ATTENTION IS ALL YOU NEED

TRANSFORMERS

Large Language Models, Their Capabilities and Limitations

REALM

2025-02-15

TRANSFORMERS, GPT, AND EVEN MORE

Is mass unemployment looming?

Unsupervised AI?

AI Spy?

Automation of all routine work?

Will he write a PhD for me?

Will it replace all other AIs?

AND HOW DO THESE TRANSFORMERS WORK?

REALM

AI – REVOLUTION AND SUPERIORITY OF TRANSFORMERS

> Transformers are a powerful new way of representing natural language that can capture long-range dependencies and complex relationships. They have enabled us to make significant advances in natural language understanding, generation, and translation
>
> Sundar Pichai, CEO of Google

- **AI Explosive Growth**: Recently, AI has become a deafening trend, transforming industries and enriching our day-to-day experiences. From image recognition technologies to autonomous driving, AI has established itself as a key driver of innovation.

- **New horizons of AI application**: Every day brings news of breakthroughs in AI, expanding its applications from finding new drugs to analyzing composite materials. Systems like ChatGPT demonstrate the ability to not only create essays and program code, but also to conduct intelligent dialogues to help make sense of complex issues.

- **Rethinking NLP and Growing Faith in Technology**: Admiration for AI's achievements leads to a new kind of faith in the technology. Startups are looking to replicate the success of OpenAI by reimagining the use of NLP in everyday life. Investors, having forgotten about blockchain and META universes, are now looking for revolutionary opportunities in AI.

- **Purpose** : Explain the current status of transformers technology, understand their real capabilities and limitations. To present to a wide range of specialists and users how AI and transformers can be used in various industries, emphasizing their impact on the future of the industry and society.

WHAT ARE TRANSFORMERS?

Transformers in the context of Natural Language Processing (NLP) are a type of deep learning model architecture in AI specifically designed to work with data sequences such as text

The name "transformers" comes from the key mechanism behind these models, the "**attention**" mechanism, which allows the model to "transform" input data (e.g., text) into a higher-level internal representation.

Transformers are often associated with **Large Language Models** (LLMs), such as GPT and BERT, which often use the architecture of Transformers, but are scaled to a very large number of parameters, allowing them to better understand and generate natural language. Feature of Transformers:

- **Sequence Processing**: Transformers are able to process words (or other elements) in a sentence not sequentially but in parallel, making them very efficient and fast at learning and predicting.
- **Attention Mechanism**: The main component of transformers is the attention mechanism, which allows the model to focus on different parts of the input data depending on the task. For example, when translating a sentence, the model can synchronously take into account the context of the entire sentence rather than processing each word individually.
  This helps the model understand the context and semantic relationships between words in the text.
- **Scalability**: Transformers scale well due to their ability to handle large amounts of data and the complexity of models. They can be trained on huge corpora of text and are capable of generating, classifying, translating, and performing a variety of other tasks at a high level.

TRANSFORMERS EXAMPLES

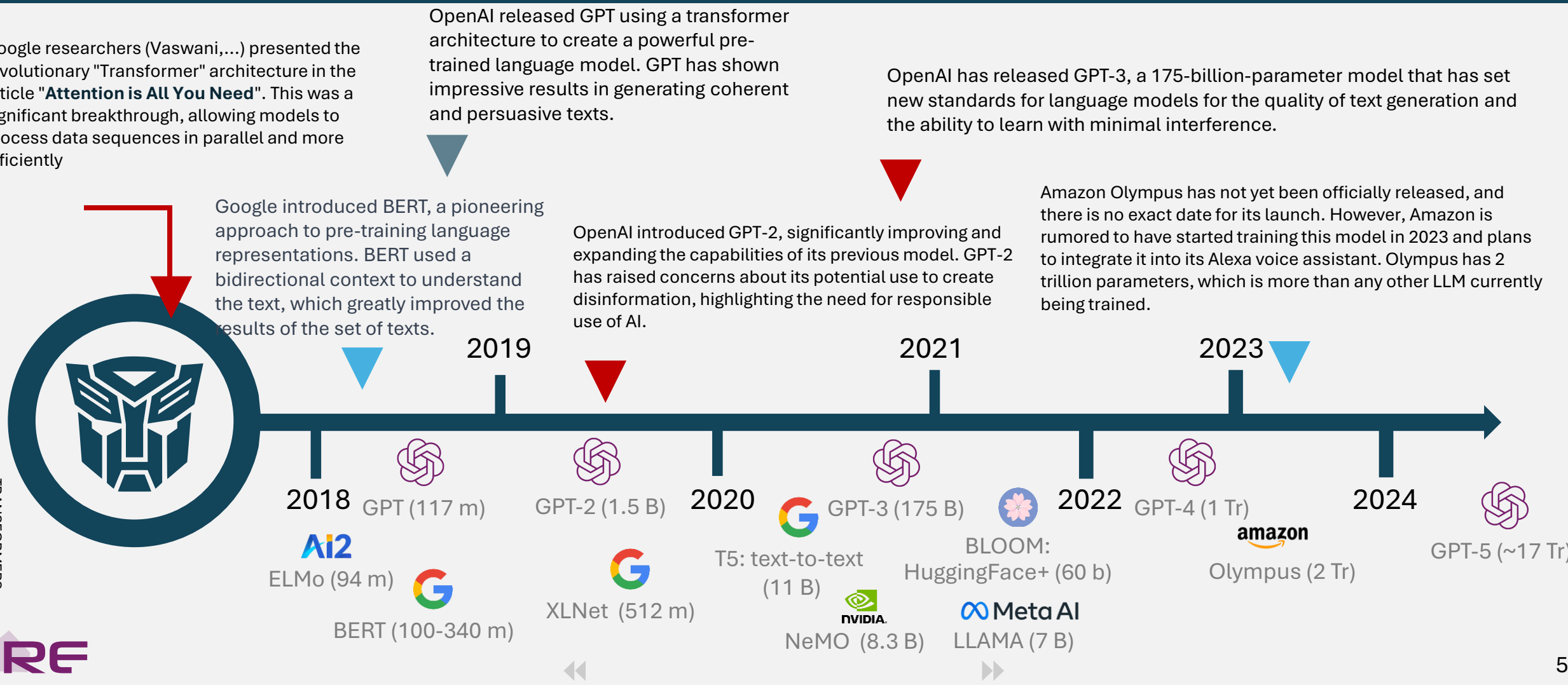
BERT (Bidirectional Encoder Representations from Transformers): Understanding the context and relationships in the text, used for text classification, sentiment analysis, information extraction, and answering questions.

