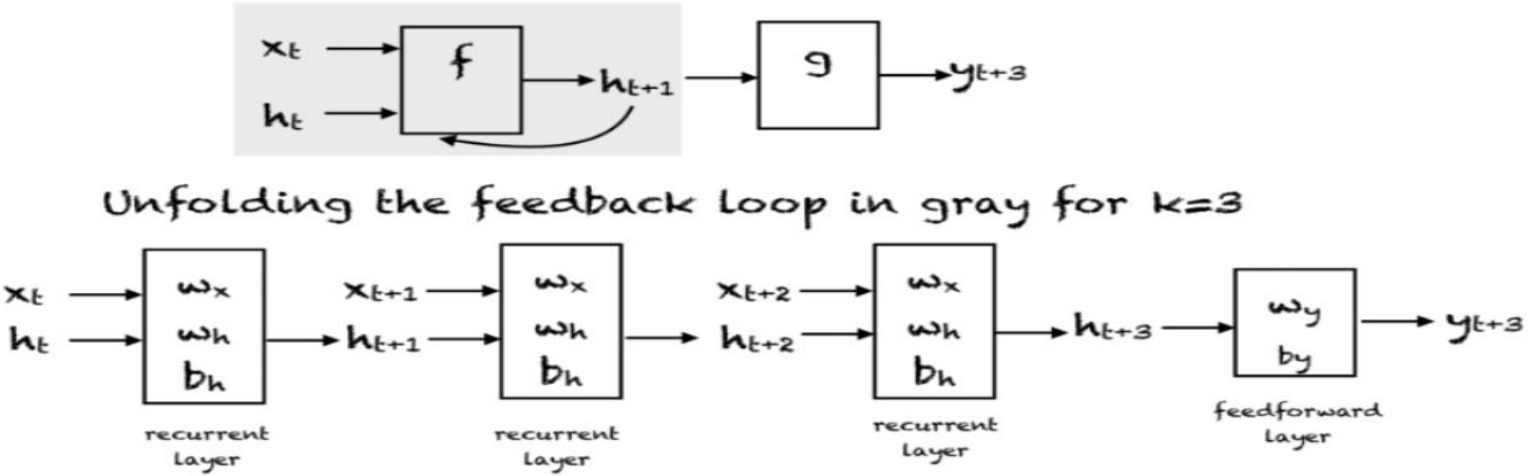
TRANSFORMER ARCHITECTURE
FROM SCRATCH

Poojitha Kathi

BACKGROUND

RNN:



Limitations of Sequential Nature: The sequential nature of RNNs poses a challenge for parallel processing. Parallelization is important for efficient training, especially for long sequences, but memory constraints limit the ability to batch process multiple examples simultaneously.

Introduction to Transformer Architecture

Transformers are a type of deep learning model originally designed for natural language processing (NLP) tasks.
The Transformer model introduces a new architecture that eliminates the need for recurrent computations entirely. Instead, it relies solely on attention mechanisms to capture global dependencies between the input and output sequences.

Advantages of the Transformer:

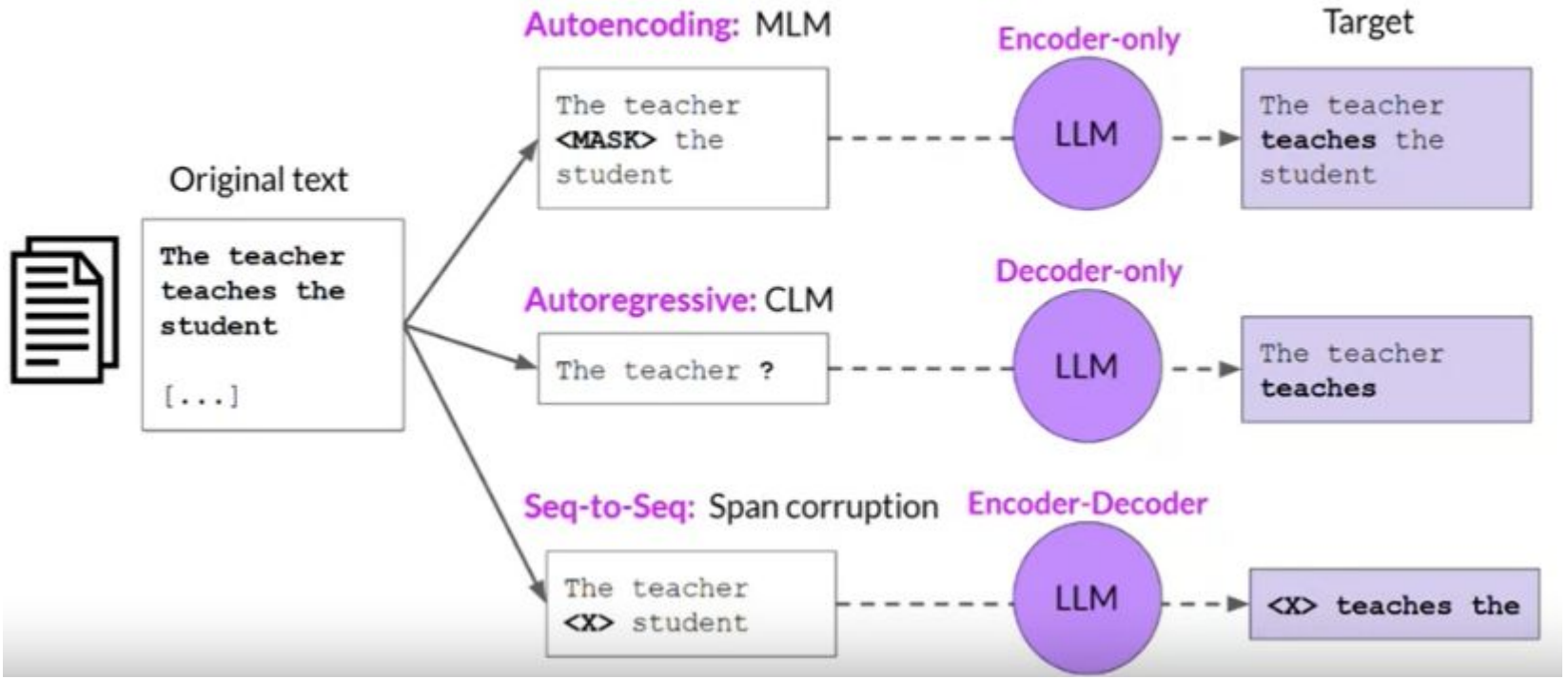
Parallelization: By removing the sequential dependency, the Transformer allows for much greater parallelization. This leads to faster training times and more efficient use of computational resources.

