LLM’S

FIRST WEEK

Gülşen Sabak
Bogazici University Computer Engineering

AGENDA

- What is an LLM?
- How does ıt dıffer from other types of ML models?
- Transformer Architechture
- Why is this revolutionary for nlp?
- Pre-training of LLM’s
- Fine tuning on LLM’s
- The difference between pre-training and fine-tuning
- Tokenization on LLM’s
- Embeddings on LLM’s
- Evaluation of LLM’s
- Real World Applications of LLM’s

WHAT IS AN LLM?

- base concept -> next word prediction
- neural networks specifically designed to process and generate human language at scale.
- They learn patterns in language by training on vast amounts of text data, developing an ability to understand context and generate coherent responses across a wide range of topics.

HOW DOES IT DIFFER FROM OTHER TYPES OF ML MODELS?

- **traditional ml models:**
  - handle structured data with clear input-output relationship
  - excel at specific tasks

- **earlier NLP models:**
  - often specialized for specific tasks and required separate models for different functions
  - use simpler architechture and smaller datasets
  - focusing on specific linguistic features or patterns

- **LLM's:**
  - handle unstructured text data
  - perform multiple tasks without being explicitly programmed for each one
  - use sophisticated transformer architechture
  - process long range dependencies in text and understand context

TRANSFORMER ARCHITECHTURE

- The Transformer architecture is a foundational model in natural language processing (NLP) and serves as the backbone for large language models (LLMs). Introduced in the paper "Attention is All You Need"

- The core innovation of Transformers is their ability to process all words in a sequence simultaneously while understanding how each word relates to all others, primarily through the self-attention mechanism.

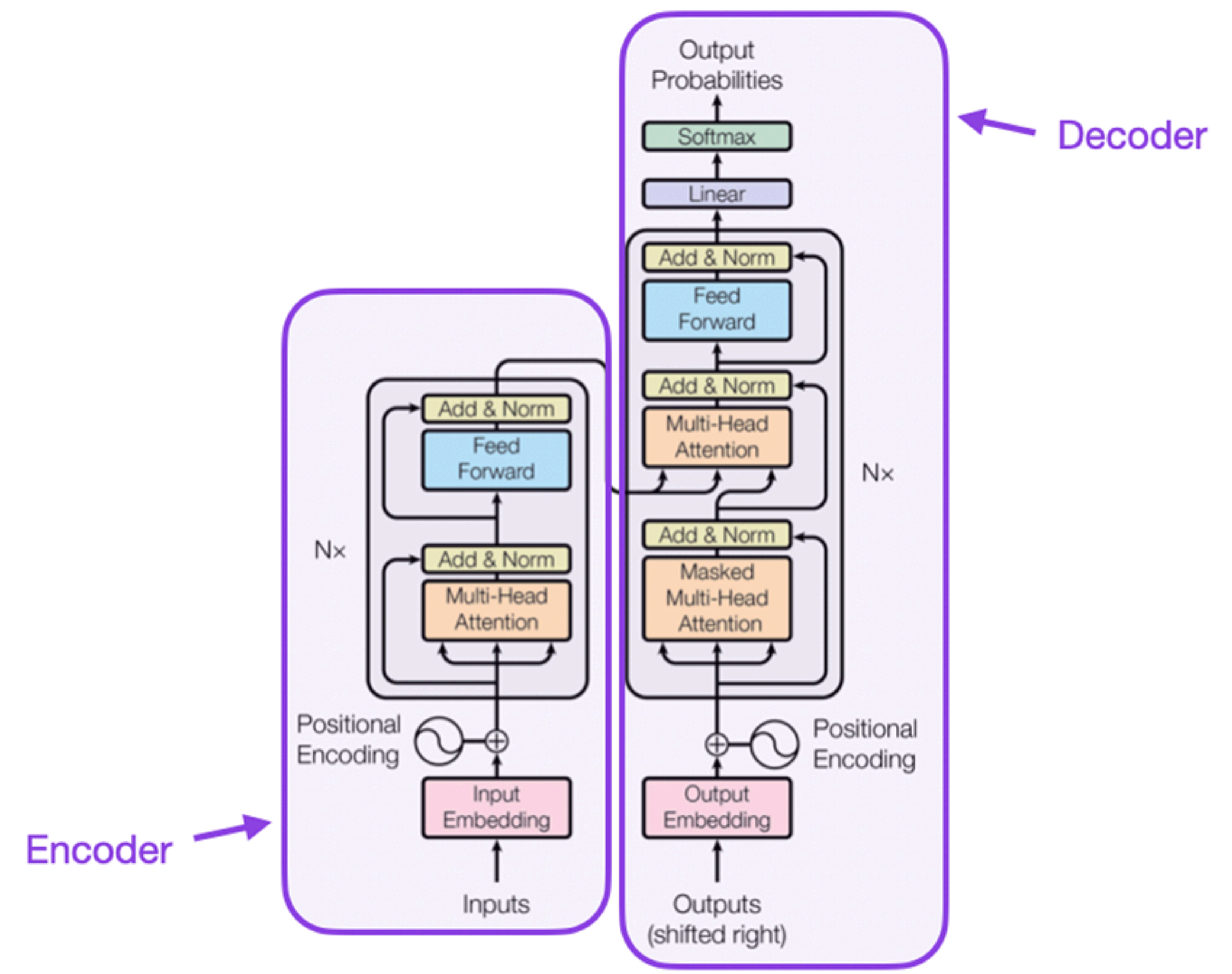


Figure 1: The Transformer - model architecture.

TRANSFORMER ARCHITECHTURE

- **Embedding Layer:**
  - Trainable vector embedding space. Each token represented as a vector and occupies a unique location within that space.
- **Positional Encoding:**
  - Since Transformers process all words simultaneously, they need a way to understand word order
  - Sinusoidal position embeddings are added to word embeddings to encode position information
- Token embedding + positional encoding works parallel for each token

