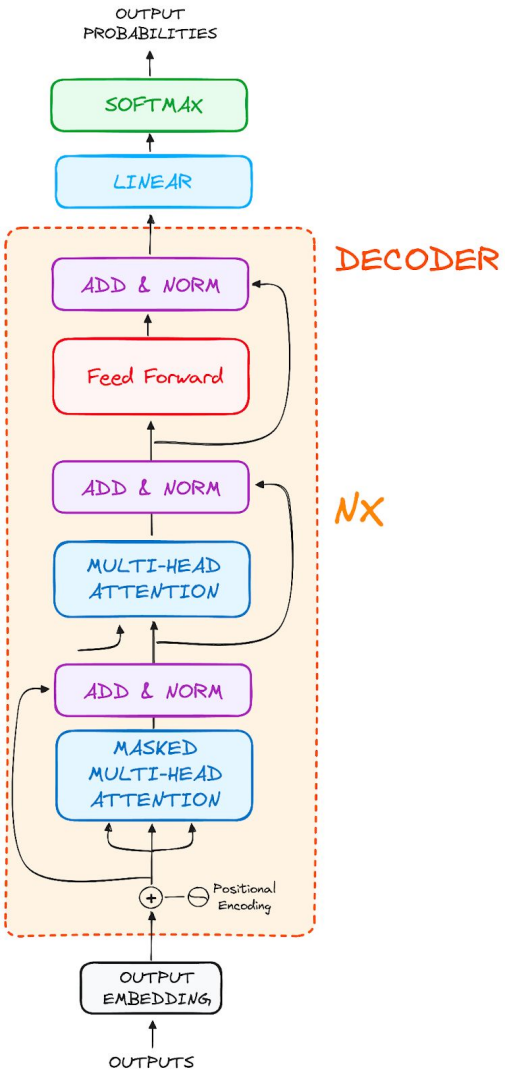
Output Embeddings

Positional Encoding

Stack of Decoder Layers(6 in the original Transformer model).

- **Masked Self-Attention Mechanism**
- **Encoder-Decoder Multi-Head Attention or Cross Attention**
- **Feed-Forward Neural Network**

Linear Classifier and Softmax for Generating Output Probabilities



Pre-training

Pre-training is the process until we have a general purpose model that understands the basics of the language.

Gather the data > Preprocess the data > Design the NN architecture > Training objective > Validate

In **Masked Language Modeling (MLM)**, the model is trained to predict missing or masked words in sentences (used in BERT).

In **Causal Language Modeling (CLM)**, the model is trained to predict the next word in a sequence, based on the words that came before (used in GPT).

Significance of Pre-training

- General Language Understanding
- Enabling Transfer Learning
- Data Efficiency and Lower Training Cost
- Easy Customization for Specific Tasks

Fine-tuning

With fine tuning, we train this existing pretrained model for a specific task with domain specific data.

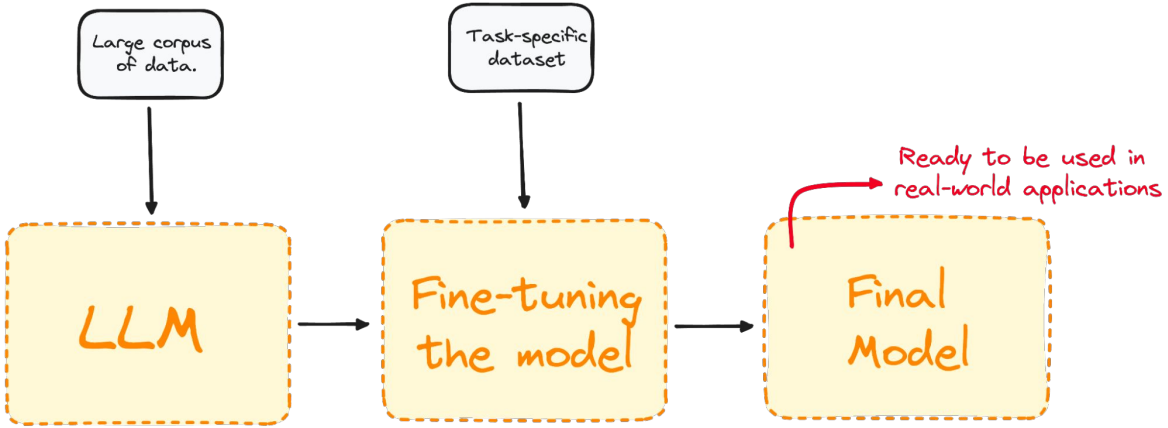
Choose a pre-trained model and a dataset > Load the data > Preprocess the data > Initialize base model > Train > Validate