Performance: The Transformer can achieve state-of-the-art performance in tasks like machine translation. The text notes that it reached high translation quality after just twelve hours of training on eight P100 GPUs.

Different Transformer Architectures

“Understanding transformer architectures and their variants is essential for leveraging their capabilities in various business applications. Whether deploying encoder-only models for contextual understanding, decoder-only models for text generation, or encoder-decoder models for sequence-to-sequence tasks, businesses can harness these models to automate tasks, improve accuracy, and drive innovation in natural language processing and beyond.”

Model architectures and pre-training objectives



Learning Paradigm in Transformer Architectures

Encoder-Only Models (e.g., BERT)

- **Pre-training:** Utilizes unsupervised learning to learn general language representations from large unlabeled corpora (e.g., Wikipedia).
- **Fine-tuning:** Uses supervised learning with labeled data for specific tasks like sentiment analysis or named entity recognition.

Decoder-Only Models

- **Training:** Primarily uses unsupervised learning during pre-training to predict the next token in sequences from large text corpora.
- **Fine-tuning:** Can be fine-tuned on labeled data for tasks such as text completion or dialogue generation, adjusting weights based on specific task objectives.

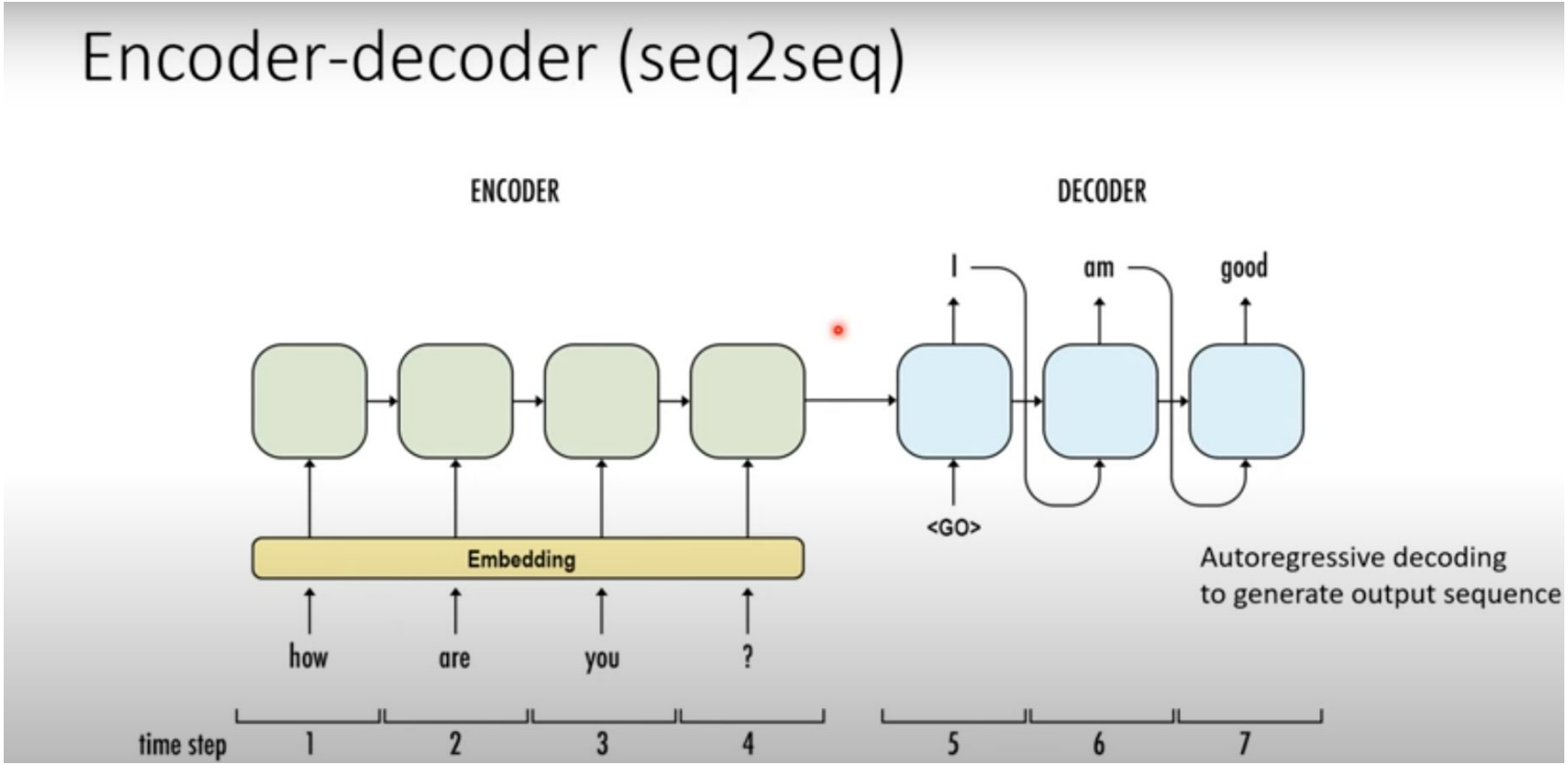
Encoder-Decoder Models

- **Pre-training:** Involves unsupervised learning tasks such as masked token prediction and sequence-to-sequence modeling on large unlabeled datasets.
- **Fine-tuning:** Adapts the model to specific supervised tasks using labeled data, adjusting both the encoder and decoder components for tasks like translation or summarization.

