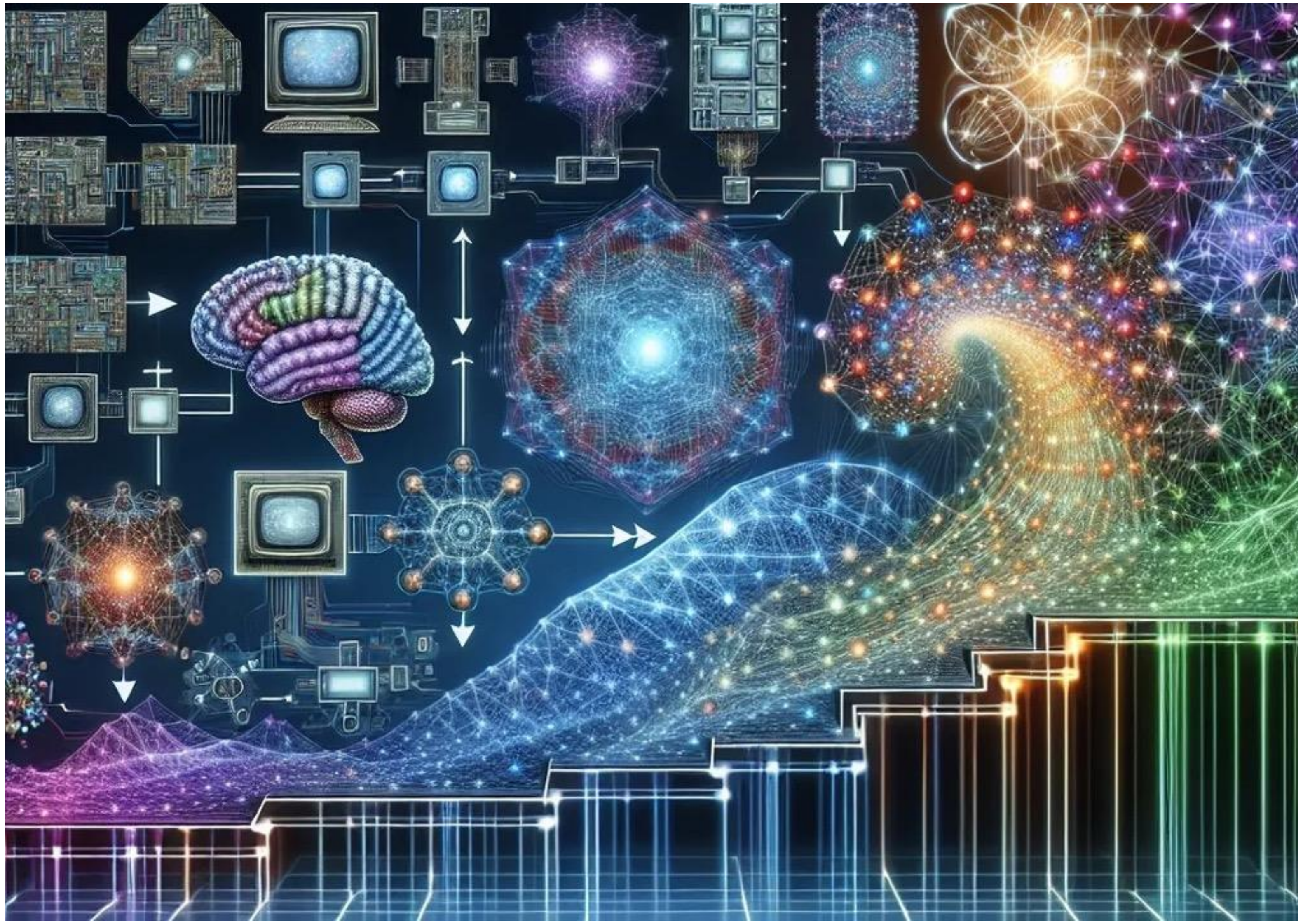


From Recurrent Neural Networks (RNNs) to Generative AI

How Recurrent Networks Led to Transformers, LLMs, and Modern Generative AI

Introduction

The journey from traditional RNNs and LSTMs to cutting-edge models like Transformers and Generative AI