Significance of Fine Tuning

Specializing General Language Model

Efficiency in Resource Utilization

Limited Labeled Data



Difference Between Pre-training and Fine-tuning

1. **Training Efficiency**: Pre-training usually requires substantial computational resources and time because it involves training all model parameters on a large-scale dataset. Fine-tuning is relatively efficient as it builds on the pre-trained model and only requires further optimization on task-specific data.
2. **Model Performance**: The pre-trained model has already learned general language features, allowing fine-tuning to converge faster and perform better on specific tasks. Training a task-specific model from random initialization typically requires more data and time, and its performance may not match that of the pre-training + fine-tuning approach.
3. **Application Scenarios**: Pre-trained models can serve as general-purpose base models suitable for various downstream tasks. Fine-tuning allows for quick adaptation to different task requirements without the need to train a model from scratch.

Tokenization

We can not feed the model with text data, so we need to enumerate them. By tokenization, we split the text data into smaller tokens which correspond to a dense vector representing the semantic meaning of each token in a continuous, high-dimension space. We feed the model with **sequences** which are ordered lists of tokens.

Types of Tokenization

- **Word tokenization**
  - **Each word is a token.**
  - **Semantic meaning is kept.**
  - **OOV and variation (playing,played) problem**
- **Subword tokenization**
  - **Meaningful units of words (un - touch - able)**
  - **Balances other two approaches**
  - **How to split words? This is a drawback**
- **Character tokenization**
  - **Every character is a token**
  - **Handles any input, language**

Tokenization directly affects on data quality hence, the training of the model. As the token size gets smaller, sequence length increases, which means computational complexity.

How does tokenization work?

Before training the model, we choose our tokenization method and dataset. Then we create a vocabulary with them by assigning an ID to every token.

And then we ‘embed’ the ID’s to dense vectors, which will feed the model in training phase.

After capturing relationships between tokens in the vocabulary, we can predict the next token ID’s for inputs, and detokenize them by using the ID-Token relationships.

Frog on a log.

Tokenization

[CLS]	Frog	on	a	log	.	[SEP]
101	2025	2001	1063	8532	2011	102

Embeddings

Embeddings are vector representations of tokens in a high-dimensional space. They are learned during the training of a model. Semantically similar words are closer to each other in this space.

Thus, models understand the relationship between tokens efficiently.

Good -common- example: Queen and king is close in vector space but differ on some features that reflect gender.

Every token ID is mapped to a vector, that is found in a lookup table (embedding matrix). Those vectors are initialized randomly (usually). And during training, they are updated through backpropagation.

Some common embedding techniques include:

1. **Word2Vec**: Creates word embeddings by predicting the context of words (CBOW) or the target word from the context (Skip-Gram).
2. **GloVe (Global Vectors for Word Representation)**: Learns word vectors by factoring a word co-occurrence matrix, emphasizing global statistics.
3. **FastText**: Similar to Word2Vec but includes subword information, allowing it to better handle rare or unseen words.
4. **ELMo (Embeddings from Language Models)**: Contextual embeddings that generate different vectors for the same word based on its context.
5. **BERT (Bidirectional Encoder Representations from Transformers)**: Provides contextualized embeddings for tokens by considering both left and right context.

This one is additional information.

How are LLMs evaluated?

Perplexity: Measure of how well the model predicts the next token.

If a model gives high probability to the correct next words in a sentence, the perplexity will be low, meaning it's a good model.

- **Low perplexity**: Predicting "apple" after "I like to eat an …" is easier for the model (higher probability).
- **High perplexity**: Predicting something nonsensical like "banana" after "I like to read a …" would lead to higher perplexity.

Accuracy: The percentage of correct predictions by the model

Example: In a sentiment analysis task (positive/negative):

- If the model correctly predicts 90 out of 100 samples as positive or negative, accuracy = 90%.

BLEU (Bilingual Evaluation): Compares model-generated to a reference translation by looking at overlapping n-grams.

Example: For the sentence "The cat is playing in the garden" (reference), if the model generates "The cat plays in the garden," BLEU would compare how many 1-grams ("The", "cat", etc.) and 2-grams ("The cat", "cat plays", etc.) overlap. High overlap = high BLEU score.

ROUGE (Recall-Oriented): Similar to BLEU but focuses on how much of the reference is captured by generated text (recall) in summarization tasks.

and of course**, Human Evaluation.**

Use cases of LLMs

1. Content generation → ChatGPT, Claude

2. Translation and localization → Falcon LLM and NLLB-200

3. Search and recommendation → Bard

4. Virtual assistants → Alexa, Google Assistant

5. Code development → StarCoder

6. Sentiment analysis → Grammarly

7. Question answering → LLaMA

8. Market research → Brandwatch,Talkwalker

9. Education & research → Duolingo

10. Classification → Cohere Classify