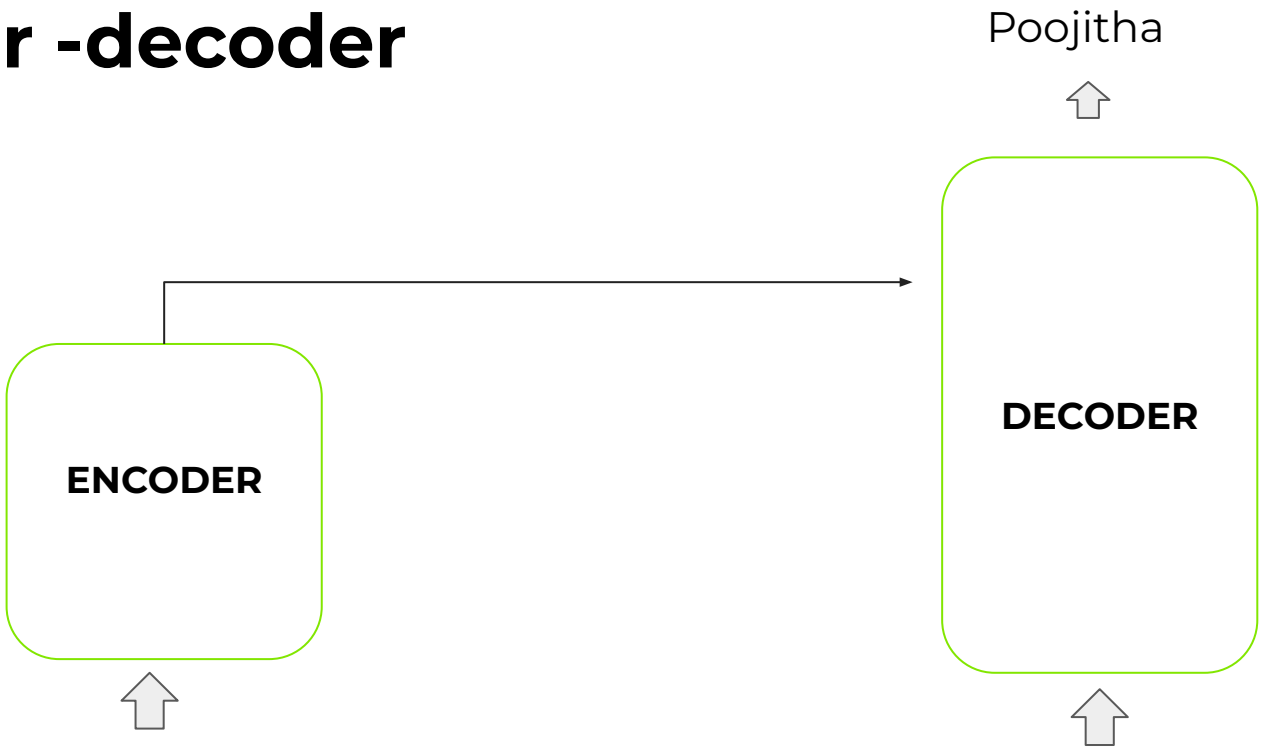
Comprehensive overview of various architectures

Architecture	Key Features	Notable LLMs	Use Cases
Encoder-only	Captures bidirectional context; suitable for natural language understanding	• BERT • Also BERT architecture based RoBERTa, XLNet	• Text classification • Question answering
Decoder-only	Unidirectional language model; Autoregressive generation	• GPT • PaLM	• Text generation (variety of content creation tasks) • Text completion
Encoder-Decoder	Input text to target text; any text-to-text task	• T5 • BART	• Summarization • Translation • Question answering

Encoder -decoder



encoder -decoder



I am Poojitha and I am working as AI/ML Engineer

Who am I?

