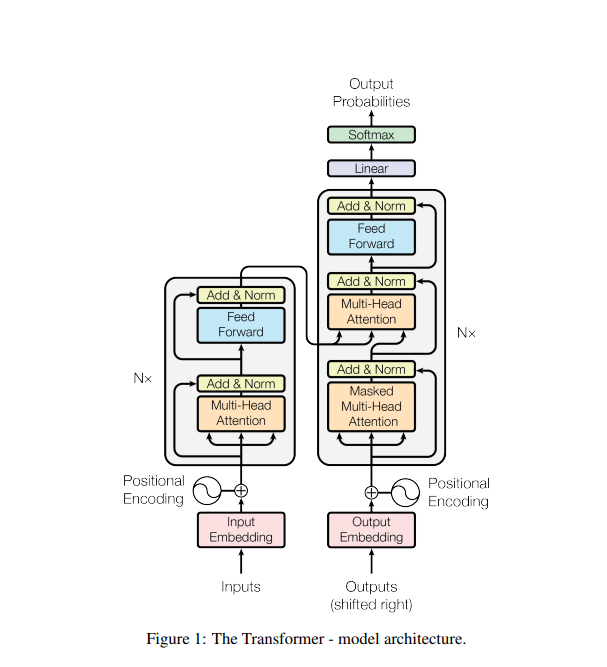
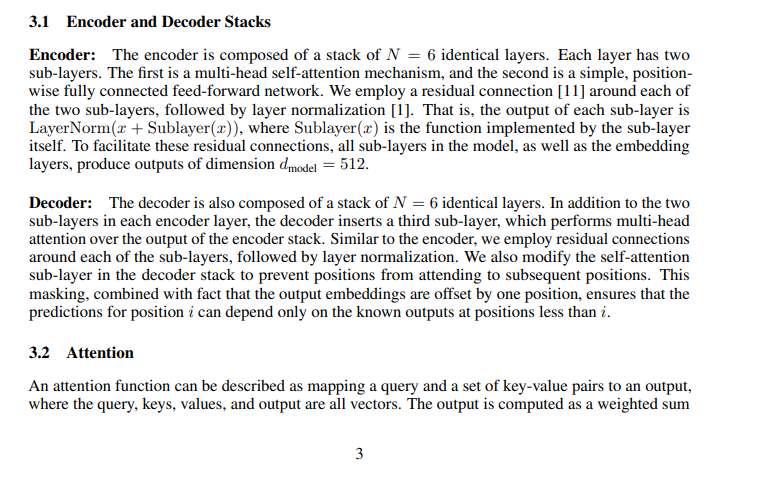
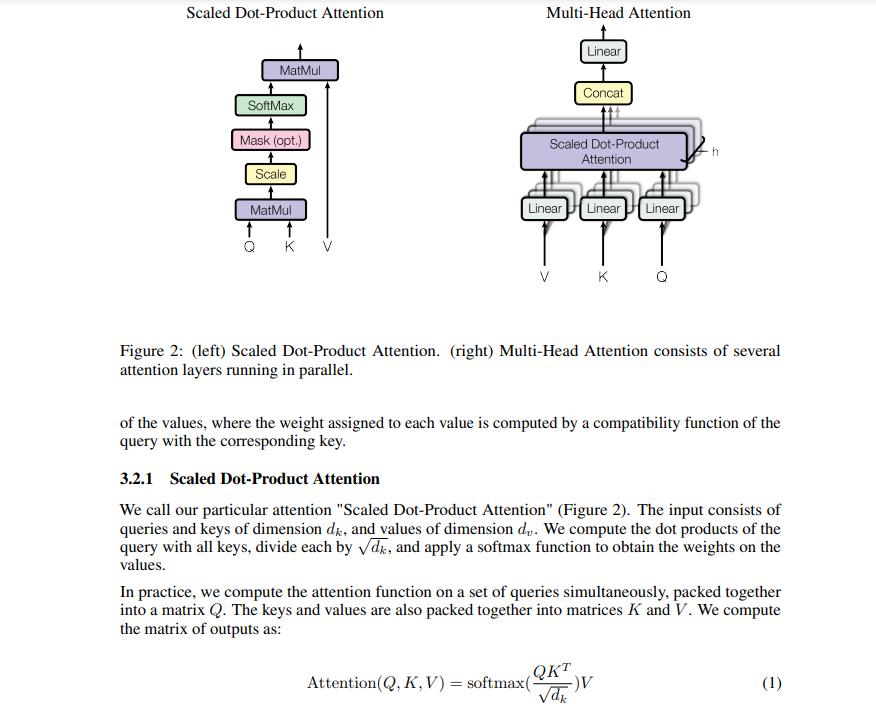
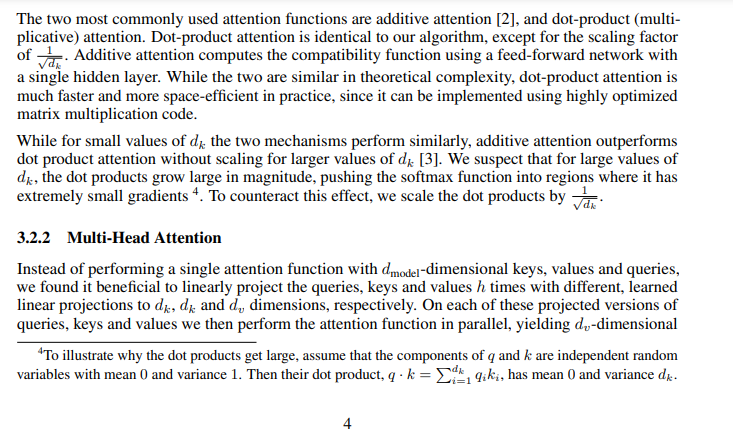
**The base architecture for large language models (LLMs) is typically the Transformer architecture, introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017. This architecture revolutionized natural language processing (NLP) by focusing on the self-attention mechanism, enabling better handling of long-range dependencies in text compared to older models like RNNs and LSTMs.**