- **RNN** (Recurrent Neural Networks) for sequential data.
  - It is suited for handling data where the current input depends not only on the current features but also on previous inputs in the sequence. This makes them well-suited for tasks like time series forecasting, speech recognition, language modeling, and more.
- **LSTM** (Long Short-Term Memory) for better memory retention.
  - Instead of having a single hidden state like in RNN, an LSTM has a more complex structure with **multiple gates** that control the flow of information through the network. These gates help the LSTM decide which information should be retained, which should be discarded, and which should be updated.
- **Transformers** revolutionizing attention mechanisms and sequence processing.
  - Replace the recurrent structure of LSTMs with **self-attention**, allowing for **parallel processing** and better handling of long-range dependencies.
- **Self-Attention** (**Scaled Dot-Product Attention)** and its role in scalability.
  - **Capture long-range dependencies** between tokens.
  - **Parallelize** computations (since attention is applied globally rather than sequentially)
  - Focus on **relevant context** at each position in the sequence. **Dynamic contextualization**, where each token can adjust its representation based on its most relevant context.
- **Large Language Models (LLMs)** and their generative capabilities.
  - Neural network model trained on a massive corpus of text data
  - Text generation (e.g., writing essays, stories, or code), Language translation, Question answering, Summarization, Sentiment analysis,

Recurrent Neural Networks (RNNs)

Problem: Sequential data (time series, language) requires models that retain context over time.

Architecture:

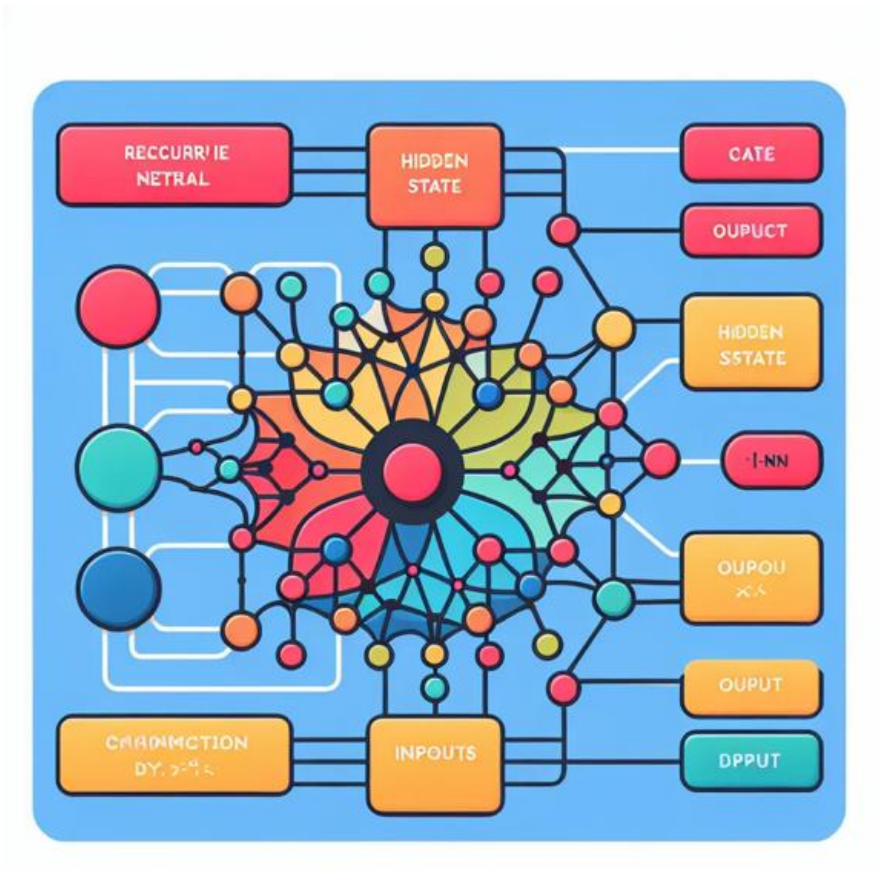
- RNNs process data sequentially (step-by-step).
- Hidden states carry information through time, making them suitable for tasks like language modeling and speech recognition.