GPT (Generative Pre-trained Transformer): a series of models (GPT-2, GPT-3) that are capable of generating text, answering questions, translating texts, and more.

LLM DEVELOPMENT TIMELINE

2025-02-15

As the market evolves, competition increases, the scale of models grows and computational costs increase at the same time, despite significant progress in the development of LLMs, there are relatively few publicly available services based on them, the sector is still at the stage of searching for effective and sustainable business models



TRANSFORMERS AND OTHER ARCHITECTURES

Transformers are an architecture of neural networks focused on working with sequential data – text strings. However, this architecture differs from the RNN (Recurrent Neural Networks) and LSTM (Long Short-Term Memory) architectures, which are similar in terms of the processed data

TRANSFORMERS

- **Parallel Processing**: Transformers can process all words (or other elements of a sequence) at the same time, so they can quickly work with large amounts of data.
- **The Mechanism of Attention**: Transformers use an attentional mechanism to determine the relationships between all the words in a sentence, which allows the model to better grasp the context and meaning of the text.
- **No feedback loops**: Transformers don't have feedback loops like RNNs and LSTMs, making them less prone to disappearing and exploding gradient issues and simplifying the learning process.

RNN & LTSM

- **Sequential processing**: RNNs process data sequentially, passing information from one step to the next, which can slow down the processing of large amounts of data.
- **Feedback loops**: RNNs have feedback loops, which allows them to retain information about previous steps, but also makes them vulnerable to problems with fading and exploding gradients.
- **LSTM - Improved Version of RNN**: LSTM is a special type of RNN designed to solve the problem of vanishing gradient by using structures called "memory cells".

  These cells allow the model to retain information for long periods of time and learn more effectively from data with long dependencies

STANDARD ATTENTION MECHANISM

2025-02-15

The attention mechanism in transformers is a key element of their architecture, allows the model to focus on certain parts of the input data (for example, words in a sentence) to perform a specific task, provides weighting of the importance of a word, creating combinations, and parallel text processing

The central components of the attention mechanism in the Transformer architecture* are the vectors Q (Query), K (Key), and V (Value). They are used so that the model can determine which parts of the input data to pay attention to when performing the task:

1. VECTOR SELECTION

- Each word (or other unit of data) in the input sequence is transformed into three vectors q, k, v, which are packaged into aggregate matrices Q, K, and V. These transformations are performed using three different sets of trained weights.
- The Query vector (Q) is associated with the current word for which attention is calculated.
- The Key (K) and Value (V) vectors are associated with all words in the input sequence (including the current word).

$$\{Q, K, V\}$$

2. ATTENTION WEIGHTS

- For each pair of Query and Key, a measure of similarity is determined, computed through the dot product between Q and K.
- The resulting similarity values are then scaled and normalized using the softmax function to get the attention weights, which are summed up in 1.

$$e_{\mathbf{q},\mathbf{k}_i} = \mathbf{q} \cdot \mathbf{k}_i$$

$$\alpha_{\mathbf{q},\mathbf{k}_i} = \text{softmax}(e_{\mathbf{q},\mathbf{k}_i})$$

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

3. WEIGHTED AMOUNT

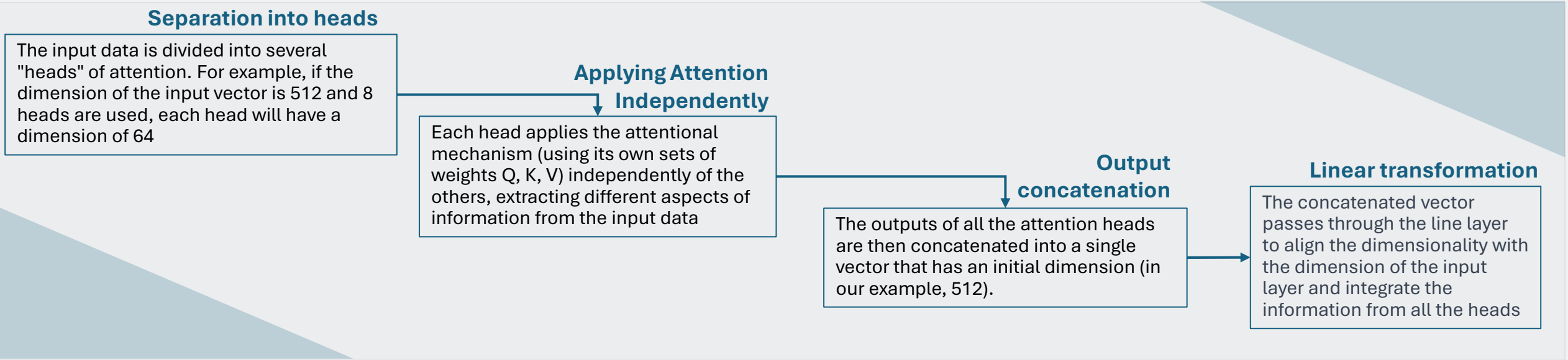
- Attention weights are multiplied by the corresponding Value vectors: the greater the attention weight of a word, the greater the contribution of its Value vector to the output.
- The Value vectors are then summed to form a weighted sum, which becomes the output for the current word.

$$\text{attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \sum_i \alpha_{\mathbf{q},\mathbf{k}_i} \mathbf{v}_{\mathbf{k}_i}$$

*) The attention mechanism was proposed somewhat earlier than the full-fledged architecture of transformers. One of the key moments in the development of the attention mechanism was in 2014, when the "**Neural Machine Translation with Attention**" model was introduced by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.