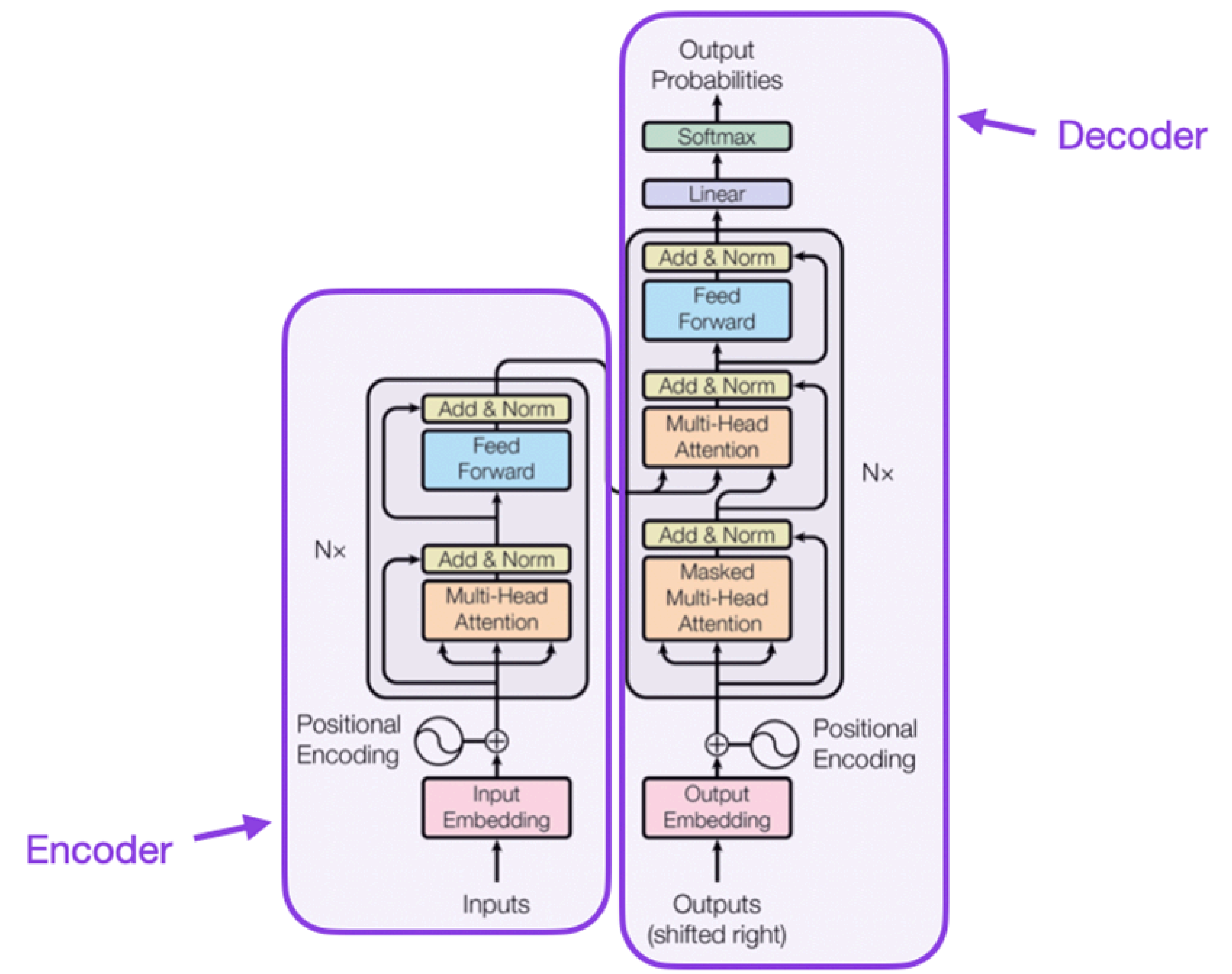


Figure 1: The Transformer - model architecture.

TRANSFORMER ARCHITECHTURE

- **Self Attention Layer:**
  - The heart of the Transformer that allows it to weigh the importance of different words in relation to each other
  - For each word, it creates three vectors: Query (Q), Key (K), and Value (V)
  - The model computes attention scores by comparing each word's query vector with every other word's key vector
  - These scores determine how much focus to place on other words when encoding the current word
  - Captures the contextual dependencies between words

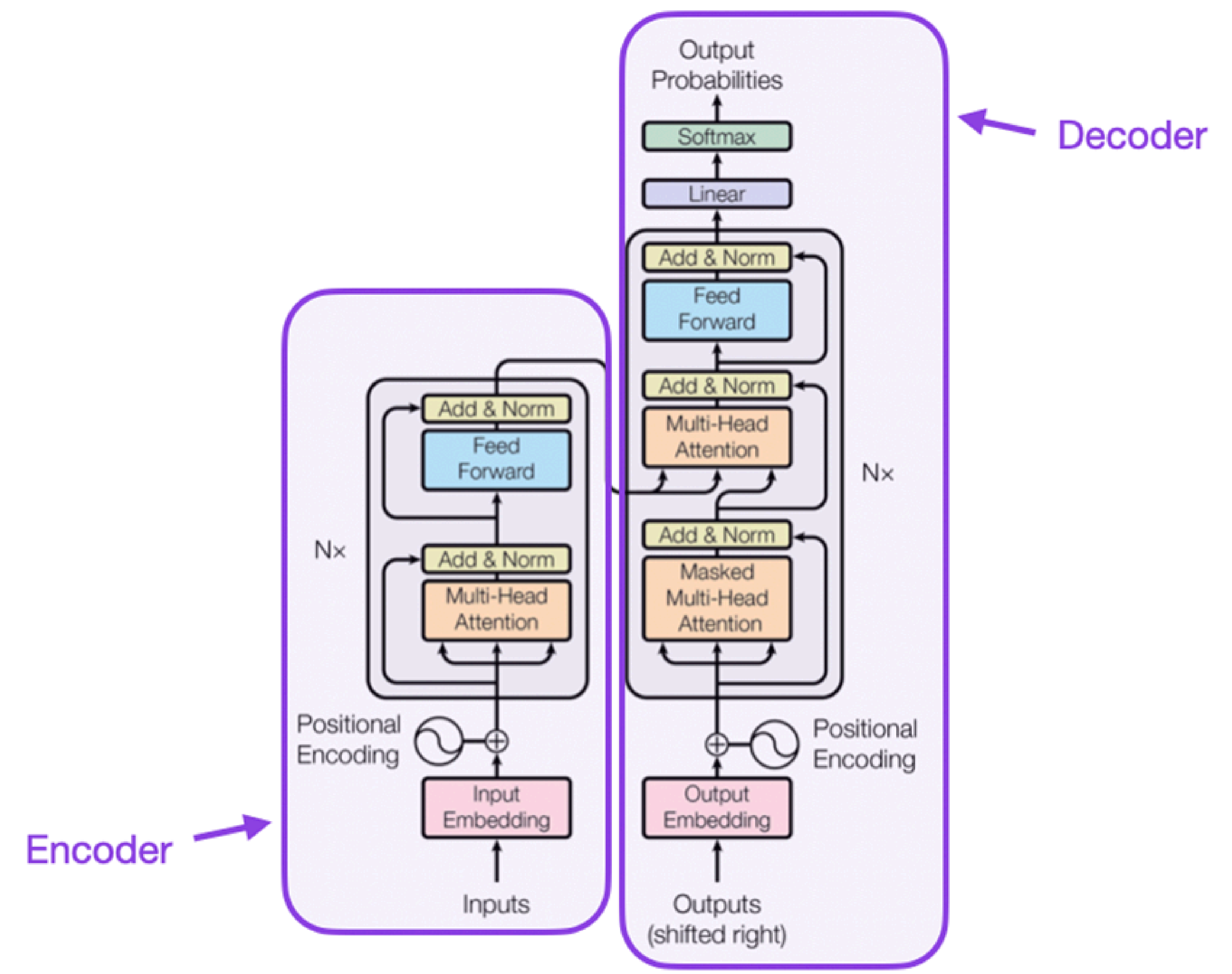


Figure 1: The Transformer - model architecture.

TRANSFORMER ARCHITECHTURE

- **Multi-Head Attention:**
  - Multiple sets of attention mechanisms run in parallel, independently
  - Each "head" can focus on different types of relationships
  - Example: One head might focus on subject-verb relationships, while another captures adjective-noun relationships
  - 12 - 100 layers are enough

