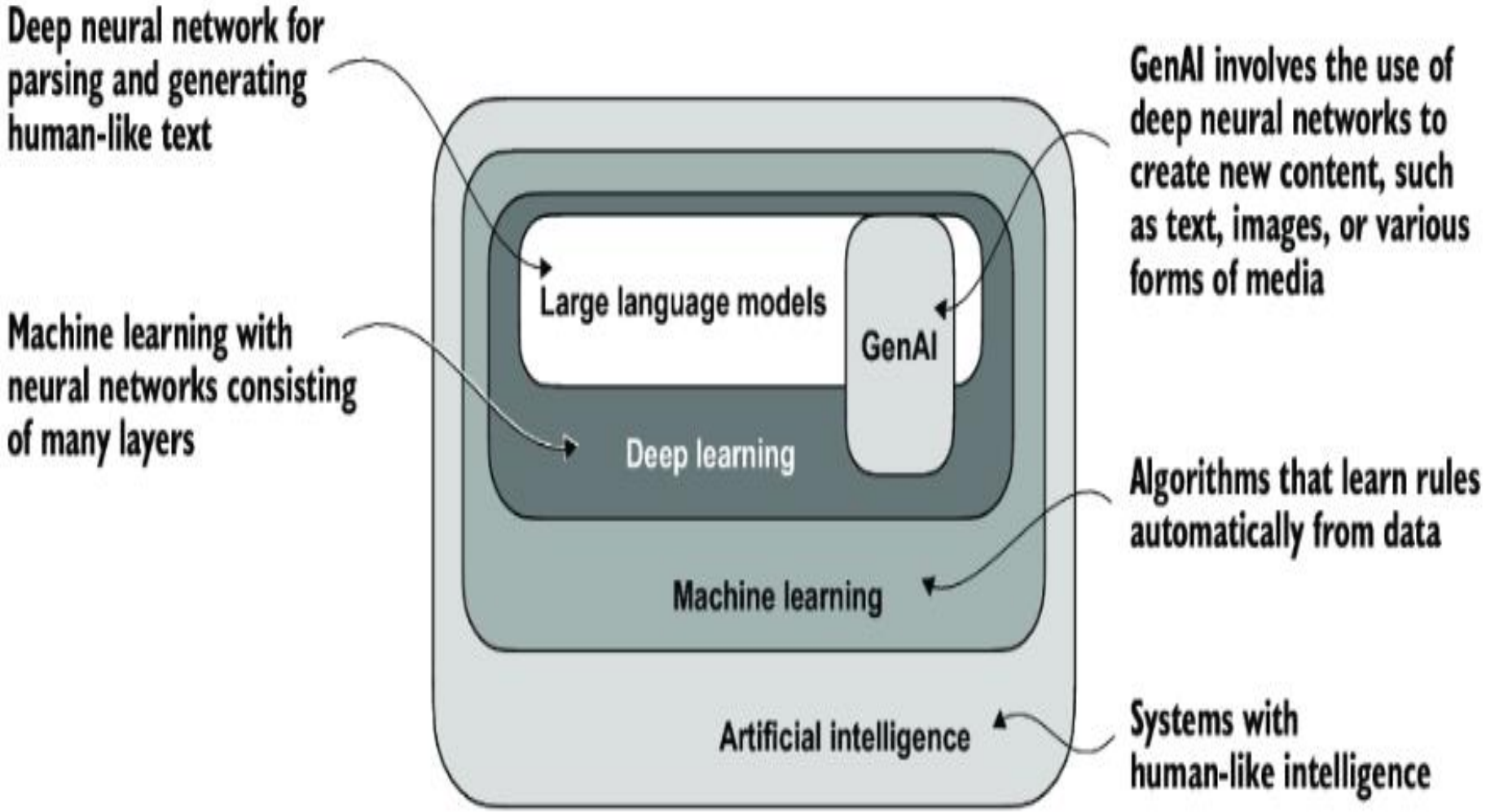
KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE
Narayanaguda, Hyderabad.