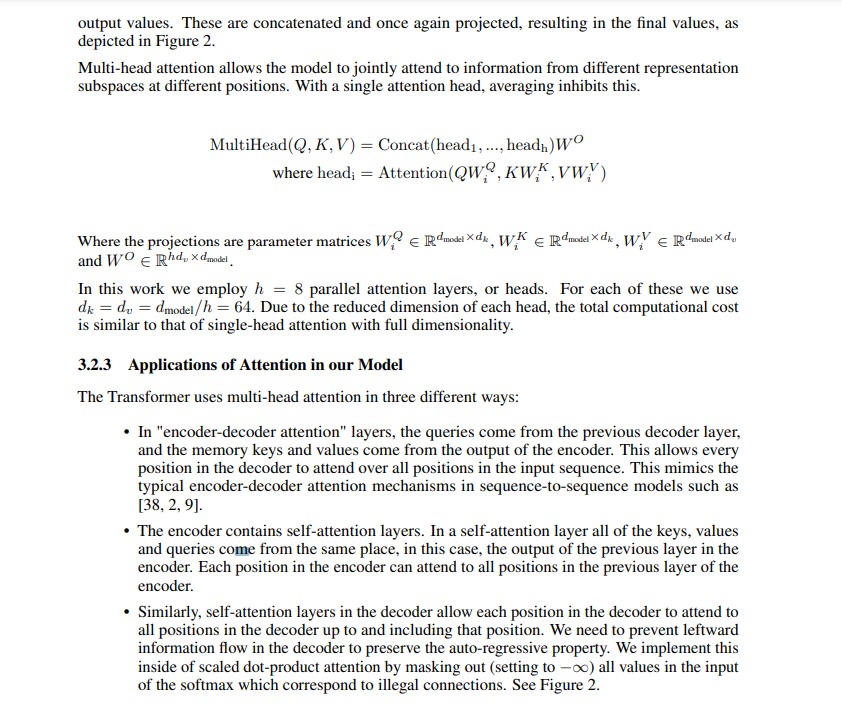
**Here are some screen-shots from paper regarding crucial information along with diagram of the base transformer architecture. :-**