MULTI-HEAD ATTENTION

WIDENING THE FIELD OF VISION: Instead of having a single set of attentional weights (as in the standard attentional mechanism), the multi-headed attentional mechanism has multiple sets of weights, each of which is trained to pick up on different aspects of information. This allows the model to process information from different angles at the same time.



The multi-head attention mechanism has significant advantages (and is the main base for the success of transformers):
Improved understanding of context (taking into account information from different levels of abstraction)
Concurrency and efficiency (each "head" works independently, which makes it possible to do parallel calculations)
Rich representation of the data (this mechanism allows the model to capture different aspects of the data)

MASKED MULTI-HEAD ATTENTION

In the Transformer model, the term "masked multi-head attention" refers to a special version of the multi-head attention mechanism that prevents information from "leaking" from future tokens when processing each token in the sequence.

- **Self-attention**, sometimes referred to as internal attention, is a mechanism that allows a transformer model to process and encode a sequence of inputs, taking into account the dependencies between all tokens in that sequence. Unlike traditional recurrent neural networks (RNNs), self-attention allows the model to consider all sequence positions at once, making the process more efficient and eliminating long-distance communication issues.

- **Masked Attention**. In the context of Transformer, "masking" means that when calculating self-attention for a given token, the model does not take into account (or "does not see") subsequent tokens in the sequence. This is achieved by applying a mask to the attention scores before applying softmax.

- **Masking Order**:
  - The model is trained to generate text, and is currently processing the word at position i.
  - To predict the next word at position i+1, the model should only use information from positions 1 before i and should not have access to information from positions i+1 onwards.
  - Masked multi-head attention accomplishes this by applying a mask to the attention values in such a way that any "future" tokens (after i) are not counted.

For a sequence of 4 tokens (when the last token is generated):

$$\begin{pmatrix} 0 & -inf & -inf & -inf \\ 0 & 0 & -inf & -inf \\ 0 & 0 & 0 & -inf \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- First Line of Mask: When processing the first token, the model can only consider information about the first token.
- Second line of the mask: allows you to take into account the information about the first and second tokens when processing the second token,
- … ….
- Values of $-\inf$ block "looking into the future", since after using softmax, tokens corresponding to such values will receive very little weight.

INPUT/OUTPUT EMBEDDING

Embedding mechanisms allow you to perform transformations between words and vectors. For Input Embedding, it is the conversion of input words (tokens) into fixed-size vectors, and for Output Embedding, the output (in the form of vectors with the probabilities of the next word) is converted back to words

Embedding, in the context of natural language processing and neural networks, is used to convert categorical data (e.g., words) into real number vectors that can be processed by a model, and vice versa.

- **Token encoding**. It is carried out by assigning indexes to each unique word in the dictionary. For example, you can build a dictionary$\{'cat'\colon 0, 'dog'\colon 1, 'fish'\colon 2\}$.
- **Using the Embedding Matrix**. Creates a matrix of size E $[vocab\_size, embedding\_\mathrm{dim}]$, here $vocab\_size$ — This is the size of the dictionary and embedding_dim is the dimension of the vector space in which the words are embedded. The elements of the E matrix are initialized with random numbers and then trained in conjunction with other model parameters.
- **Converting Words to Vectors**. For a word with an index$i$, its embedding (vector representation) is obtained as the i-th line of the matrix E. This can be expressed as $v = E[i]$.

Arbitrarily Initialized Embedding Matrix

$$E = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{pmatrix}$$

Embedding для "cat" будет: $\{0.1, 0.2, 0.3\}$

In some architectures (e.g., OpenAI's GPT models), the matrix for Input Embedding and Output Embedding may be shared, which is one way to reduce the total number of parameters in the model.

POSITIONAL ENCODING

Positional Encoding in Transformers is used to provide information about the order of words or tokens in a sequence. Transformers are inherently order-insensitive, as they process inputs in parallel, so without adding any position information, the transformer model will not be able to take into account word order, which is critical for many NLP tasks

Positional Encoding solves the problem of word positioning by adding additional signals to the input so that the model can take into account the order of the words in the sentence. Basic algorithm:

1. **Adding Positional Encoding Vectors**. A positional encoding vector of the same dimension is added to each input vector (after input embedding). In this way, the model receives information about both the content of the word (via input embedding) and its position in the sentence.

2. **Positional Encoding (PE) :** It is carried out with the help of the sine and cosine function. For each pos position and each dimension i in the embedding vector, we find the values:

$$PE(pos, 2i) = sin\frac{pos}{10^{4\times\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = cos\frac{pos}{10^{4\times\frac{2i}{d_{model}}}}$$

Here:

- pos – Position of the word in the sequence
- d_{model} – Embedding Dimension
- 10^4 – Base of a hyperbolic function for frequency scaling (heuristic)

3. **Item Accounting for Processing:** After adding positional encoding, the vectors are passed to the transformer layers, where the model takes into account both the content of the words and their position in the sequence.