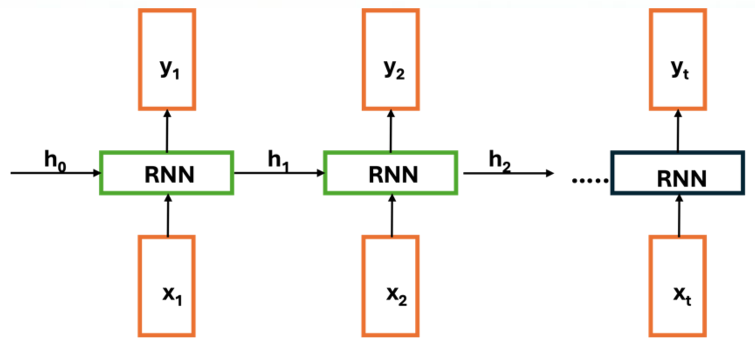
Technical Detail:

- **Mathematics:**
  - Update rule: $h_t = \sigma(Wx_t + Uh_{t-1} + b)$; where h_t is the hidden state and h_{t-1} is the hidden state from the previous step, W and U are the weight matrices and b is the bias and σ is the activation function.
  - Challenges: Difficulty with **vanishing gradients** (long-term dependencies not well learned).

Drawback:

- **Vanishing/Exploding Gradients** during backpropagation, making it hard for RNNs to learn long-range dependencies.





Long Short-Term Memory (LSTM)

Problem:

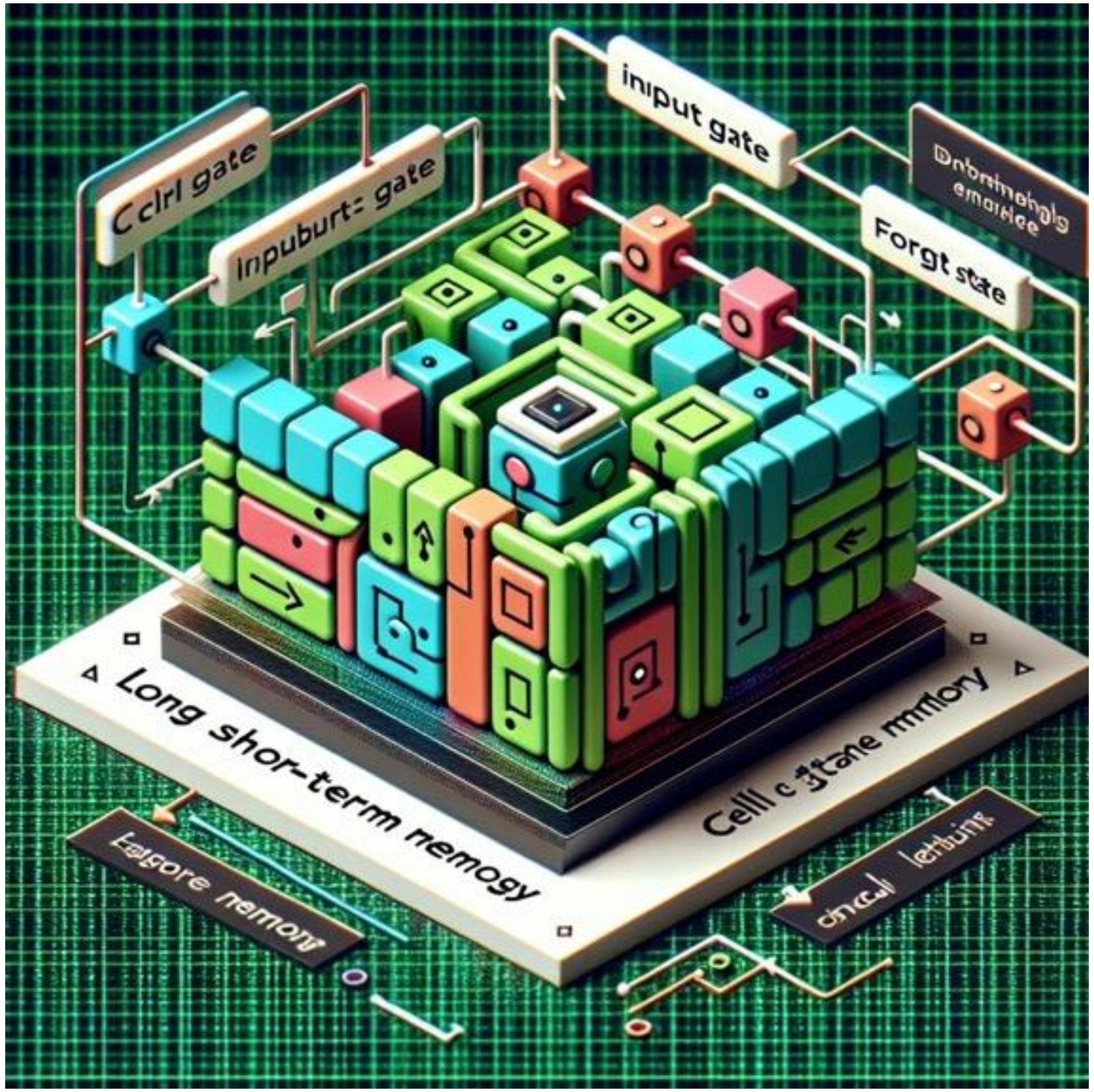
- RNNs fail to learn long-term dependencies effectively.