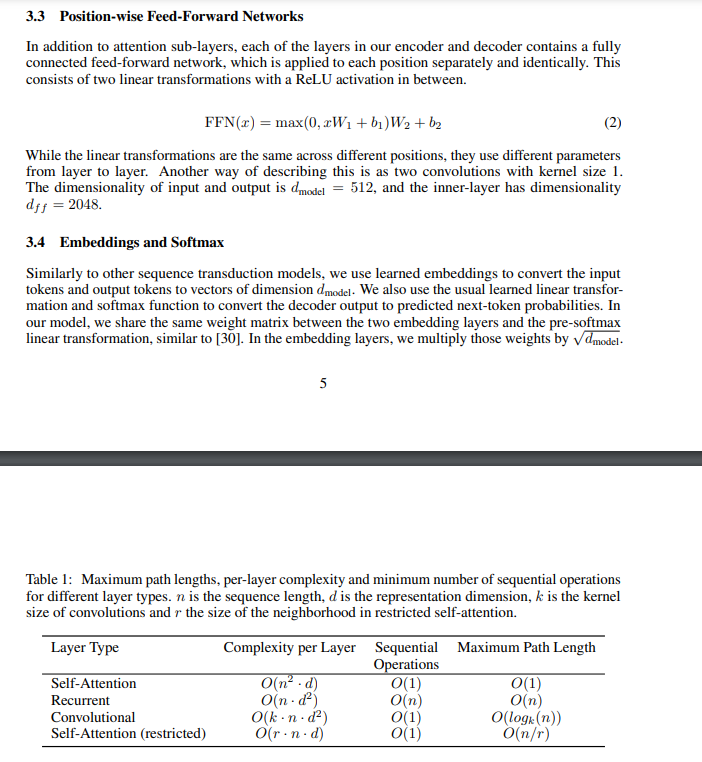
****

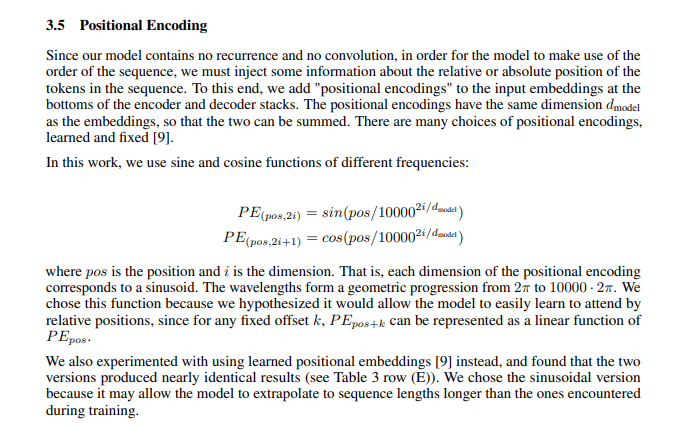
****

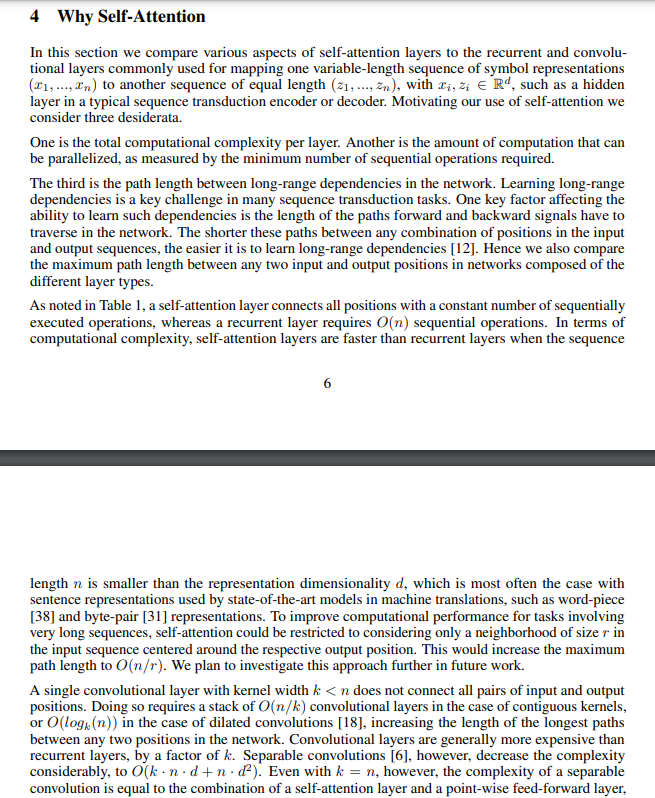
****

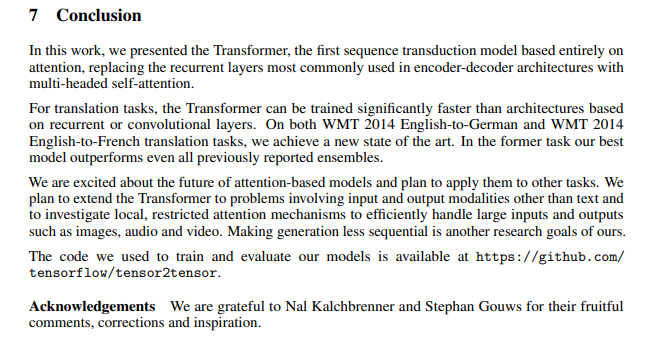
****

****

****

****

****

****

Here’s a detailed breakdown of all key components of the Transformer architecture:

### 1. ****Input Representation (Tokenization and Embeddings)****:

* **Tokenization**: Before feeding data into the model, text is tokenized into subword units or tokens. Tokenization methods like **Byte Pair Encoding (BPE)** or **WordPiece** are often used, which break words down into smaller units that make learning more flexible.
* **Positional Encoding**: Since the Transformer doesn’t have any built-in notion of the order of the tokens (unlike RNNs or CNNs), it uses **positional encodings** to inject information about the position of each token. These encodings are added to the input embeddings and help the model understand the relative positions of tokens in the sequence.
* **Word Embeddings**: Each token is converted into a vector representation (embedding) via a learned embedding layer.

### 2. ****Self-Attention Mechanism****:

The **self-attention** mechanism is at the core of the Transformer architecture. It allows the model to look at other tokens in the sequence and determine which tokens are important for each position.

**Key Components of Self-Attention**:

* **Query (Q), Key (K), and Value (V)**: For each token, the model computes three vectors: a Query vector, a Key vector, and a Value vector. These are linear transformations of the token’s embedding.
  + **Query**: What the model is trying to understand or match for this token.
  + **Key**: The information that might match the Query in other tokens.
  + **Value**: The actual information of the token.
* **Attention Score**: The dot product of the Query and Key vectors is computed to determine the "attention score." These scores reflect the importance of each token with respect to others. The scores are then normalized using **softmax**, so they sum up to 1.
* **Weighted Sum of Values**: The normalized attention scores are used to weight the Value vectors, and the weighted sum produces the attention output. This output is a weighted combination of all the token embeddings in the input sequence.

**Multi-Headed Self-Attention**: Instead of performing self-attention once, Transformers use multiple attention “heads” (called **multi-head attention**), where each head learns to focus on different parts of the sequence. These attention heads are run in parallel and their outputs are concatenated together.

### 3. ****Feed-Forward Neural Networks (FFNN)****:

After the self-attention mechanism, the output goes through a **position-wise feed-forward neural network**. This is a fully connected network that applies transformations to each position independently.