FEED FORWARD

In the Transformer model, the Feed Forward Neural Network (FFNN) is a component of each transformer layer and is a simple neural network that is applied to each position of the input sequence separately and independently

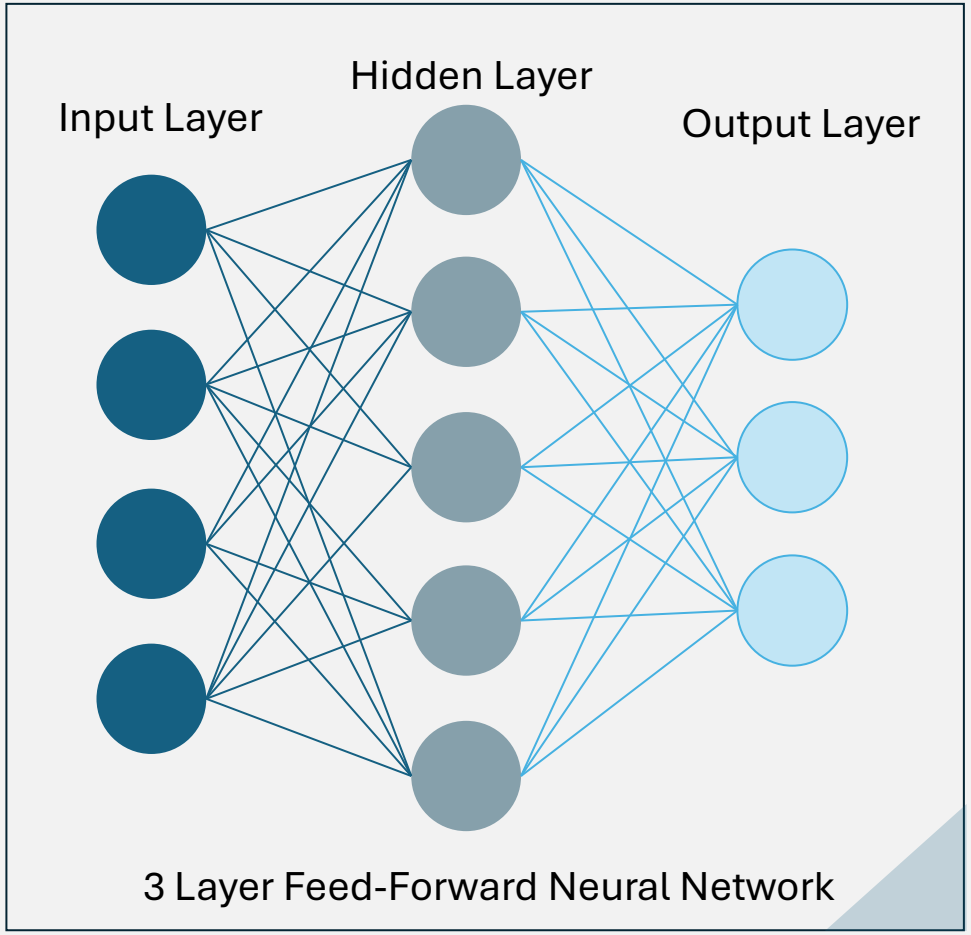
Operation Scheme:

- **Pointwise Transformation.** The Feed Forward network is applied separately to each position in the sequence. This means that for each element (e.g., for each token or each word vector), the network performs the same operation.
- **Operation**. A typical Feed Forward network in a transformer consists of two linear transformations with nonlinear activation between them:

$$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$$

Здесь W_1,W_2 ,b_1,b_2 – Network Settings, $\max(0, z)$ – Activation FunctionReLU

With Feed Forward, the network increases the presentation power of the transformer model, allowing it to learn from more complex aspects of the data, accounting for complex dependencies in the data. At the same time, FFNN, unlike multi-head attention, processes each position independently



3 Layer Feed-Forward Neural Network

ADD AND NORM OPERATIONS

The ADD & NORM operations yield what's known as addition, which eliminates the problem of a fading or exploding gradient, as well as normalization, which stabilizes the learning by bringing the outputs of each layer so that they have a zero mean and a single standard deviation

In deep neural networks, the signal passes through many layers and can be significantly weakened (especially due to the contribution of the activation function) or, conversely, unpredictably amplified. This leads to unstable learning, and both non-saturation activation functions (such as ReLU) and various normalizations are used to combat the problem. In the transformer model:

- The ADD function is a residual connection and is applied to the output signal from the Multi-Head Attention or FFNN. The main mechanism is to add the output signal to the input signal before transferring it to the next layer:

$$x' = Layer(x) + x$$

Here Layer(x) – This is the result of the Attention or FFNN,
x – input

- After the addition operation (ADD) and before the result is fed to the next layer, the normalization (NORM) of each element is performed individually:

$$x'' = Norm(x')$$

Here $Norm(x')$ – is a normalization function applied to the previous result x' after the Add operation:

Вычисление среднего

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x'_i$$

Вычисление дисперсии

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (x'_i - \mu)^2$$

Собственно нормализация

$$\hat{x_i} = \frac{x'_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Масштабирование и сдвиг

$$x''_i = \gamma \hat{x}_i + \beta$$

ϵ — This is a small constant added for numerical stability (to avoid division by zero).

γ и β —scale and panning parameters that are trained with the rest of the model. These parameters allow the normalization layer to change the shape of the data distribution if needed to train the model efficiently.

LINEAR LAYER

A Linear Layer is the simplest type of layer in neural networks, where each input is connected to each output by a linear function. In the context of transformers, it is used to transform inputs by multiplying by weights and adding bias.