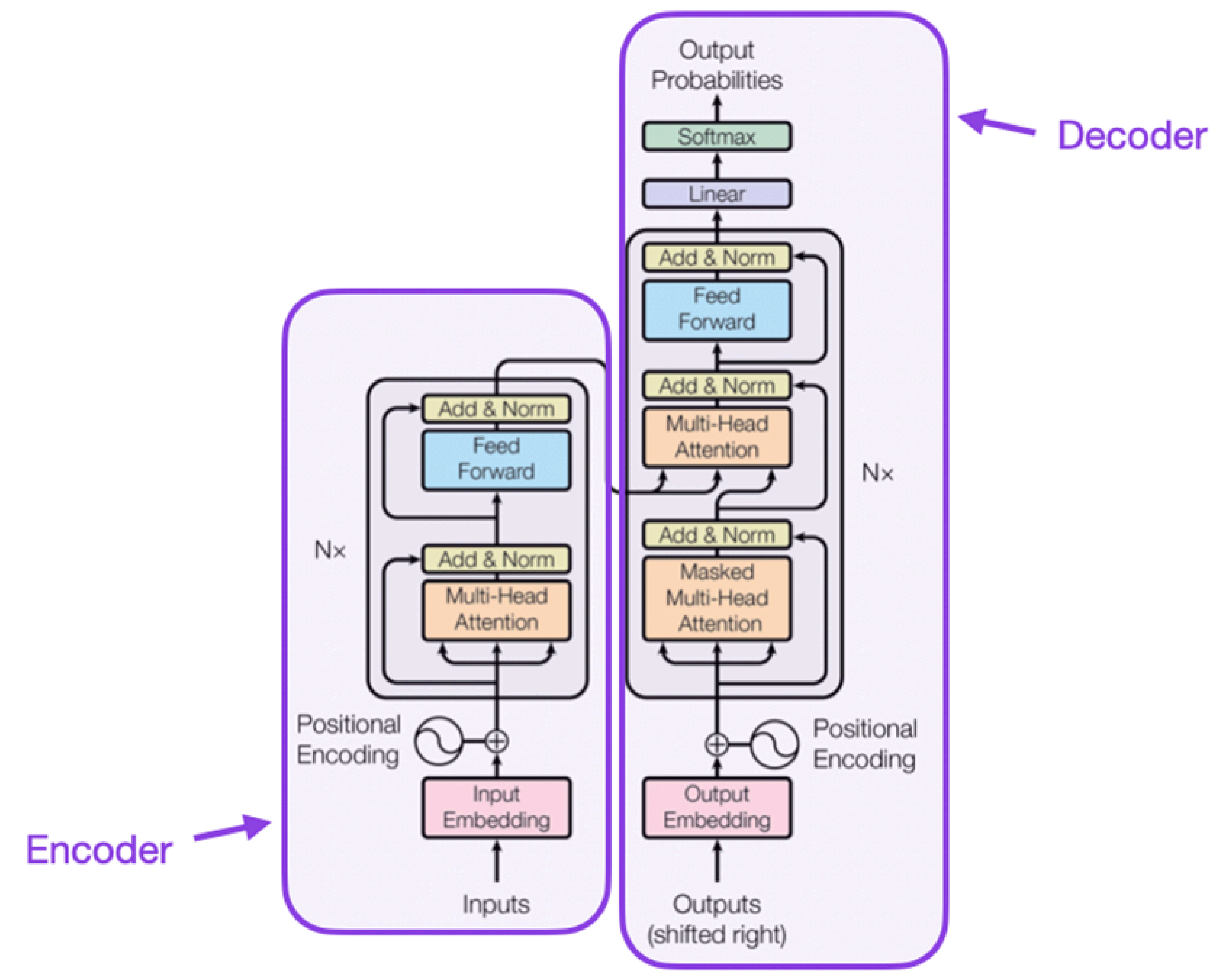


Figure 1: The Transformer - model architecture.

TRANSFORMER ARCHITECHTURE

- **Feed Forward:**
    - Process the output to learn complex relationships
    - helps to create more meaningful and complex embeddings
    - with the help of activation functions like ReLU, model is being stronger
    - It increases the data dimension(x4) and then reduce it. With that way model learns the hidden representations

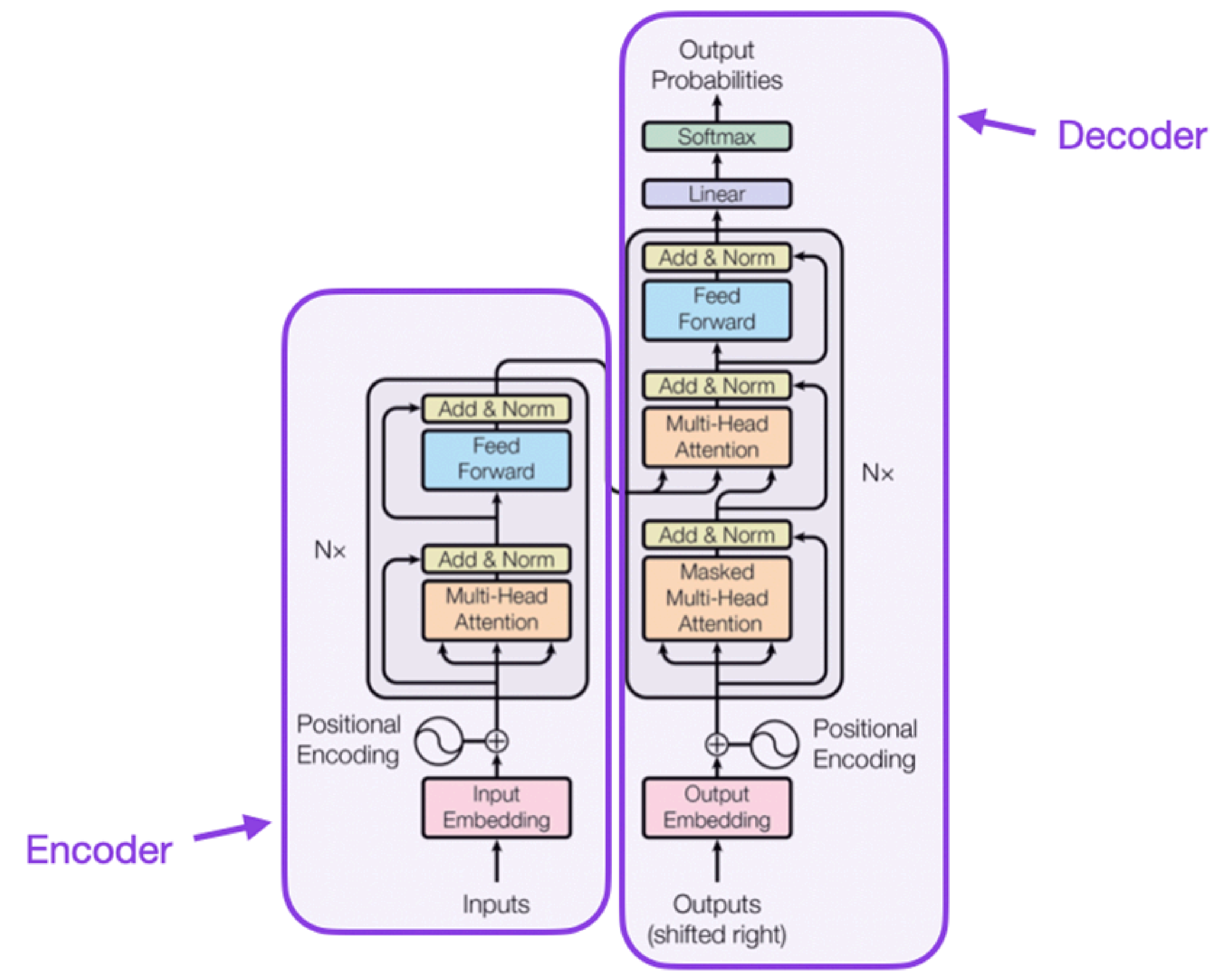


Figure 1: The Transformer - model architecture.

WHY IS THIS REVOLUTIONARY FOR NLP?