This FFNN consists of two linear layers with a ReLU activation in between:

* + FFNN(x)=max⁡(0,xW1+b1)W2+b2\text{FFNN}(x) = \max(0, xW\_1 + b\_1)W\_2 + b\_2FFNN(x)=max(0,xW1​+b1​)W2​+b2​

The purpose of this is to allow the model to learn more complex transformations of the data after the self-attention mechanism has aggregated contextual information.

### 4. ****Normalization and Residual Connections****:

* **Layer Normalization**: Transformers use **layer normalization** after the attention and feed-forward layers. This helps stabilize training by normalizing the activations across the feature dimensions for each token.
* **Residual Connections**: To improve the flow of gradients during backpropagation and make training deeper models easier, Transformers use **residual (skip) connections**. These connections add the original input of a layer to its output:
  + Output=LayerNorm(x+SubLayer(x))\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))Output=LayerNorm(x+SubLayer(x))

### 5. ****Encoder-Decoder Structure (for Sequence-to-Sequence Tasks)****:

The original Transformer architecture is designed for sequence-to-sequence tasks (like translation) and consists of two main components:

**Encoder**:

* Takes the input sequence and produces a hidden representation.
* Contains **multiple layers** (often 6 or 12) of multi-head self-attention and feed-forward networks.
* The output of each layer is passed to the next, and the final output is used by the decoder.

**Decoder**:

* Generates the output sequence, one token at a time.
* It has a similar architecture to the encoder but with an additional mechanism for **masked self-attention**. Masking ensures that the decoder can’t “see” future tokens when generating a sequence.
* It also includes a second multi-head attention mechanism that attends over the encoder's output, allowing the decoder to focus on the relevant parts of the input.

### 6. ****Attention Masking****:

Masking is used in different parts of the Transformer:

* **Padding Mask**: Prevents the model from attending to padding tokens (e.g., when sequences are padded to the same length).
* **Look-Ahead Mask (Causal Mask)**: Ensures that during training, the model doesn't look ahead to future tokens when predicting the next word in the sequence.

### 7. ****Output Layer****:

For language generation tasks, the output of the decoder or the final encoder layer is passed through a linear layer followed by a **softmax** to generate a probability distribution over the vocabulary for the next token.

* **Language Modeling**: The model generates text one token at a time by predicting the next token in a sequence.

### 8. ****Training and Loss Function****:

* Transformers are trained using **cross-entropy loss**, where the model’s predicted probability distribution is compared to the ground truth labels (next token in the sequence, for example).
* **Optimization**: The model is typically trained using **Adam optimizer** with techniques like **learning rate scheduling** (e.g., warm-up followed by a decay) to stabilize training.

### Transformer Architecture Summary:

| **Component** | **Description** |
| --- | --- |
| **Input Representation** | Tokens, embeddings, and positional encodings. |
| **Self-Attention** | Scales input features by attention weights across tokens. |
| **Multi-Head Attention** | Multiple parallel attention layers learning different relationships. |
| **Feed-Forward Neural Network** | A position-wise FFNN for non-linear transformations. |
| **Normalization** | Layer normalization to stabilize training. |
| **Residual Connections** | Skip connections to ease the flow of gradients. |
| **Encoder-Decoder (for seq2seq)** | Encodes the input and decodes the output. |
| **Masked Attention** | Prevents peeking at future tokens during training. |
| **Output Layer** | Generates token probabilities through softmax. |
| **Loss Function** | Typically cross-entropy loss for training. |

### Applications:

* **Text generation** (e.g., GPT, GPT-3/4).
* **Machine translation** (e.g., BERT, T5).
* **Question answering**.
* **Text summarization**.

### Transformer Derivatives:

The base Transformer architecture has been adapted for various models:

* **GPT** (Generative Pre-trained Transformer): Focuses on autoregressive language modeling (decoding only).
* **BERT** (Bidirectional Encoder Representations from Transformers): Focuses on masked language modeling (encoding only).
* **T5** (Text-to-Text Transfer Transformer): A unified text-to-text framework for multiple tasks.

These variants use the same core Transformer building blocks but modify them for specific tasks or training objectives.