Solution:

- LSTMs introduce **gates** to manage the flow of information.
- **Forget gate**: Decides what information to discard.
- **Input gate**: Decides what new information to store.
- **Output gate**: Controls what information to output.

Benefit:

- LSTMs help mitigate the vanishing gradient problem by maintaining long-term memory.



Challenges with RNNs and LSTMs

- Even with LSTM improvements,
- **training speed** and **parallelization** remain issues.
- RNNs and LSTMs are still **sequential** models and cannot fully leverage parallel hardware (GPUs, TPUs).
- **Global dependencies** in data may still be challenging to capture efficiently.

Introduction to Transformers

Problem: Sequential nature of RNNs limits efficiency in handling long sequences.

Solution: Transformers introduce **self-attention**, allowing models to process entire sequences at once regardless of their distance in the sequence, addressing the vanishing gradient problem faced by RNNs

Key Concept:

- **Self-Attention Mechanism**: A model can attend to all parts of the input sequence simultaneously, not just previous steps. Allow the model to relate words to each other.

Transformers Overview:

- Transformer architecture: Encoder-Decoder.
- Main building block: **Self-Attention** and **Position Encoding** (close or distant words).
- Attention Score:
  Attention(Q,K,V)= softmax($\frac{QK^T}{\sqrt{d_k}}$)V