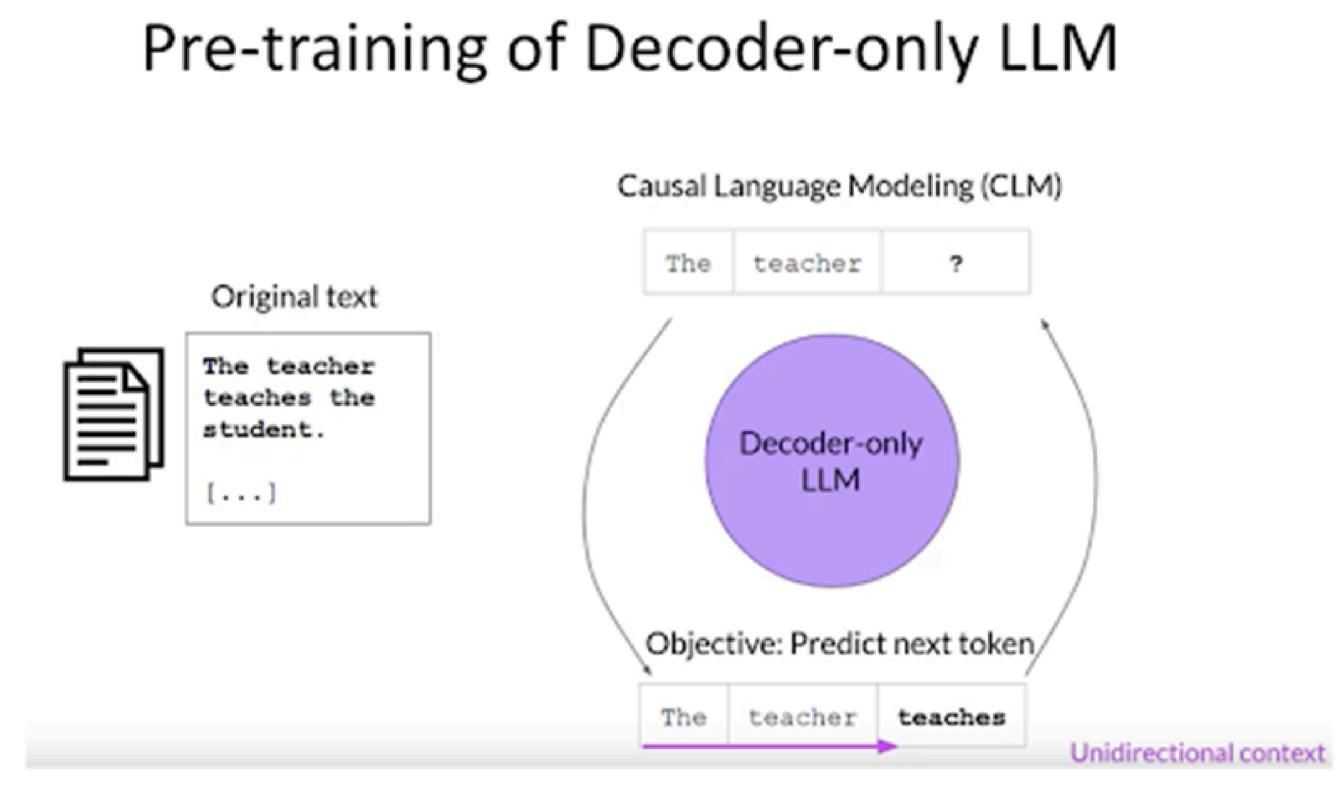
- **Parallel Processing:**
  - Unlike RNNs, which process text sequentially, Transformers process all words simultaneously
  - This enables much faster training and inference
  - Makes it practical to train on massive datasets
- **Better Long-Range Dependencies:**
  - Can directly model relationships between any two words, regardless of distance
  - Previous architectures struggled with long-distance relationships
- **Versatility:**
  - The same basic architecture works well for many different NLP tasks
  - Can be adapted for text generation, translation, summarization, and more
  - Supports transfer learning effectively

PRE TRAINING OF LLM'S

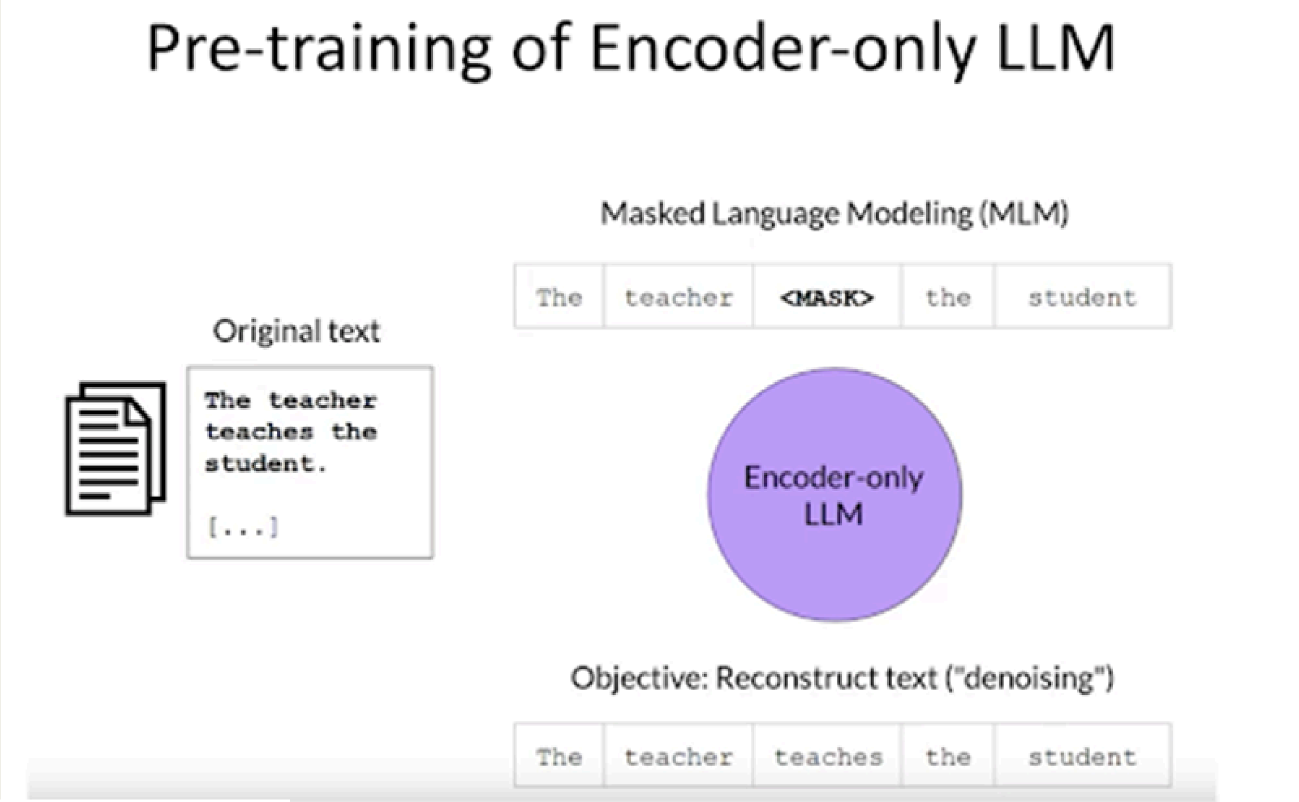
- LLM's encode a deep statistical representation of language. Understanding happens during pre-training
- Model is pre-trained with unstructured textual data
- During pre-training model weights get updated to minimize loss

- Steps of pre-training:
  - Data Collection and Preparation
    - Gathering massive amounts of text data from diverse sources (books, websites, articles)
    - Tokenizing text into subword units that the model can process
  - Objective Setting
    - Most common is the "next token prediction" task (autoregressive training)
    - The model learns to predict the next word given previous context
    - Some models also use masked language modeling, predicting hidden words in a sentence