Encoder-Decoder architecture overview

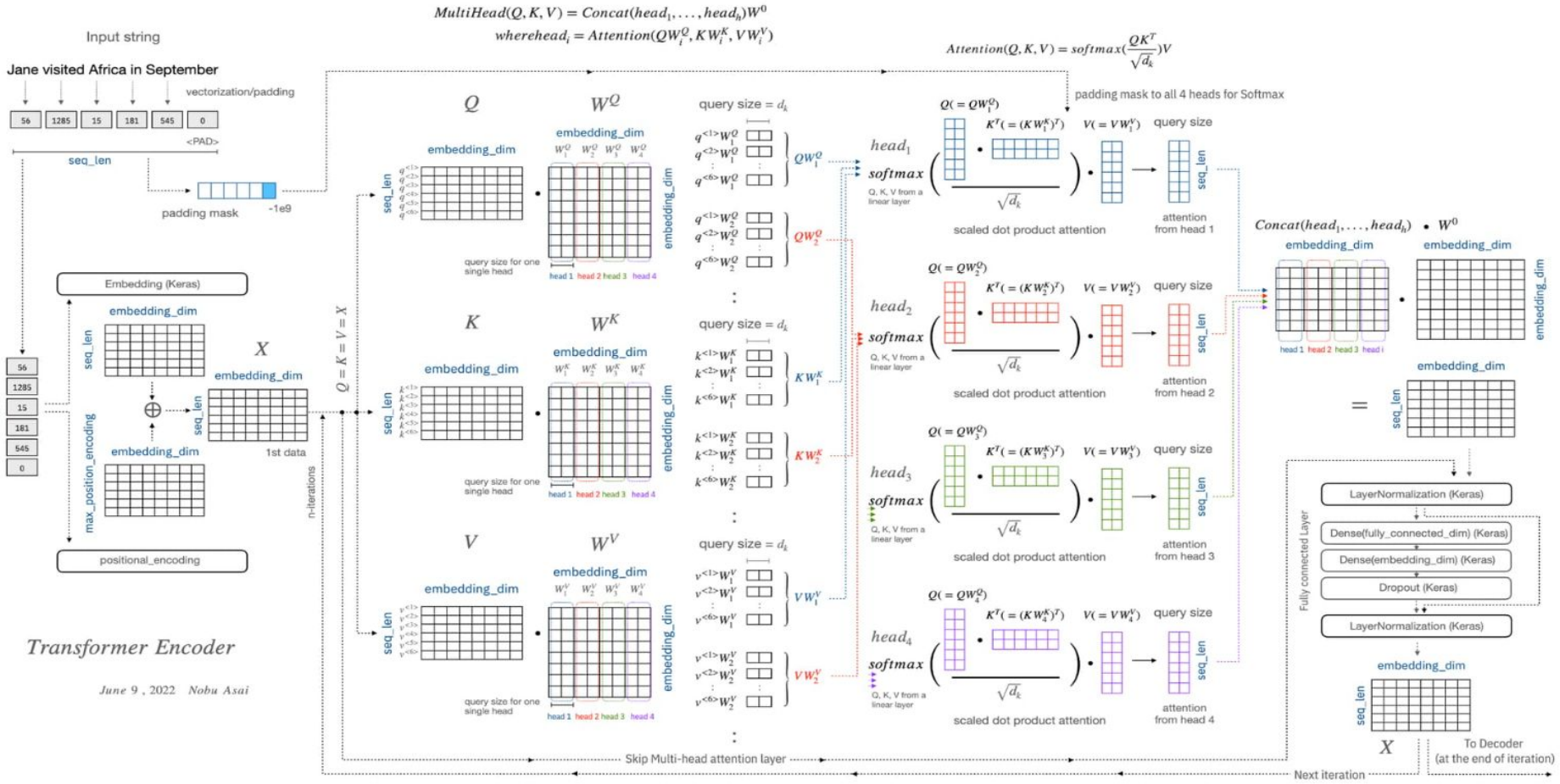
Context embedding= Sequence embeddings +Positional Encoding:
Integration with Embeddings: Adds positional encodings to token embeddings to incorporate sequence order information.

Encoder:

Self-Attention and Feed-Forward: Each layer consists of multi-head self-attention and a feed-forward network, with residual connections and layer normalization.
“encoding comes – vector representation on distance between words and sentences”
“Attention :what part of input should i focus on ?
Self attention -how ith word is relevant to the other words in english sentence
Ith attention vectors capture relation between words”

Decoder:

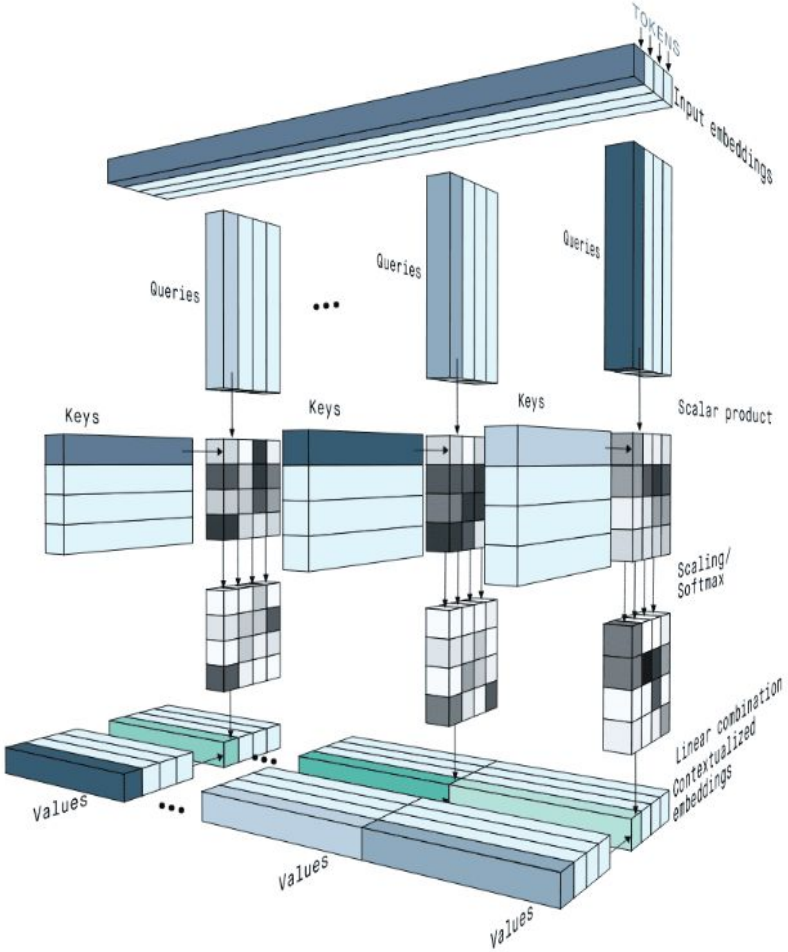
Masked Self-Attention: Prevents future token information flow, maintaining the autoregressive property.
Encoder-Decoder Attention: Focuses on the encoder’s output for generating accurate sequences.W



Transformer Encoder

June 9 , 2022 Nobu Asai

Q, K, and V Matrices, in the context of the self-attention mechanism:



Q (Query): This represents the processed information of the current word. It's a matrix that helps in the scoring process to see how relevant other words are to the current word.

"Represents what we are looking for in the input sequence."