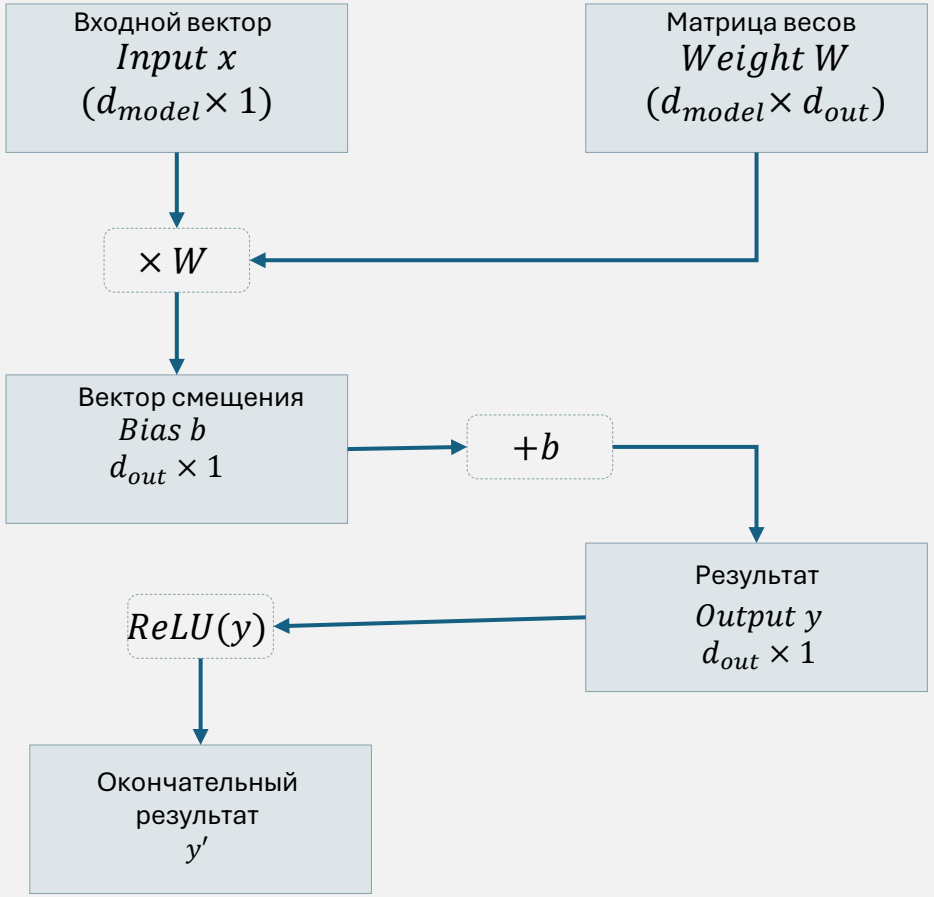
If the input of a line layer is a vector x, then the output y of that layer can be expressed as:

$$y = xW + b$$

Here, W is the weight matrix, b is the bias vector, and both parameters are trained with the model

How it use in transformers:

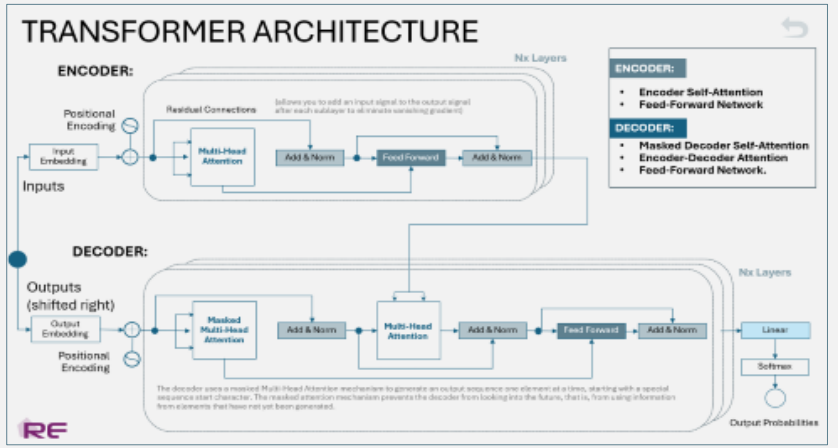
- In the attention block, line layers are used to convert inputs into keys, queries, and values before passing them to the attention engine.
- In the FFNN block, line layers are used to transform the output from the attention engine. Typically, an FFNN contains two line layers with a nonlinear activation function between them.
- In the final part of the transformer model, a linear layer (often followed by softmax) is used to transform the model output into the probabilities of the next token in the sequence.



GENERAL TRANSFORMER ARCHITECTURE

Transformers that implement LLM have a complex architecture, which consists of a first approximation of an encoder and a decoder (Encoder, Decoder). Both subsystems contain multiple layers. Each Encoder and Decoder layer has sublayers.

In specific implementations of transformers (BERT, GPT), depending on the model, only Encoder (BERT, focus on text understanding), only Decoder (GPT, focus on text generation), or both can be used. The main sublayers are:



Embedding Layer

- **Token Embeddings**: Converts each word or token of the input text into a fixed-size vector. This is the first layer that receives the input data and converts it into a format suitable for the model.
- **Positional Embeddings**: Because Transformers don't have an internal idea of the order of words in a text, they need positional vectors to store information about the position of each word.

Encoder (for BERT models) or Decoder (for GPT models))

- **Attention Layers**: A key component of transformers that allows the model to take into account the context of all input text when processing each word. It consists of the vectors Q (Query), K (Key) and V (Value), as well as a mechanism for calculating attention weights.
- **Feed-Forward Layers**: These layers follow the layers of attention and are usually fully connected neural networks that are applied to each input position separately and in the same way.
- **Normalization Layers**: Layers to normalize the output of each sublayer (attention and perceptron) before passing it to the next layer.
- **Residual Connections**: Help avoid the problem of fading gradient in deep networks by adding the input of the previous layer to the output of the next layer.

Output & Classification

- **Output layer**: For models that predict the next word (as in GPT), the last layer is often a layer that converts the output of the last transformer layer back into a word vector space, where each element corresponds to the probability of the next word.
- **Classification layer**: For models that perform classification tasks (as in BERT), the output may be one or more layers that predict classes for the input data.

LIMITATIONS OF TRANSFORMERS ARCHITECTURE

Transformers take a significant step forward by providing high-quality text generation and processing. At the same time, this architecture has a number of limitations, which are still being overcome.

High Compute Requirements: Training Transformers, especially large language models such as GPT-3 or BERT, requires significant computational resources, including powerful GPUs or even TPUs. This makes them less accessible to researchers or companies with limited resources.

Limited processing of tabular and numeric data: Transformers are optimized for working with sequences and text data, where context and semantic connections are important. Numerical data in tables, such as transaction data, balances, customer ages, etc., are of a completely different nature.

Privacy concerns: Data processing for privacy-sensitive environments (e.g., finance and healthcare) requires not only accurate data generation, but also ensuring that the data provided does not contain information that identifies real users, which is difficult or impossible to integrate into the work of transformers due to centralization.