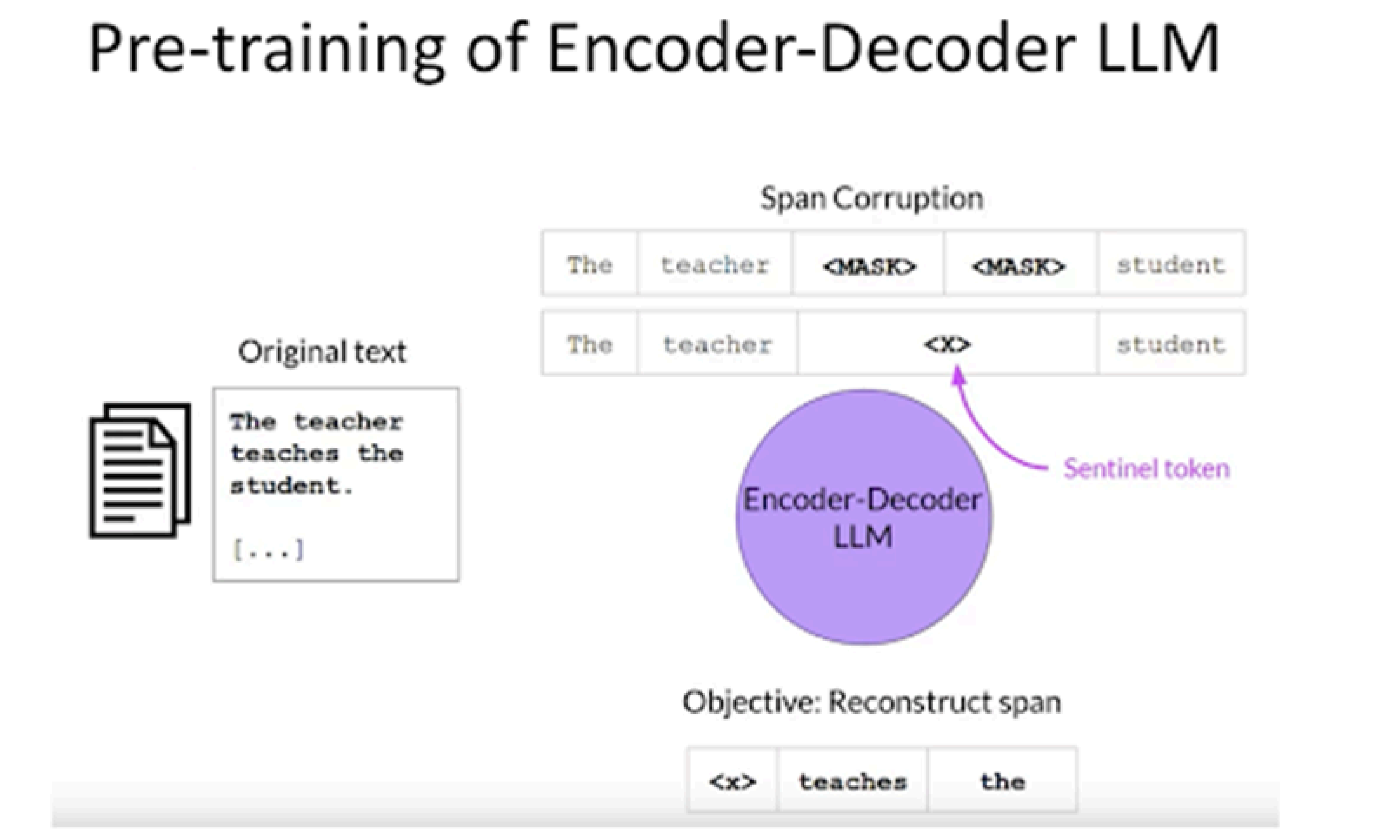
PRE TRAINING OF LLM'S



Training objective is to predict the next token



Training objective is to reconstruct the masked text



Training objective is to reconstruct the span of tokens called the sentinel token

PRE TRAINING OF LLM'S

- Steps of Pre-training:
  - Training Process
    - Model starts with random weights and learns patterns from the data
    - Processes text in chunks/windows (typically 512-2048 tokens)
    - Updates its parameters using backpropagation and optimization algorithms
    - Training continues until performance plateaus or computational budget is reached
  - Outcomes
    - Model develops broad understanding of language patterns
    - Learns grammar, facts, reasoning capabilities
    - Acquires general knowledge across many domains
    - Can perform basic tasks without additional training

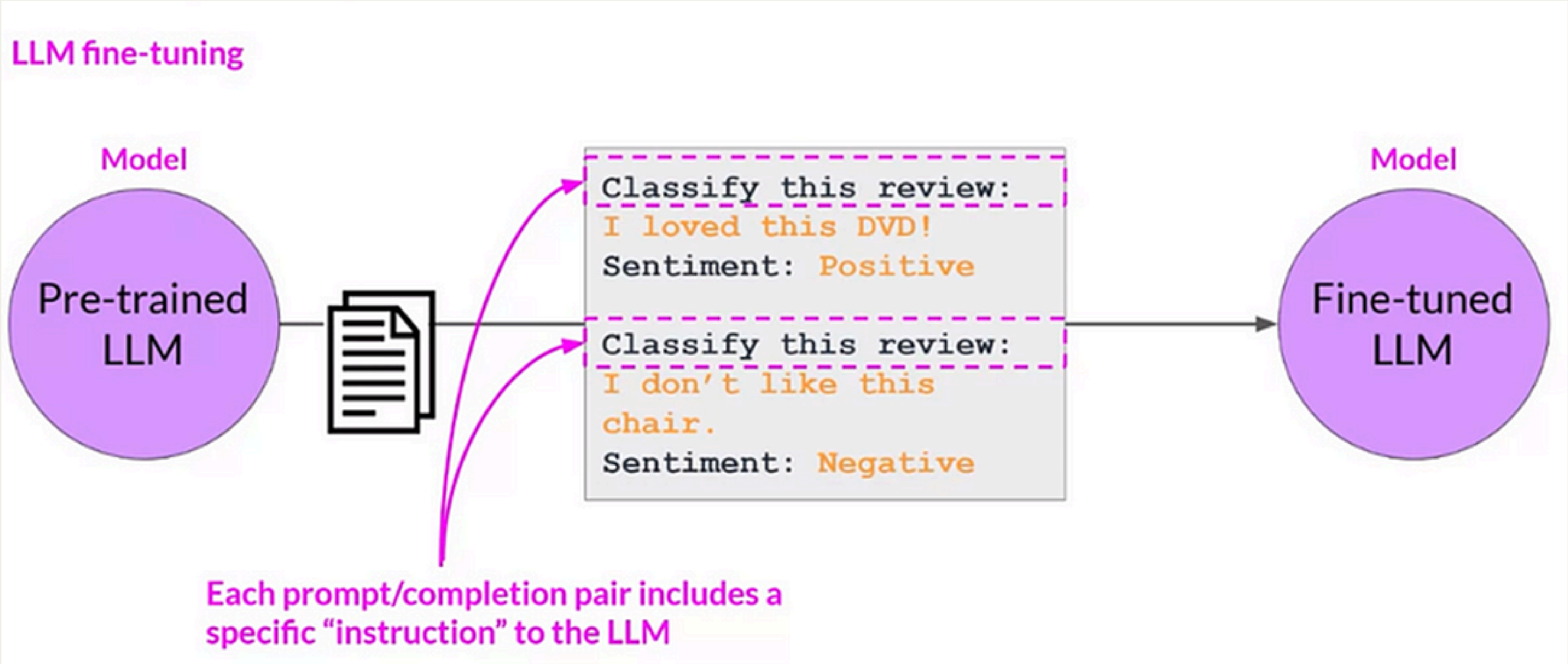
FINE TUNING OF LLM'S

- In context learning may not work for smaller models since examples take up space in the context window. Instead try fine tuning the model
- Fine tuning is taken up supervised learning with labeled examples.
- Aim of fine tuning: Updating the weights of LLM to gain from space and increase accuracy of the model