Where Q is Query, K is the key and V is the value

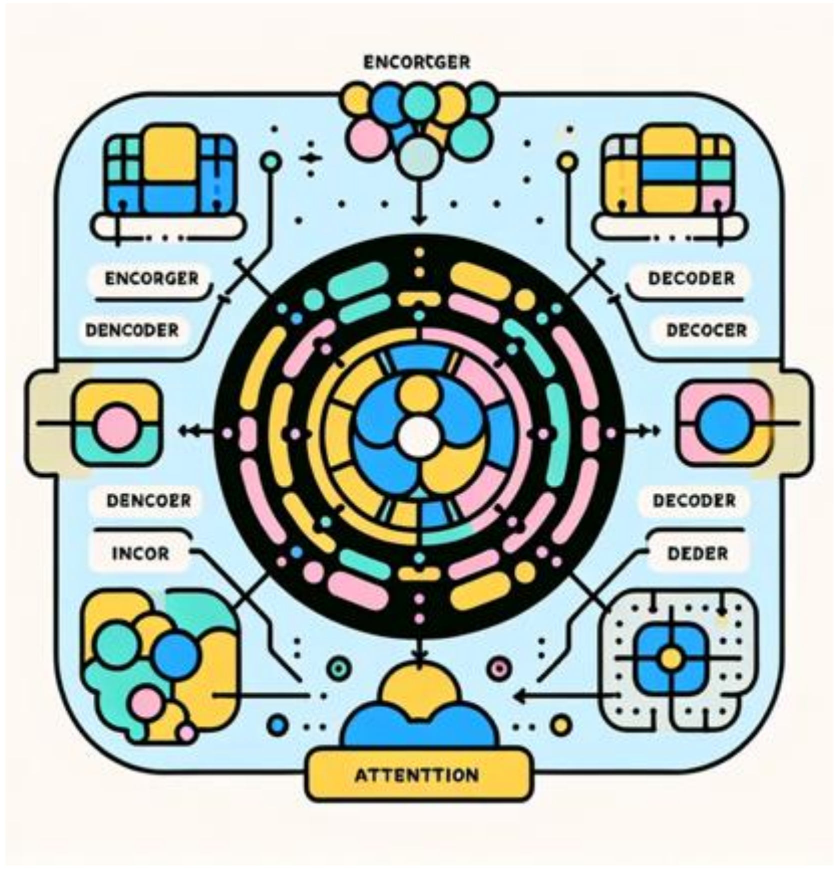
Query vector: Represents the word we're focusing on

Key vector: Helps in matching with other words

Value vector: Carries the actual content of the word

d_k is size of the embedding vector / number of heads

softmax will make the sum of all entries in each row to be equal to 1 for the $\frac{QK^T}{\sqrt{d_k}}$.



Self-Attention Mechanism

Self−Attention enables the model to weigh the importance of each token in the input sequence relative to others.

- It computes attention scores for each token pair and uses them to derive context−sensitive representations.
- Scaled Dot-Product Attention:

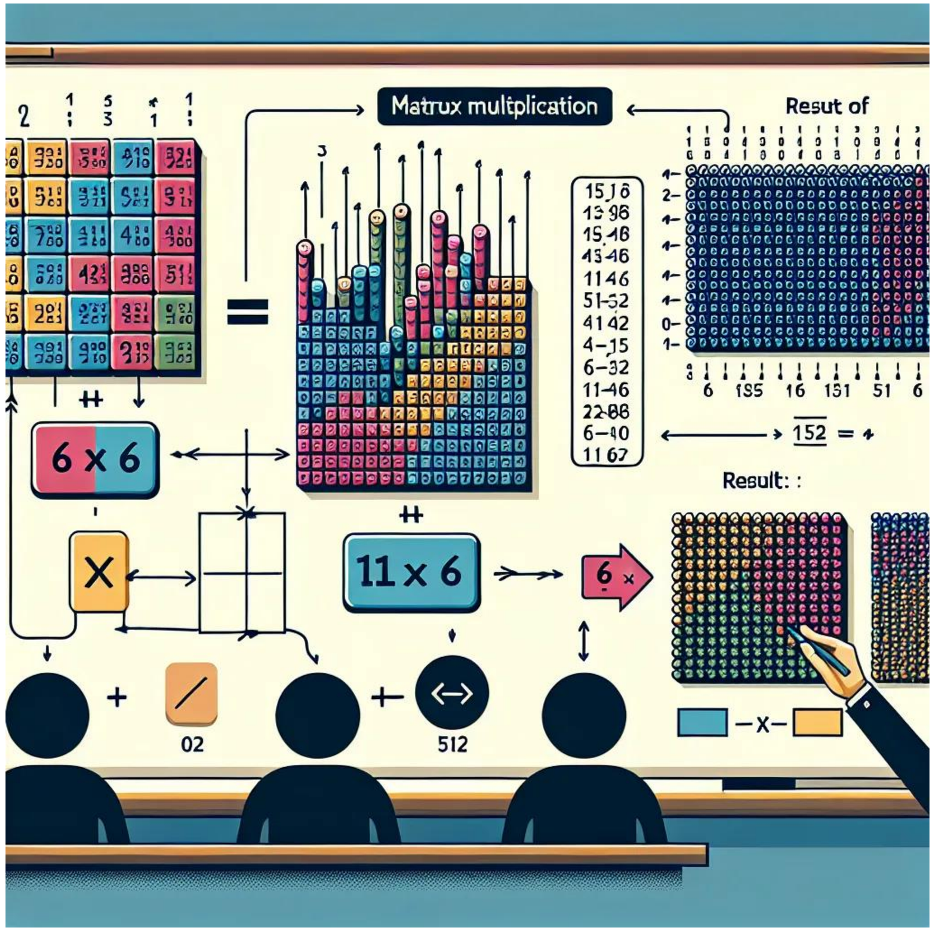
$$\text{Attention(Q,K,V)= softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\text{V}$$

Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence represented by the positional encoding but also each word's interaction with other words.

Self-Attention is permutation invariant

The product matrices are diagonally dominant

Multi-Head Attention:
Instead of using a single attention mechanism, use multiple heads to capture different aspects of relationships between tokens. One word maybe noun, verb or adverb depending on the context, multi-head attention will catch that context meaning.



Benefits:

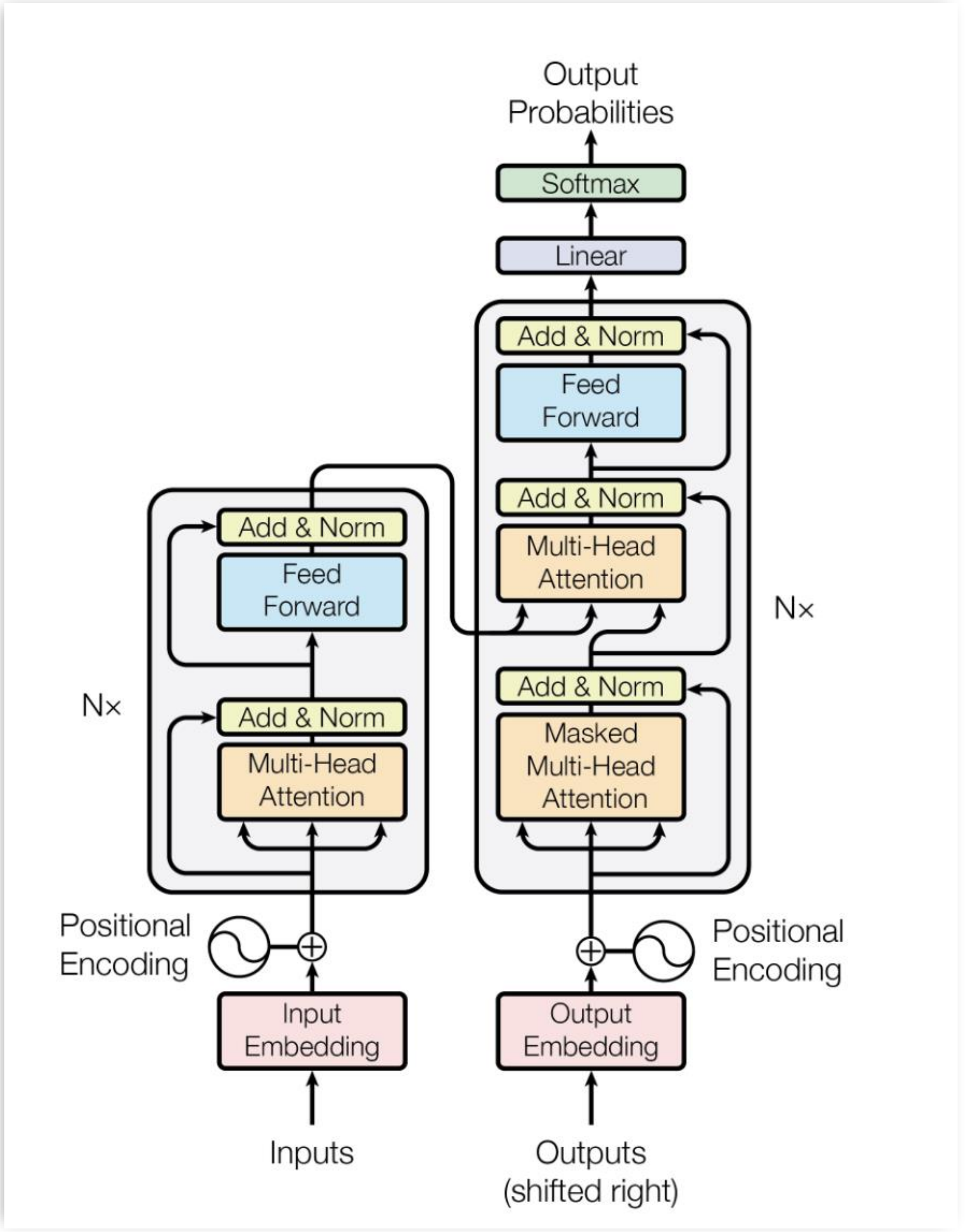
- **Parallelization**: Attention mechanisms are parallelizable, improving training efficiency.
- **Global Context**: Each token can attend to all other tokens in the sequence.