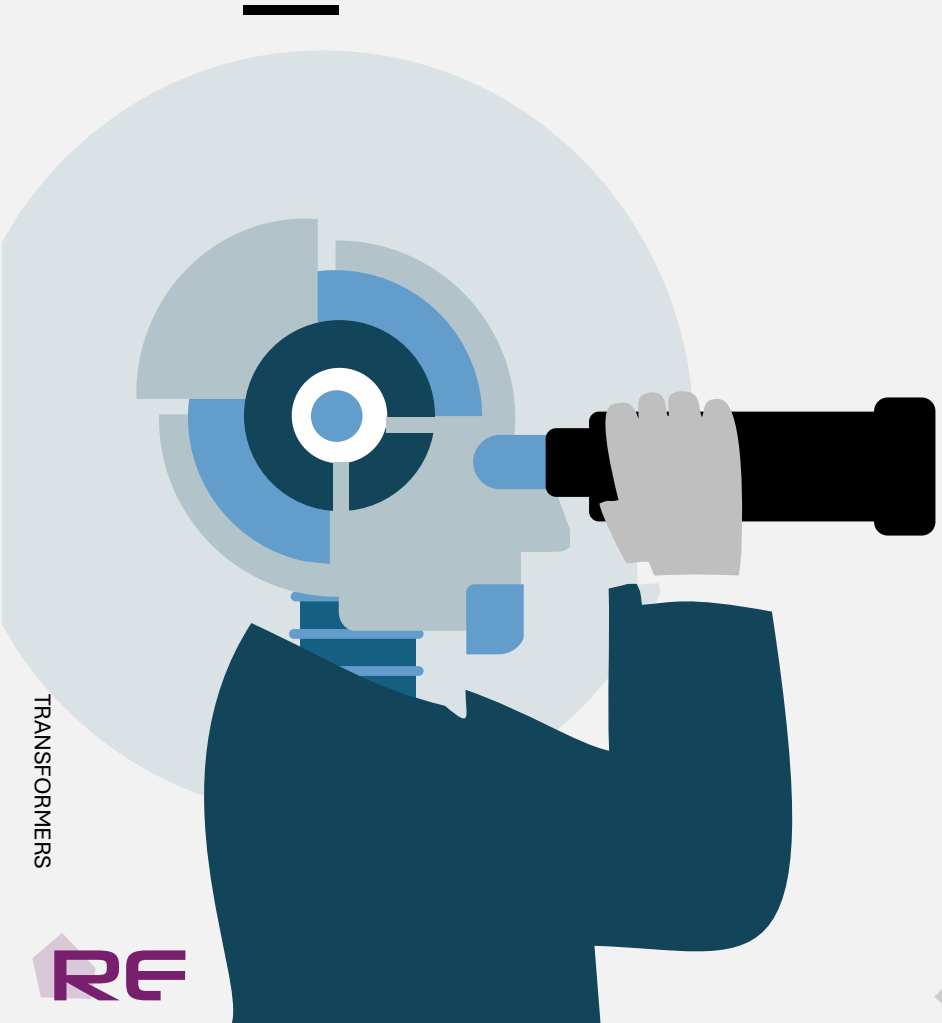
Problems with long sequences: Transformers use an attentional mechanism to process each element of the sequence in the context of all other elements. This results in a quadratic increase in the number of calculations and memory consumption relative to the length of the sequence.

No time component: Many tasks require consideration of temporal dependencies (for example, the sequence of transactions or the history of communication with the client). Transformers do not include components for time series analysis by default, unlike models that focus on temporal data, such as RNN or LSTM.

Bias and Ethical Issues: Transformers are trained on data that may contain biased or unwanted information. Models can adopt and amplify these biases, which can lead to biased or incorrect texts.

PROMISING AREAS

The architecture of transformers continues to be the subject of intense research and development. Scientists and engineers are constantly looking for ways to improve, optimize, and expand the capabilities of Transformers.



EFFICIENCY AND SCALE

- Sparse transformers (reduce computational complexity)
- Optimization of attention mechanisms

VERSATILITY AND MULTIMODALITY

- Multimodal transformers (work simultaneously with text, images, sound)
- Universal language models (not requiring retraining for the task)

PRIVACY & SUSTAINABILITY

- Differential Privacy
- Eco-friendly and energy efficient

INTERPRETABILITY AND TRANSPARENCY

Developing methods and tools to better understand and interpret how Transformers make decisions and what underpins their predictions.

TRANSFORMERS INFRASTRUCTURE

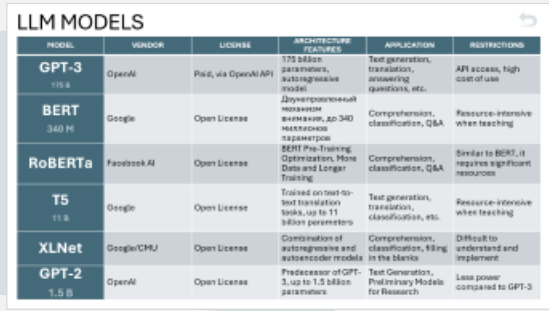
Transformers take a significant step forward by providing high-quality text generation and processing. At the same time, this architecture has several limitations, which are still being overcome.

1 Basic Libraries and Frameworks

- **Hugging Face's Transformers**: One of the most popular libraries for working with Transformers. Provides pre-trained models, integration with TensorFlow and PyTorch.
- **TensorFlow и PyTorch**: The two main libraries for deep learning that are used to build, train, and deploy transformer models.

2 LLM Models

- **Large Models**: GPT-4 (OpenAI, 1.7 Tr), Jurassic-1 (AI21, 178 B), Megatron-Turing NLG(Nvidia+ MS, 530 B), Switch Transformer (Google, 1/6 Tr), Gopher (DeepMind, 280 Tr)
- **Small Models**: GPT-2 (OpenAI, 1.5 B), BERT (Google, 340 M), T5 (Google, 11 B)



3 Repositories & Communities

- **Hugging Face's Model Hub**: A platform for sharing and using pre-trained transformers models. Users can upload their own models and use models provided by the community.
- **Google's TensorFlow Model Garden**: A repository from Google containing implementations of various machine learning models, including transformers-based models.