Instruction Fine Tuning:

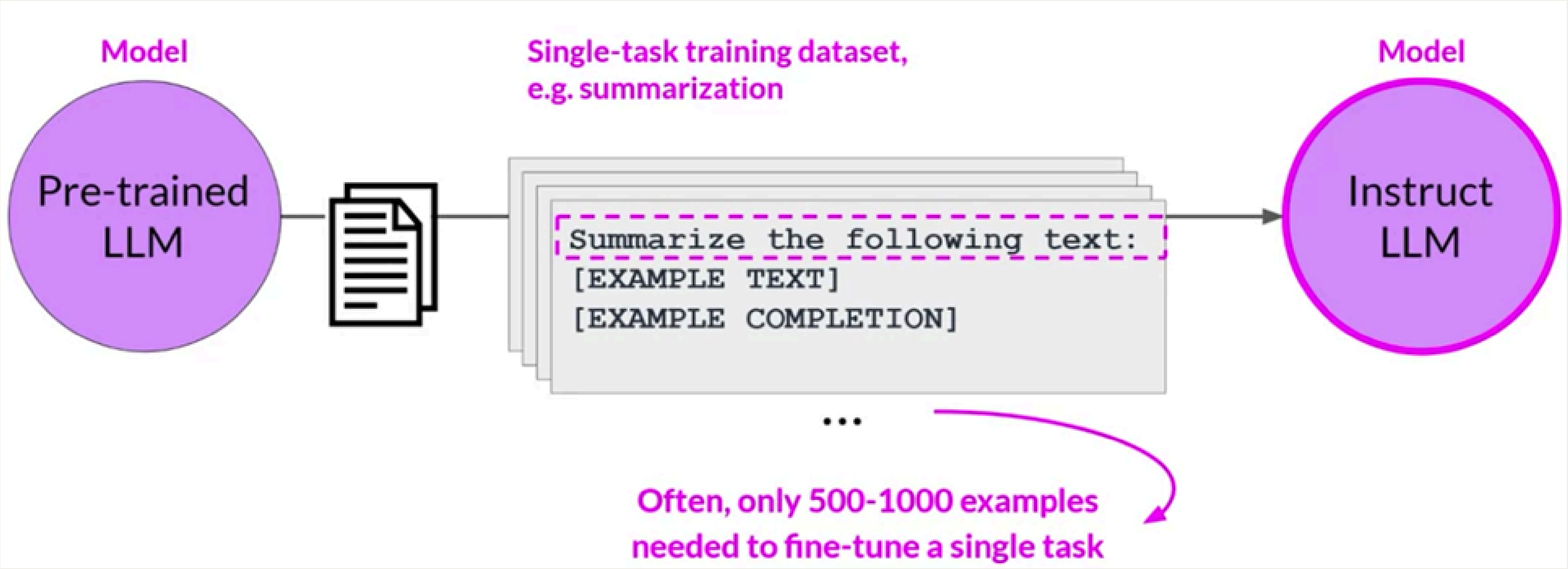
• Good for improving a model's performance on a variety of tasks,

• IFT trains the model using examples that demonstrate how it should respond to a specific instruction



FINE TUNING OF LLM'S

- Fine tune a pretrained model to improve performance on only one task
- 500 – 1000 examples is enough

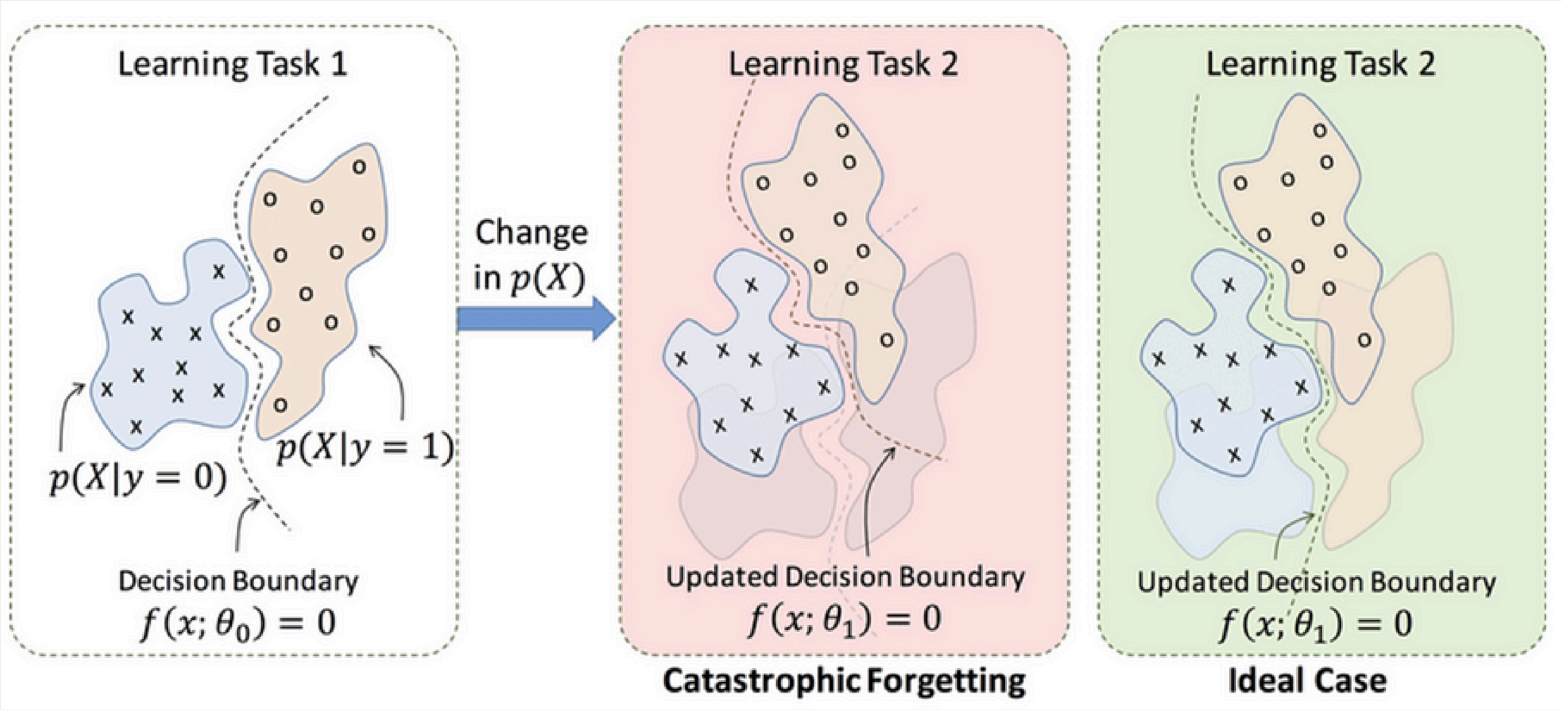


- Potential downside:
  - Catastrophic forgetting:
  - Models tend to forget what they were originally trained on when you do a fine tuning. If you do fine tuning too much, you may end up with catastrophic forgetting

FINE TUNING OF LLM'S

- How to avoid catastrophic forgetting?