$$\begin{bmatrix} Q_{1,1} & Q_{1,2} & \cdots & Q_{1,512} \\ Q_{2,1} & Q_{2,2} & \cdots & Q_{2,512} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{6,1} & Q_{6,2} & \cdots & Q_{6,512} \end{bmatrix} \times \begin{bmatrix} K_{1,1} & K_{1,2} & \cdots & K_{1,6} \\ K_{2,1} & K_{2,2} & \cdots & K_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ K_{512,1} & K_{512,2} & \cdots & K_{512,6} \end{bmatrix} = \begin{bmatrix} QK_{1,1} & QK_{1,2} & \cdots & QK_{1,6} \\ QK_{2,1} & QK_{2,2} & \cdots & QK_{2,6} \\ \vdots & \vdots & \ddots & \vdots \\ QK_{6,1} & QK_{6,2} & \cdots & QK_{6,6} \end{bmatrix}$$

Transformer Architecture

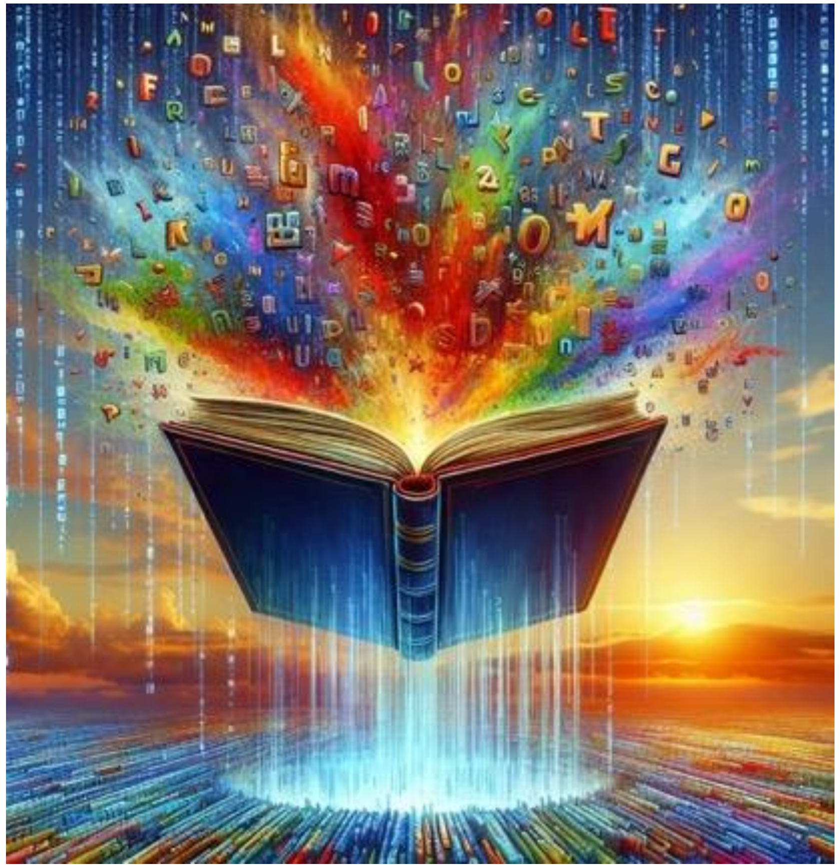
- **Encoder-Decoder Structure:**
- **Encoder**: Processes input sequence (e.g., text) and produces context-aware embeddings.
- **Decoder**: Generates output sequence (e.g., translation).
- **Multi-Head Self-Attention:** Captures relationships within input tokens.
- **Feed-Forward Networks:** Fully connected layers applied to each token.
- **Positional Encoding:** Since Transformers don't process data sequentially, positional encodings are added to retain order.
- **Overview of Steps:**

1. **Input Sequence**: A sequence of tokens (e.g., words in a sentence).
2. **Queries (Q), Keys (K), Values (V)**: Each token is represented as a set of Q, K, and V vectors.
3. **Attention Scores**: Calculate the relevance of one token to others by computing dot products of Q and K. ($\mathbf{Q_i} \bullet \mathbf{K_j}$)
4. **Softmax Normalization**: Normalize scores to ensure they sum to 1. softmax ($\mathbf{Q_i} \bullet \mathbf{K_j}$)
5. **Weighted Sum**: Compute the output by combining values based on the attention weights. softmax$(\frac{QK^T}{\sqrt{d_k}})$V



Rise of Large Language Models (LLMs)

- Transformers form the backbone of **Large Language Models (LLMs)** like GPT, BERT, T5, etc.These models are trained on massive amounts of text and fine-tuned for specific tasks.