4 Computing Projects & Solutions

- **Google TPUs** (Tensor Processing Units): Google's purpose-built coprocessors optimized for machine learning.
- **NVIDIA GPUs**: NVIDIA GPUs widely used to train neural networks.
- **LLM Mesh**: Infrastructure Solutions for Efficient Scaling and Deployment of Large Language Models.
- **Forks & Extensions**: There are many forks and extensions of the core libraries that make additions, optimizations, or specialized solutions for specific tasks or requirements.

PRACTICAL USE

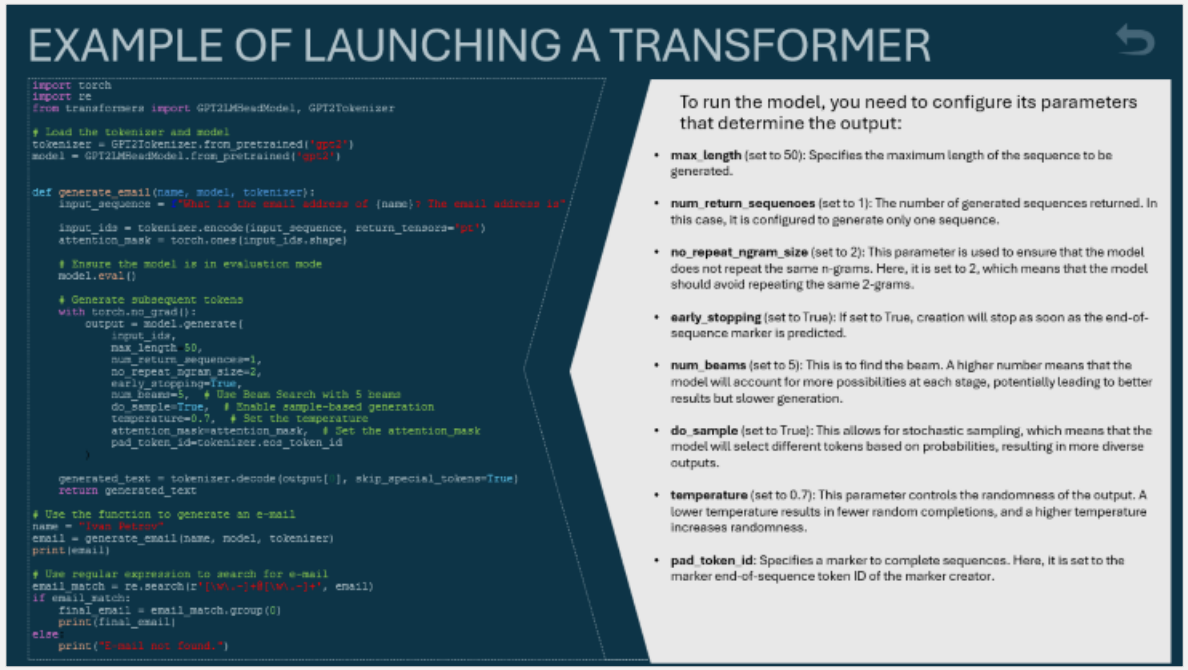
Here, as an illustration, we consider the simplest example of using GPT-2 to generate e-mail using a given username.

This example shows how to run the GPT-2 model (free to use) to generate an e-mail address for a selected user's full name

1. The model is loaded in advance, then a "prompt" is used for practical work. The quality of the propmpt, as well as parameters such as temperature, affect the output of the result.

2. To run on-premises, a Python VM must be configured, and the necessary packages must be added using the pip manager (in this case, PyTorch and tranformers)

3. To work successfully, you need the correct version of CUDA for the GPU, a sufficient amount of RAM (RAM from 16 GB



INPUT: Ivan Petrov

OUTPUT: What is the email address of Ivan Petrov? The email address is: Ivan.Petrov@gmail.com