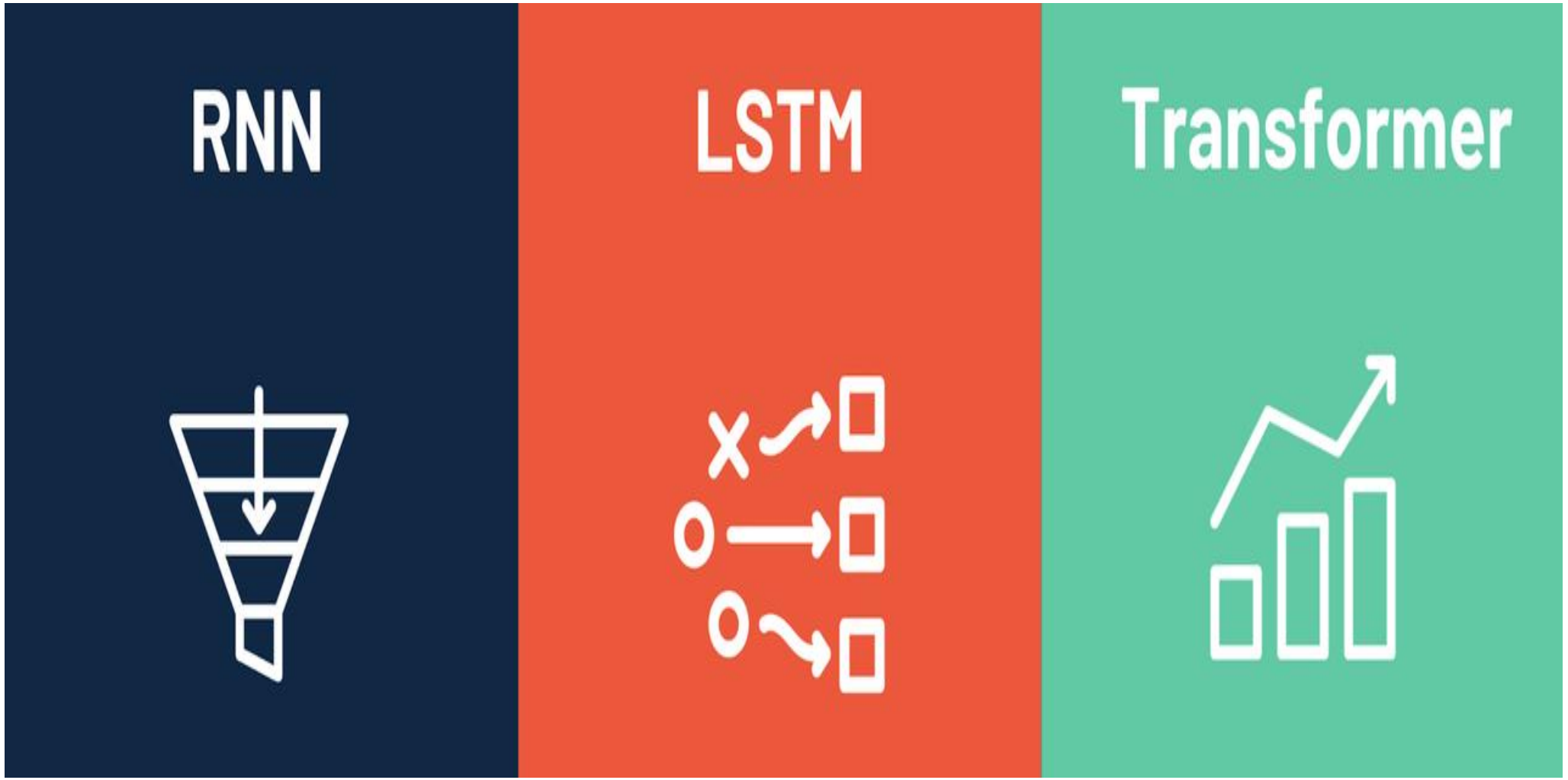
Embedded Learning

TRANSFORMERS

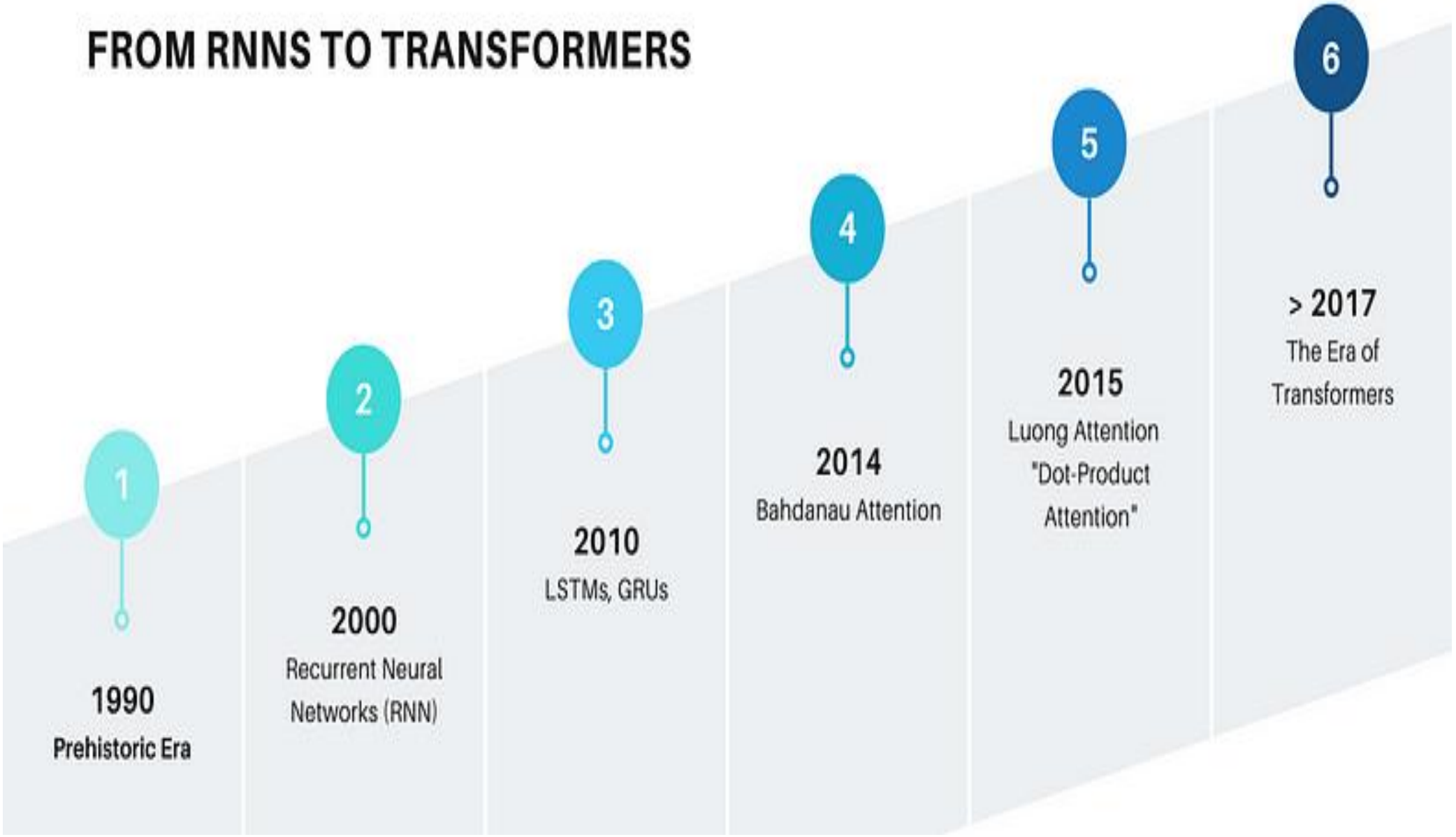
03-02-2025

By
Asha.M
Assistant professor
CSE(AI&ML)
KMIT

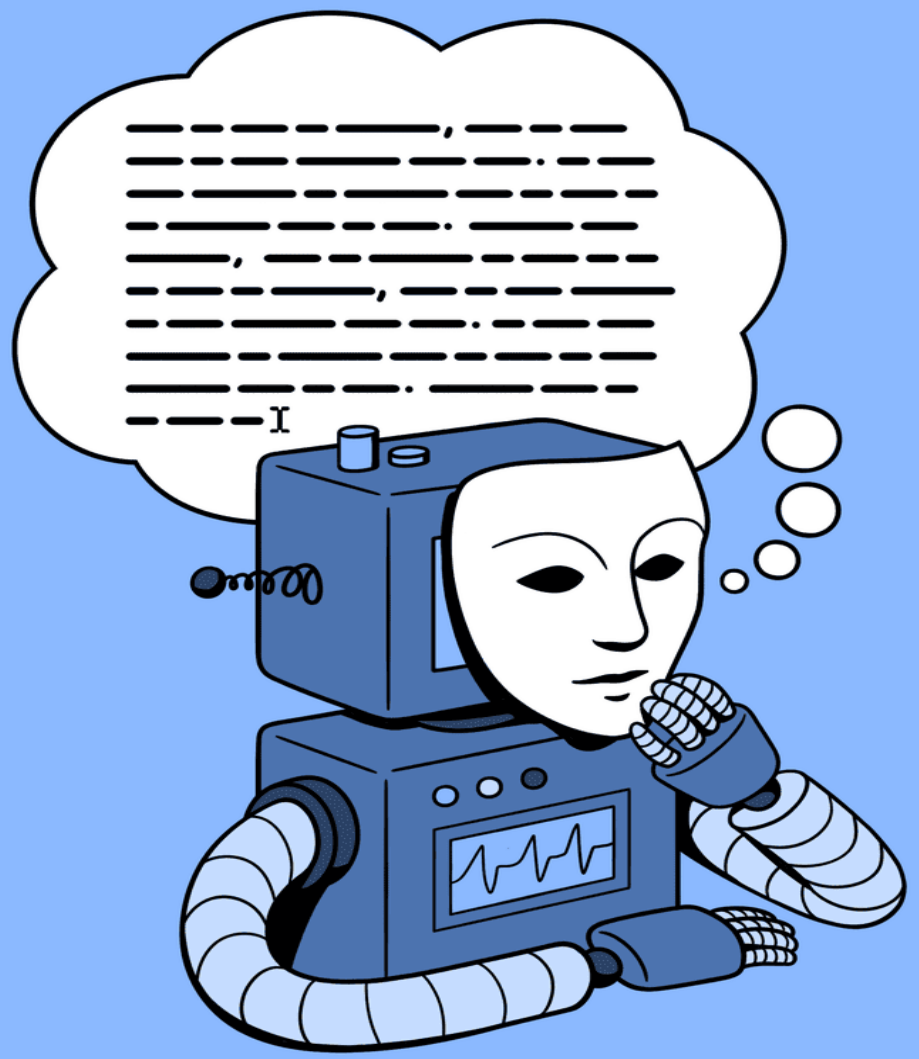

Deep neural network for parsing and generating human-like text