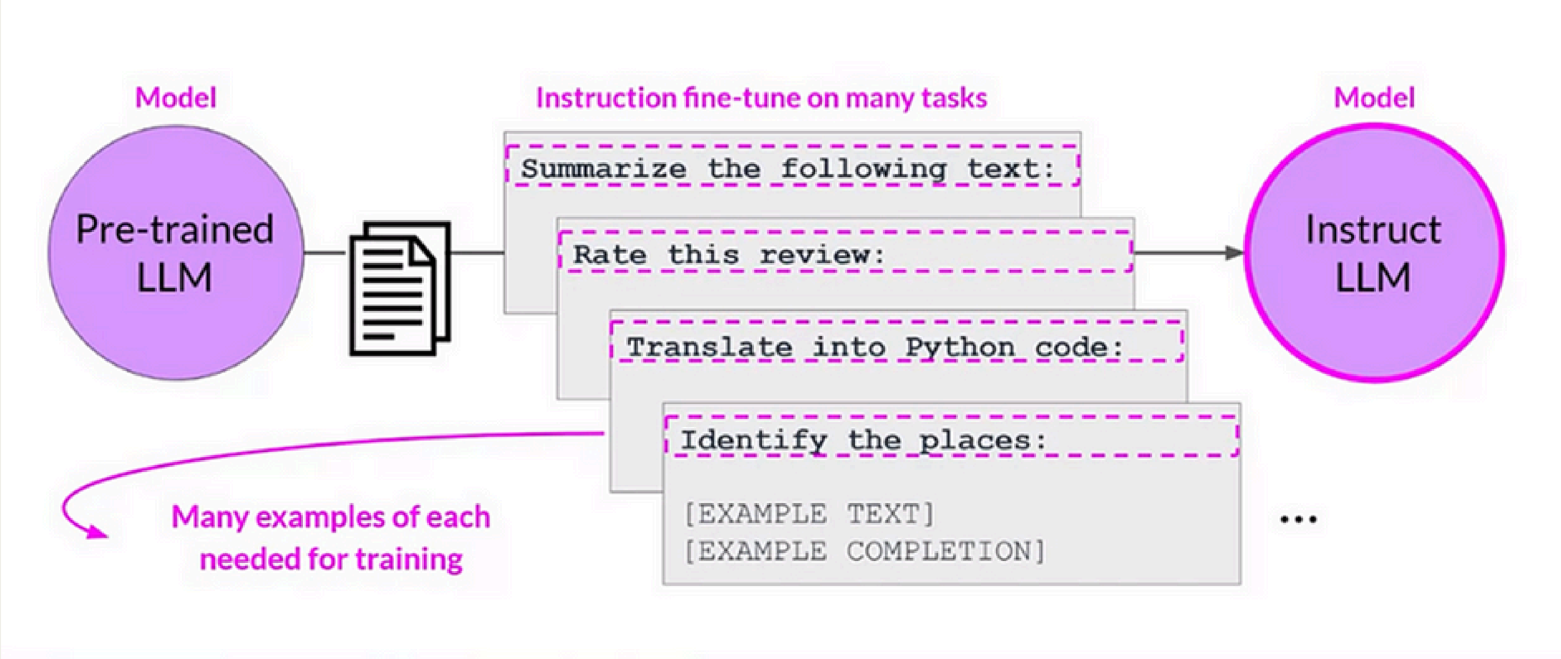
1) Do not have to
2) Fine tune on multiple tasks at the same time
3) Consider parameter efficient fine tuning (PEFT)



- Multi-task instruction fine tuning:

Training dataset is comprised of example inputs and outputs for multiple tasks

Drawback:
Requires a lot of data 50 – 100000 examples in training data set



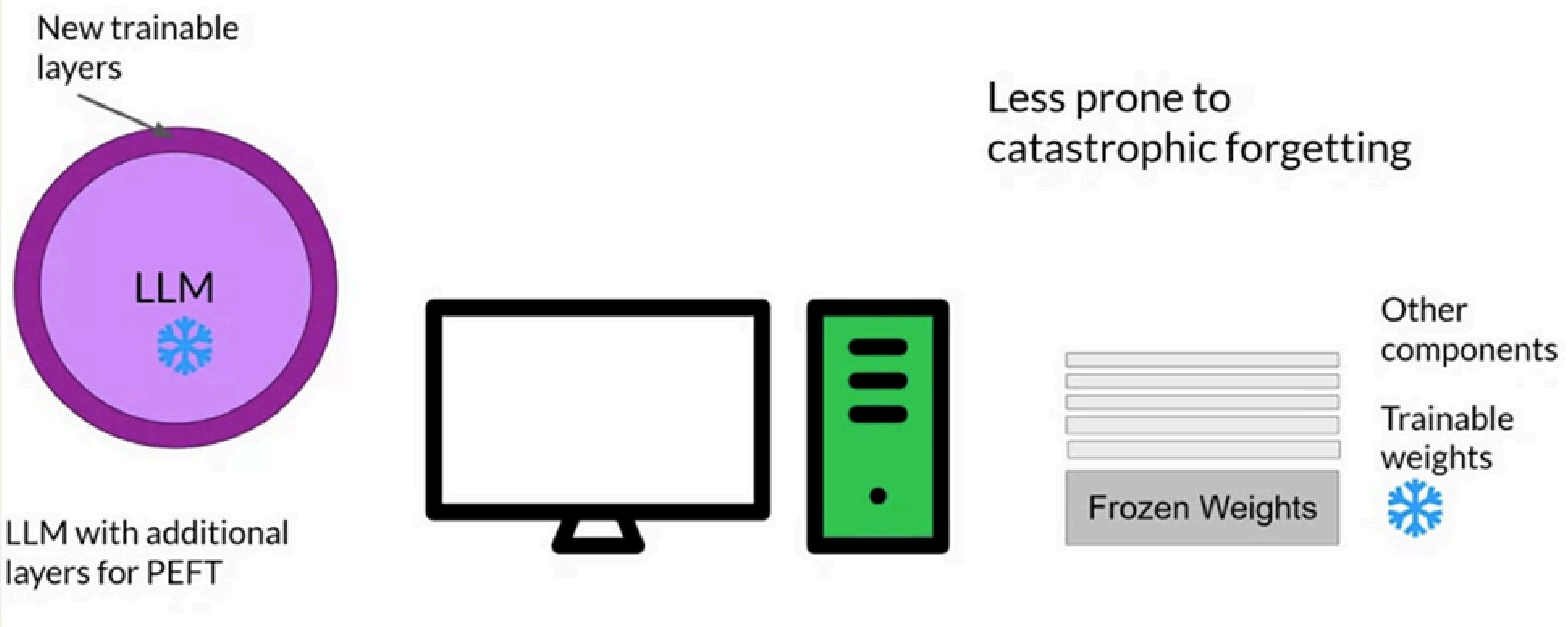
FINE TUNING OF LLM’S

- Steps of Fine Tuning:
  - Loading and preparing the dataset
  - Preprocessing the data
  - Model preparation
  - Initial Predictions
  - Training the model
  - Evaluating the model
  - Making predictions with the trained model