K (Key): This represents the processed information of all the words in the sentence, including the current word. It's used to compute a score that represents the relationship between different parts of the sentence.

"Represents the criteria or the context used to match the queries against the input sequence."

V (Value): This represents the raw information of all words in the sentence. Once the scores between different parts of the sentence are computed, they are used to weight the Value matrix, which in turn gives an aggregated representation of the words in context.

"Represents the actual information or content associated with each key."

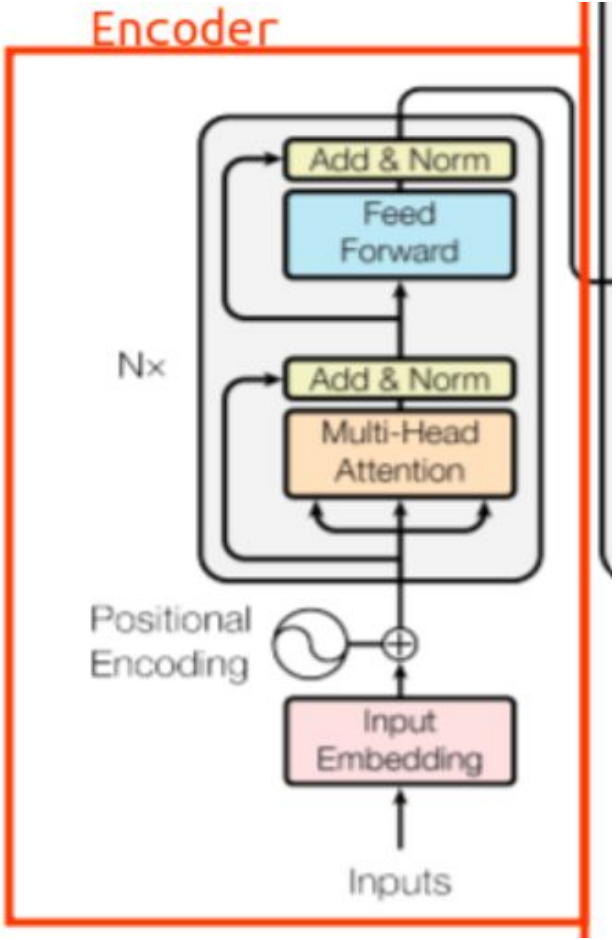
Multi-Headed vs. Single-Headed Attention

In single-headed attention, the Q, K, and V matrices are derived directly from the input, often through different learned weight matrices.

In multi-headed attention, the idea is extended by having multiple sets of weight matrices for Q, K, and V, resulting in multiple heads that attend to different parts of the input space. Each head might learn to pay attention to different aspects or relationships in the data.

One important detail in multi-headed attention is that the dimensions of the Q, K, V are affected by the number of heads.

Encoder



- **Input Embedding + Positional Encoding**
  - Converts tokens into dense vectors and adds positional info.
- **Multi-Head Attention**
  - Focuses on different parts of the sequence simultaneously.
- **Add & Norm**
  - Adds residual connections and normalizes.
- **Feed Forward**
  - Applies two-layer neural network to each token.
- **Add & Norm**
  - Adds residual connections and normalizes.
- **Final Output**
  - Normalized token representations after N layers.

- **Final Output**

$$\text{Output} = \text{LayerNorm}(X + \text{FFN}(X))$$

⇧

- **Add & Norm**

⇧

- **Feed Forward**

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

⇧

- **Add & Norm**

$$\text{Output} = \text{LayerNorm}(X + \text{MultiHead}(X))$$

⇧

- **Multi-Head Attention**

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

⇧

- **Input Embedding + Positional Encoding**

$$\mathbf{E}'_{\text{token}} = \mathbf{E}_{\text{token}} + PE_{\text{pos}}$$

TOKEN EMBEDDING

Input Sentence	Token
I am Poojitha an	I
	am
	Poojitha
	and
	I
	am
	working
	as
	as
	AI/ML
	intern

Embedding: Converts each token (word) in the input sequence into a dense vector representation using a lookup table (embedding matrix). This captures the semantic meaning of each token.

INPUT: I am Poojitha and I am working as AI/ML intern

For the first "I am":

$$\mathbf{z}_1 = \mathbf{e}_{\mathrm{I}} + \mathbf{p}_1$$

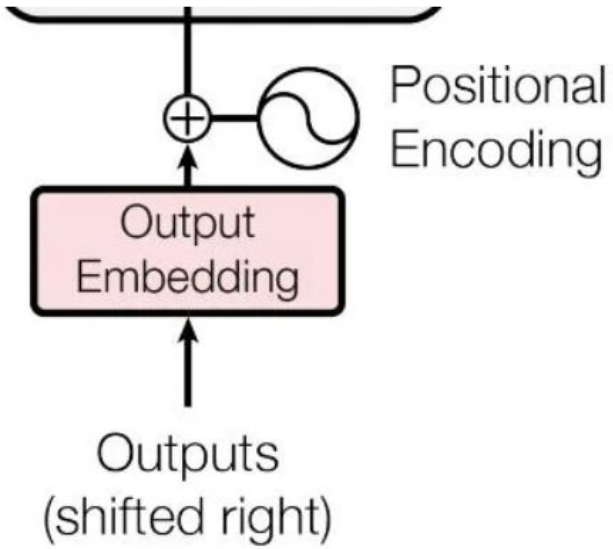
$$\mathbf{z}_2 = \mathbf{e}_{\mathrm{am}} + \mathbf{p}_2$$

For the second "I am":

$$\mathbf{z}_5 = \mathbf{e}_{\mathrm{I}} + \mathbf{p}_5$$

$$\mathbf{z}_6 = \mathbf{e}_{\mathrm{am}} + \mathbf{p}_6$$

Positional Encoding



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

“Same word in different sentences have different meaning here positional encoding comes – vector representation on distance between words and sentences”

This is necessary because the self-attention mechanism in the transformer model doesn't have any inherent sense of word order, which is important in many language tasks.