Machine learning with neural networks consisting of many layers
GenAI involves the use of deep neural networks to create new content, such as text, images, or various forms of media
Algorithms that learn rules automatically from data
Systems with human-like intelligence
Large language models
GenAI
Deep learning
Machine learning
Artificial intelligence


RNN
LSTM
Transformer

FROM RNNS TO TRANSFORMERS




00101
11001
LLM
LARGE LANGUAGE MODEL



Large Language Model (LLM)

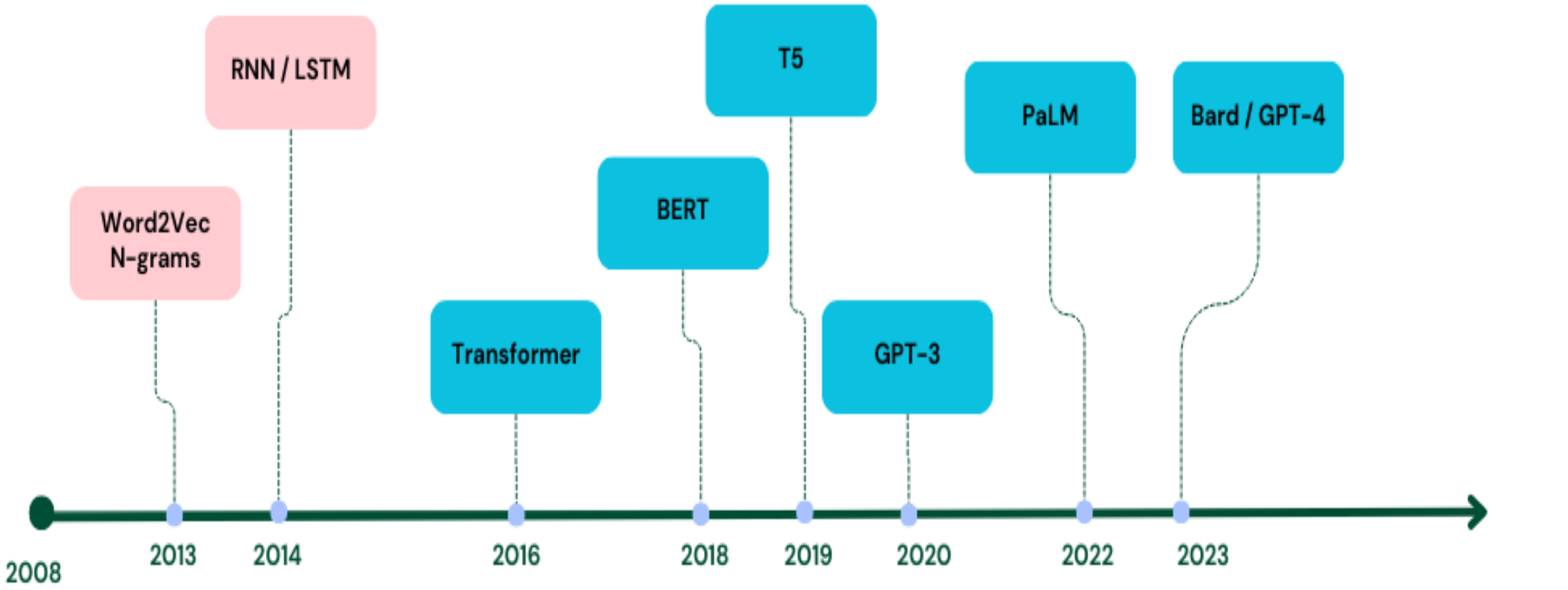
['lärj 'laŋ-gwij 'mä-dᵊl]