FINE TUNING OF LLM’S

- PEFT

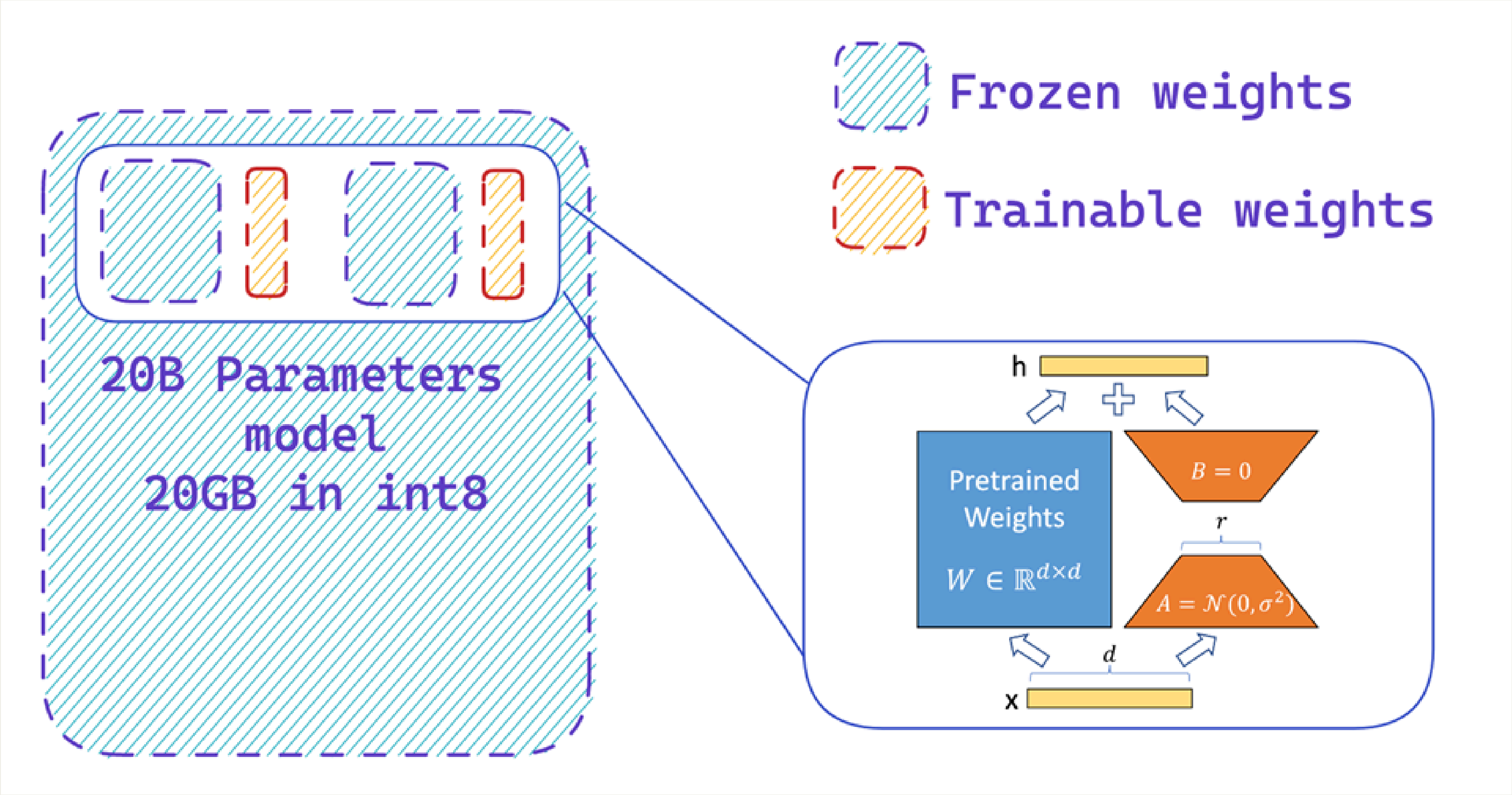
•Freeze the original model weights.
Add a small number of new Parameters or layers and fine tune only new components



•PEFT can be performed on a single GPU
•PEFT is less prone to catastrophic forgetting problems of fine tuning

FINE TUNING OF LLM'S

- LoRA
- Freeze the original model's weights
- Inject 2 rank decomposition matrices (adapters)
- Train the weights of the smaller matrices



THE DIFFERENCE BETWEEN PRE-TRAINING AND FINE-TUNING

- **Pre-training:**
  - Uses vast amounts of general text data
  - Higher learning rates
  - More computational resources
  - Builds foundational capabilities
  - More expensive and time-consuming

- **Fine-tuning:**
  - Uses targeted, task-specific data
  - Lower learning rates
  - Less computational intensity
  - Preserves general knowledge while specializing
  - More accessible to organizations with limited resources

TOKENIZATION ON LLM'S

- Tokenization is the process of breaking text into smaller units called tokens, which can be words, subwords, or characters. LLMs process text as tokens rather than raw words.
- How Tokenization works:
  - Word-based: Splits text into words (e.g., "Hello world!" ↠ ["Hello", "world!"]).
  - Subword-based (Byte-Pair Encoding, WordPiece): Splits words into meaningful parts (e.g., "unhappiness" ↠ ["un", "happiness"]).
  - Character-based: Each character is a token, useful for handling unknown words.
  - Byte-level (GPT-style): Works at the byte level, handling all languages and special characters efficiently.
- Good tokenization improves model comprehension and generation quality.
- Proper tokenization directly impacts model accuracy, speed, and cost in LLMs.

EMBEDDINGS ON LLM'S

- Embeddings are numerical vector representations of words, phrases, or sentences that capture their meanings in a multi-dimensional space. LLMs use embeddings to understand and process language efficiently.
- How embeddings work:
  - Conversion: Words/sentences are mapped to dense vectors (arrays of numbers).
  - Semantic Relationships: Similar words have closer vectors (e.g., "king" and "queen" are near each other).
  - Context Awareness: Modern embeddings (e.g., from transformers like BERT) capture meaning based on context. Same word can have different vectors based on usage
- Types of Embedding:
  - Word2Vec, GloVe: Static word embeddings (fixed meaning per word).
  - Transformer-Based (BERT, GPT): Contextual embeddings that change based on sentence structure

EMBEDDINGS ON LLM'S

- Modern embeddings typically use 256-1024 dimensions
- Each dimension may capture different aspects of meaning
- Higher dimensions allow for richer representations
- Trade-off between expressiveness and computational cost
- Words with similar meanings cluster together in vector space
- Captures multiple types of relationships:
  - Synonyms (happy/joyful)
  - Antonyms (hot/cold)
  - Analogies (Paris:France :: Berlin:Germany)
  - Categories (apple/orange/banana)

EVALUATION OF LLM'S

- **Perplexity (PPL):** Measures how well a model predicts the next word in a sequence; lower values indicate better performance. Calculated as the exponential of the average negative log-likelihood. Useful for comparing models during training
    - Limitation: Doesn't directly measure task performance or output quality
- **Accuracy:** Percentage of correct predictions
- **F1 Score:** Balances precision and recall
- **ROC-AUC:** Measures discrimination ability
- **BLEU:** Compares generated text with reference translations
- **ROUGE:** Measures overlap with reference summaries
- **METEOR:** Considers synonyms and stems in evaluation
- **GLUE/SuperGLUE:** Natural language understanding tasks
- **Humans rate** outputs for: Quality, Coherence, Factual accuracy, Helpfulness
- **A/B Testing**
    - Compare outputs from different models
    - Humans choose preferred response

REAL WORLD APPLICATIONS OF LLM'S

- Chatbots & Virtual Assistants – Customer support, healthcare Q&A, AI tutors (e.g., ChatGPT, Alexa).
- Content Generation – Marketing copy, blog writing, code assistance (e.g., GitHub Copilot).
- Translation & Language Processing – Real-time translation, transcription, localization.
- Education – AI tutors, automated grading, personalized learning.
- Healthcare – Medical documentation, drug discovery, diagnosis assistance.
- Finance & Business – Fraud detection, market analysis, legal document review.
- Personalization – Product recommendations, content curation, HR screening.
- Scientific Research – Paper analysis, AI-driven simulations, debugging assistance.

Thank you!

Gülşen Sabak
Bogazici University Computer Engineering