Use in Tasks:

- **Language Translation:** Maintains word order and context.
- **Text Generation:** Preserves sequence integrity.
- **Document Summarization:** Keeps key points in correct order.

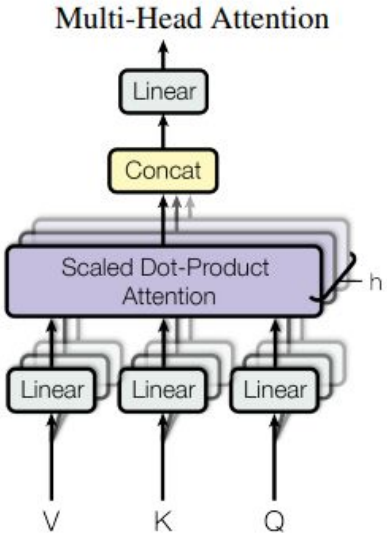
Output Embedding:

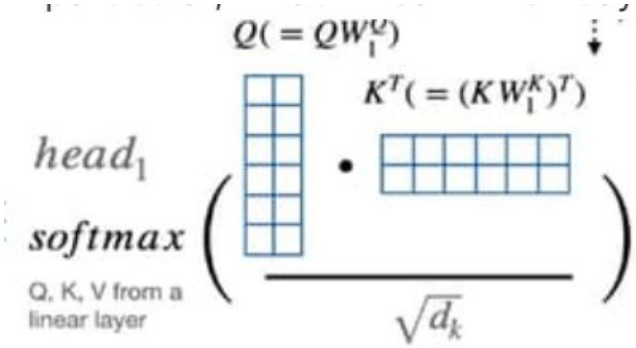
$$\mathbf{E}'_{\text{token}} = \mathbf{E}_{\text{token}} + PE_{\text{pos}}$$

Where:

- $\mathbf{E}_{\text{token}}$ = Original token embedding.
- PE_{pos} = Positional encoding vector.

Contextual Understanding: Preserves context and relationships between words.





Multi-Head Attention

Multi-head attention is a mechanism used in neural networks, particularly in transformers, **to weigh different parts of a sequence differently.**
Attend sto various types of information and relationships within the input sequence similar to multiple filters in convolutional networks to capture different features from input images.
Multi-Head Attention Usage:
Sequential Data: Effective for tasks involving sequences such as text, time-series, or event sequences.
Long-range Dependencies: Captures long-range dependencies efficiently within sequences.
Complex Relationships: Suitable for tasks with complex relationships in data.
Enhanced Representation:
Learns detailed and nuanced content representations.
Contextual Understanding: Beneficial for tasks requiring contextual understanding like translation or summarization.

Leveraging Multi-Head Attention

Multi-head attention allows the model to focus on different parts of the input sequence simultaneously.
Each attention head learns different relationships within the data.
It's effective for capturing dependencies and patterns in sequential data.
Document Understanding is Crucial for use cases dealing with documents
Understanding the relationships between words in documents is vital for our use cases.
MHA Provides the flexibility and power needed to address these challenges effectively.

- **Capturing Contextual Dependencies:**
  Captures dependencies between words or tokens in a document.
- **Attention to Relevant Information:**
  Allows the model to focus on important parts of the document.
- **Enhanced Representation Learning:**
  Learns rich representations of document content.

Cases where Multi-Head Attention may not be used:

- **Simple Classification Tasks**: For tasks not requiring complex relationships in data.
- **Resource Constraints:** When computational resources are limited.
- **Small Datasets:** With small datasets, simpler models may generalize better.
- **Non-Sequential Data:** Other architectures may be more suitable for non-sequential data.

Challenges of Multi-Head Attention:

- **Computational Complexity:** Increased computational requirements due to multiple attention heads.
- **Interpretability:** Understanding contributions of each head can be challenging.
- **Overfitting:** Prone to overfitting, especially with limited data.
- **Attention Spread:** Attention might spread across multiple heads, diluting focus.