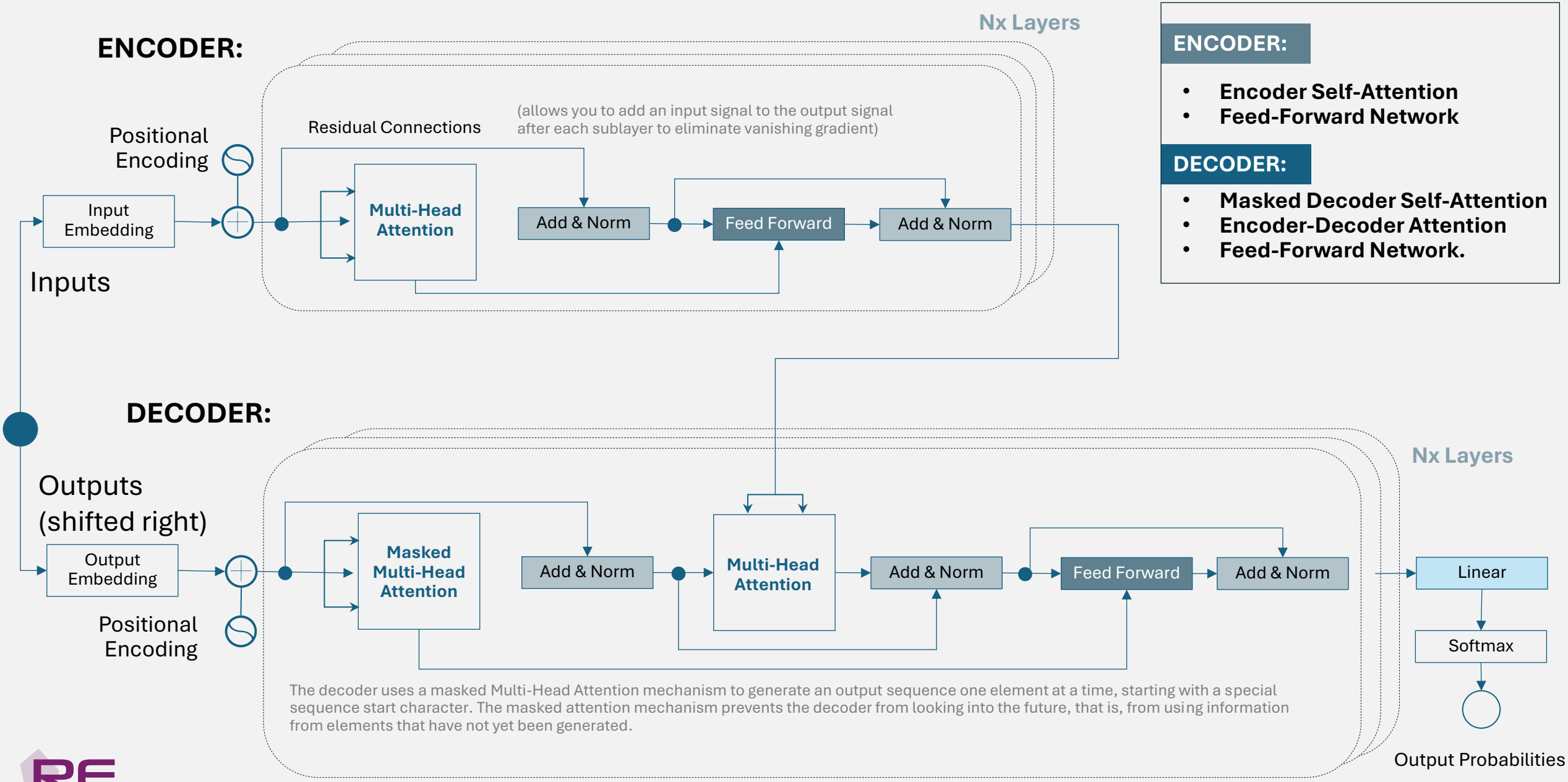
THANK YOU

APPENDIX

LLM MODELS

MODEL	VENDOR	LICENSE	ARCHITECTURE FEATURES	APPLICATION	RESTRICTIONS
GPT-3 175 B	OpenAI	Paid, via OpenAI API	175 billion parameters, autoregressive model	Text generation, translation, answering questions, etc.	API access, high cost of use
BERT 340 M	Google	Open License	Двунаправленный механизм внимания, до 340 миллионов параметров	Comprehension, classification, Q&A	Resource-intensive when teaching
RoBERTa	Facebook AI	Open License	BERT Pre-Training Optimization, More Data and Longer Training	Comprehension, classification, Q&A	Similar to BERT, it requires significant resources
T5 11 B	Google	Open License	Trained on text-to-text translation tasks, up to 11 billion parameters	Text generation, translation, classification, etc.	Resource-intensive when teaching
XLNet	Google/CMU	Open License	Combination of autoregressive and autoencoder models	Comprehension, classification, filling in the blanks	Difficult to understand and implement
GPT-2 1.5 B	OpenAI	Open License	Predecessor of GPT-3, up to 1.5 billion parameters	Text Generation, Preliminary Models for Research	Less power compared to GPT-3

TRANSFORMER ARCHITECTURE



ENCODER:

- **Encoder Self-Attention**
- **Feed-Forward Network**

DECODER:

- **Masked Decoder Self-Attention**
- **Encoder-Decoder Attention**
- **Feed-Forward Network.**

EXAMPLE OF LAUNCHING A TRANSFORMER

```python
import torch
import re
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')


def generate_email(name, model, tokenizer):
    input_sequence = f"What is the email address of {name}? The email address is"

    input_ids = tokenizer.encode(input_sequence, return_tensors='pt')
    attention_mask = torch.ones(input_ids.shape)

    # Ensure the model is in evaluation mode
    model.eval()

    # Generate subsequent tokens
    with torch.no_grad():
        output = model.generate(
            input_ids,
            max_length=50,
            num_return_sequences=1,
            no_repeat_ngram_size=2,
            early_stopping=True,
            num_beams=5,  # Use Beam Search with 5 beams
            do_sample=True,  # Enable sample-based generation
            temperature=0.7,  # Set the temperature
            attention_mask=attention_mask,  # Set the attention_mask
            pad_token_id=tokenizer.eos_token_id
        )

    generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
    return generated_text

# Use the function to generate an e-mail
name = "Ivan Petrov"
email = generate_email(name, model, tokenizer)
print(email)

# Use regular expression to search for e-mail
email_match = re.search(r'[\w\.-]+@[\w\.-]+', email)
if email_match:
    final_email = email_match.group(0)
    print(final_email)
else:
    print("E-mail not found.")
```

To run the model, you need to configure its parameters that determine the output:

- **max_length** (set to 50): Specifies the maximum length of the sequence to be generated.
- **num_return_sequences** (set to 1): The number of generated sequences returned. In this case, it is configured to generate only one sequence.
- **no_repeat_ngram_size** (set to 2): This parameter is used to ensure that the model does not repeat the same n-grams. Here, it is set to 2, which means that the model should avoid repeating the same 2-grams.
- **early_stopping** (set to True): If set to True, creation will stop as soon as the end-of-sequence marker is predicted.
- **num_beams** (set to 5): This is to find the beam. A higher number means that the model will account for more possibilities at each stage, potentially leading to better results but slower generation.
- **do_sample** (set to True): This allows for stochastic sampling, which means that the model will select different tokens based on probabilities, resulting in more diverse outputs.
- **temperature** (set to 0.7): This parameter controls the randomness of the output. A lower temperature results in fewer random completions, and a higher temperature increases randomness.
- **pad_token_id**: Specifies a marker to complete sequences. Here, it is set to the marker end-of-sequence token ID of the marker creator.