- **Pretraining** on vast datasets to learn general language patterns.
- **Fine-tuning** on smaller, task-specific datasets (e.g., sentiment analysis, translation).
- **Transfer Learning**: LLMs can transfer knowledge to multiple downstream tasks.



Generative AI and Transformers

- **Generative AI**: Transformers have become the foundation for generative tasks like text generation, image synthesis, and even code generation.
  - **GPT (Generative Pretrained Transformer)**:Trained to predict the next word/token in a sequence.
  - **Autoregressive**: The model generates tokens one by one, conditioned on previously generated tokens.
- **Zero-shot learning**: LLMs can perform tasks without explicit task-specific training.
- **Prompt Engineering**: In generative AI, the quality of the output depends on how the prompt is structured
- **Scaling up** model size and dataset leads to significant improvements in model performance
- **GPT-3** and beyond show the power of large-scale models with billions of parameters.
- Compute costs, environmental concerns of large models.

Current Challenges and Future Directions

- **Challenges with Transformers**:
- **Memory consumption**: Attention scales quadratically with sequence length.
- **self-attention mechanism**, grows as O(n^2) where n refers to the length of the input sequence (i.e., the number of tokens or elements in the sequence).
- **attention score matrix** of size $n \times n$, where each element in the matrix corresponds to the attention score between two tokens.
- The total memory consumption is:
  - **Input embedding matrix**: $O(n \times d)$ where d is the dimensionality of the token embeddings
  - **Attention score matrix**: O(n^2)
  - **Output of attention**: $O(n \times d_v)$ where d_v is the dimensionality of the value vectors
- **Multimodal models**: Vision and language models like CLIP and DALL·E.