Layer Normalization in Encoder-Decoder Architecture

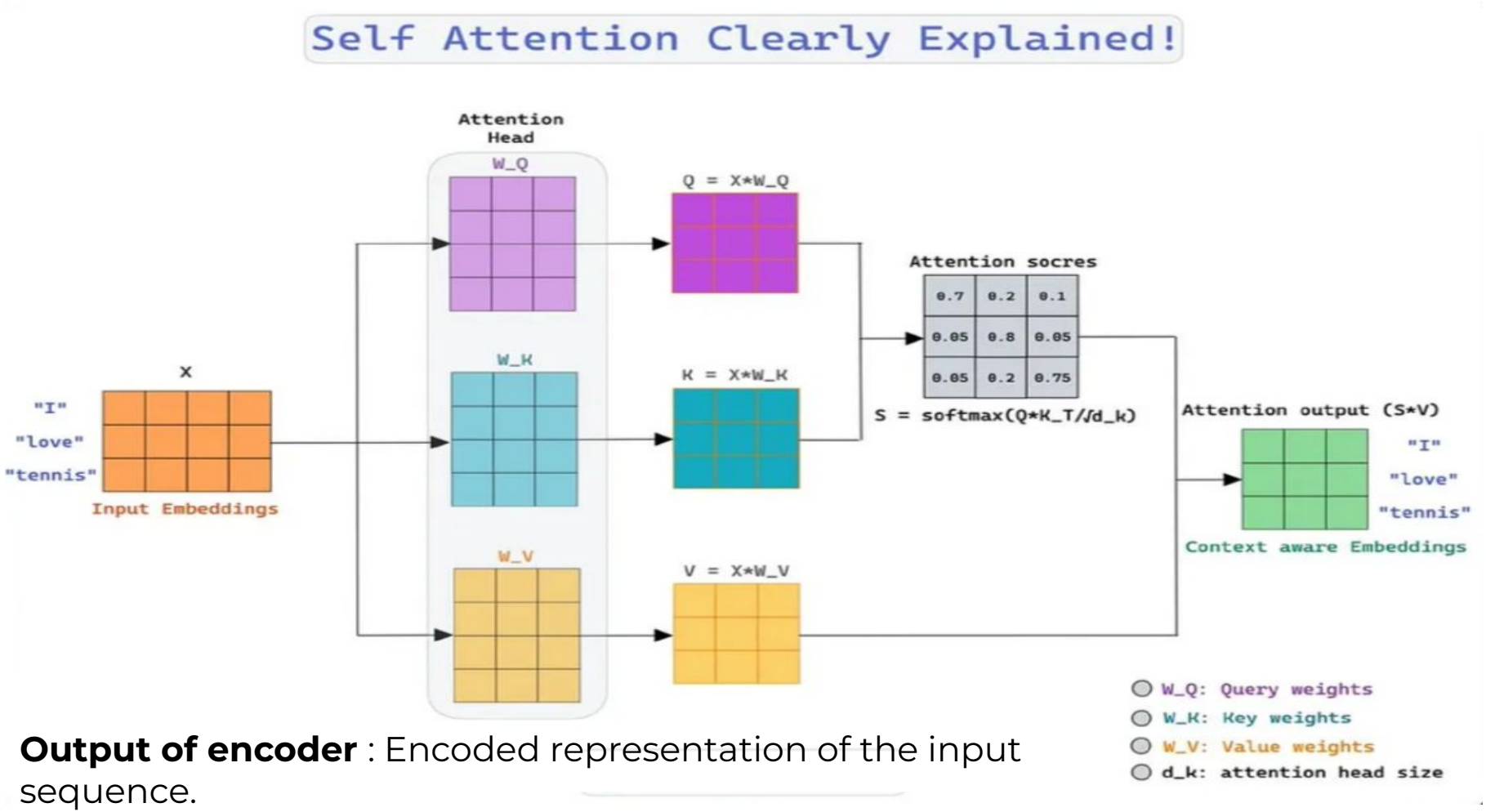
Usage:
Normalize activations in each layer independently.
Normalization helps in stabilizing and speeding up the training process by keeping activations within a certain range.

$$\mathrm{LN}(x) = \gamma \frac{x-\mu}{\sigma} + \beta$$

μ: mean, σ: standard deviation

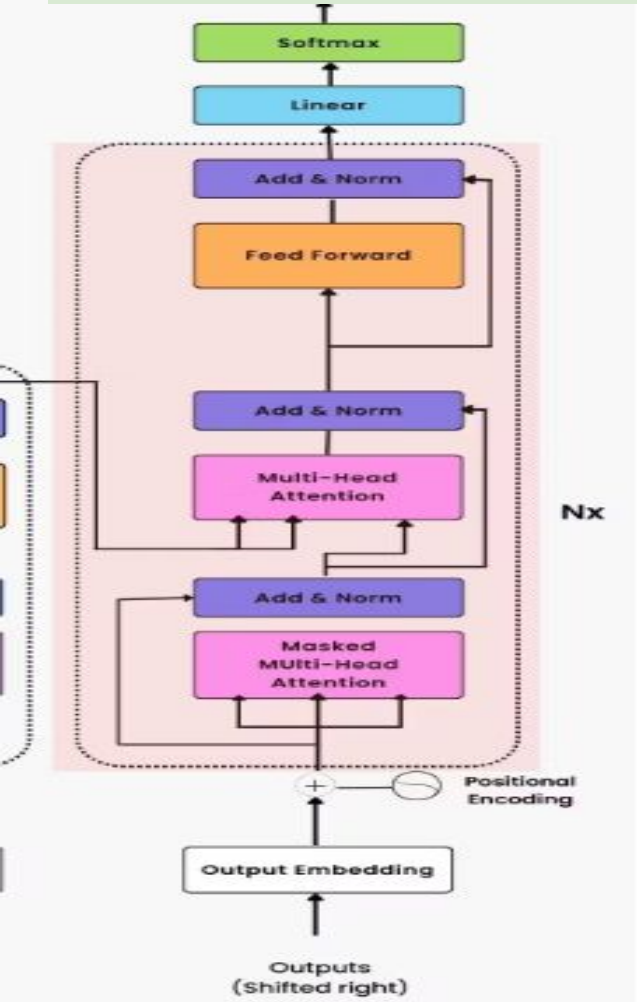
γ (gain) and β (bias) are learnable parameters.

Implications of Not Using:
Training Instability: Unstable training without normalization.
Gradient Issues: Risk of gradient explosion or vanishing.
Slower Convergence: May require careful tuning and slower convergence.
Sensitivity: Models become more sensitive to initialization.
Performance: Potential degradation, especially in deep architectures.



Output of encoder : Encoded representation of the input sequence.

Decoder architecture



- **Output Embedding + Positional Encoding**
  - Converts input tokens into embeddings and adds positional information.
- **Masked Multi-Head Attention**
  - Performs self-attention to focus on relevant parts of the input sequence, masking future tokens to prevent information leakage.
- **Add & Norm**
  - Adds the input to the output of the previous layer (residual connection) and normalizes.
- **Multi-Head Attention**
  - Attends to different parts of the input sequence, allowing the model to consider various aspects simultaneously.
- **Add & Norm**
  - Adds the input to the output of the previous layer and normalizes.
- **Feed Forward**
  - Applies a two-layer fully connected neural network to each token independently.
- **Add & Norm**
  - Adds the input to the output of the feed forward layer and normalizes.
- **Linear**
  - Applies a linear transformation to project the final token representations to the vocabulary size.
- **Softmax**
  - Converts logits into probabilities.

- **Soft max**

$$P(\text{token}) = \text{softmax}(\text{Logits})$$

⇧

- **Linear**

$$\text{Logits} = \text{Linear}(X)$$

$$\text{Output} = \text{LayerNorm}(X + \text{FFN}(X))$$

- **Add & Norm**

⇧

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

- **Feed Forward**

$$\text{Output} = \text{LayerNorm}(X + \text{MultiHead}(X))$$

- **Add & Norm**

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

- **Multi-Head Attention**

⇧

$$\text{Output} = \text{LayerNorm}(X + \text{MaskedMultiHead}(X))$$

- **Add & Norm**

⇧

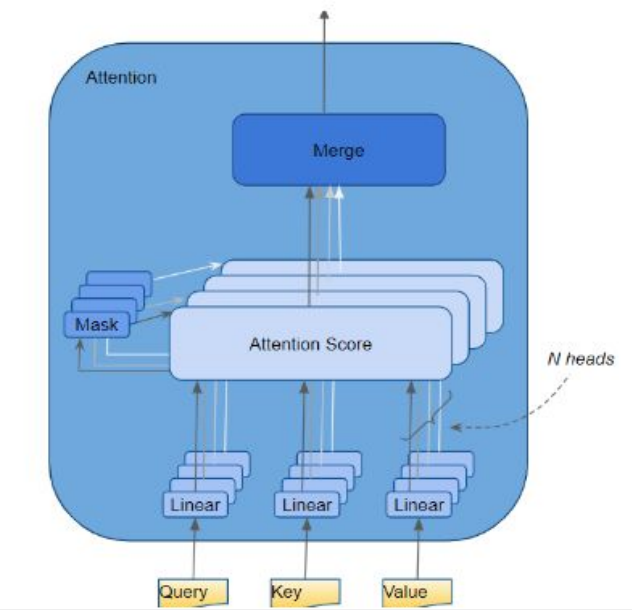
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \text{mask}\right)V$$

- **Masked Multi-Head Attention**

⇧

$$\mathbf{E}'_{\text{token}} = \mathbf{E}_{\text{token}} + PE_{\text{pos}}$$

- **Output Embedding + Positional Encoding**



Masked Multi-Head Attention

Masked Multi-Head Attention is suitable for autoregressive tasks and sequential data where future information should not be seen

When it's Used?

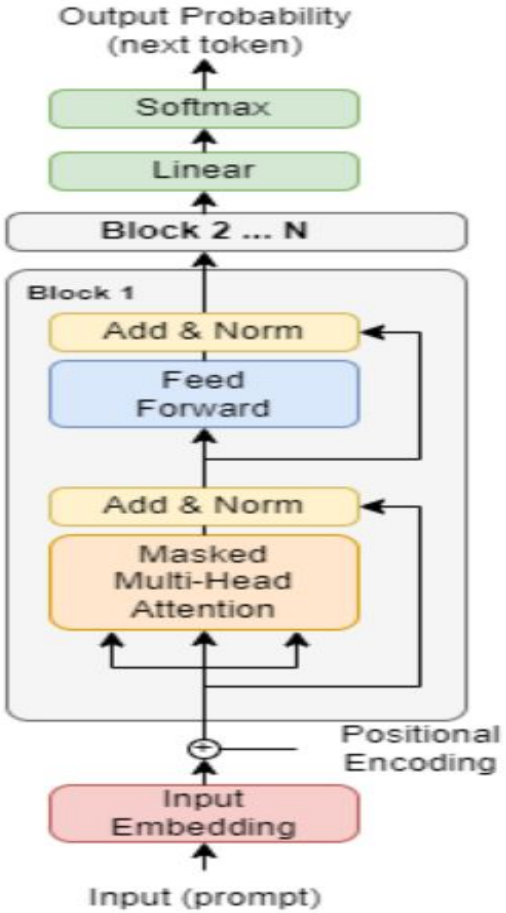
Autoregressive Tasks: Text generation, music generation. Dealing with Sequential Data-Masked multi-head attention shines in tasks where understanding the sequence of information is essential.

Sequence Labeling: Named Entity Recognition (NER), part-of-speech tagging.

Language Modeling: Tasks requiring Autoregressive Property-Language Generation Tasks-Predicting the next word in a sentence.

Use cases where it is used in Valeo :

Chatbots leverage masked multi-head attention to hold engaging conversations by considering past interactions.predicting the next question based on the conversation history.



DECODER ONLY ARCHITECTURE

Decoder-Only Transformer (Autoregressive Models):

- **Architecture:** This utilizes just the decoder for tasks where the output sequence is generated one token at a time, considering previously generated tokens.
- **Use Cases:** Text generation, chatbots, music composition.
- **Learning Approach:** Can be supervised or unsupervised. Supervised learning involves training on labeled data with the desired output sequence. In unsupervised settings, the model learns from a large corpus to predict the next likely token in a sequence.

When and Why Decoder-Only Architecture is Used:

- **Autoregressive Generation:** Suitable for tasks where each output depends on previously generated outputs.
- **Efficiency in Text Generation:** Computationally efficient for generating text word by word.
- **Fine-Grained Control:** Provides precise text manipulation.
- **Document-Level Tasks:** Suitable for tasks based on entire document context.
- **Handling Variable-Length Sequences:** Efficiently handles variable-length input and output.
- **Avoiding Encoder Overhead:** Eliminates the need for encoding the entire input sequence.

Benefits:

- **Fine-Grained Training:** Model learns to generate text word by word.
- **Efficiency:** Computationally efficient for autoregressive tasks.

Challenges:

- **Data Efficiency:** Requires large amounts of training data.
- **Prompt Design:** Important to design effective prompts/questions.

Thank You....