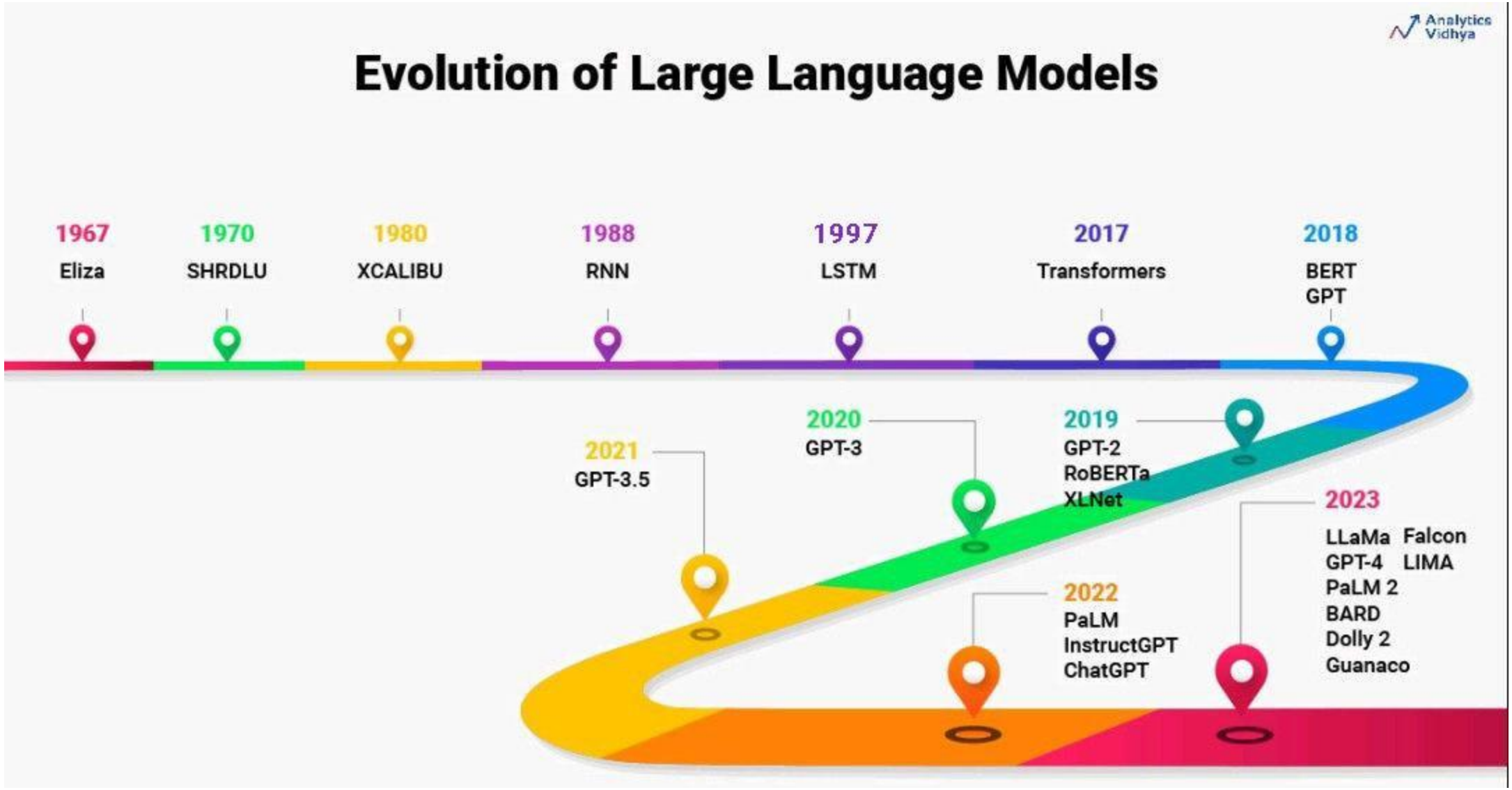
A deep learning algorithm that's equipped to summarize, translate, predict, and generate human-sounding text to convey ideas and concepts.

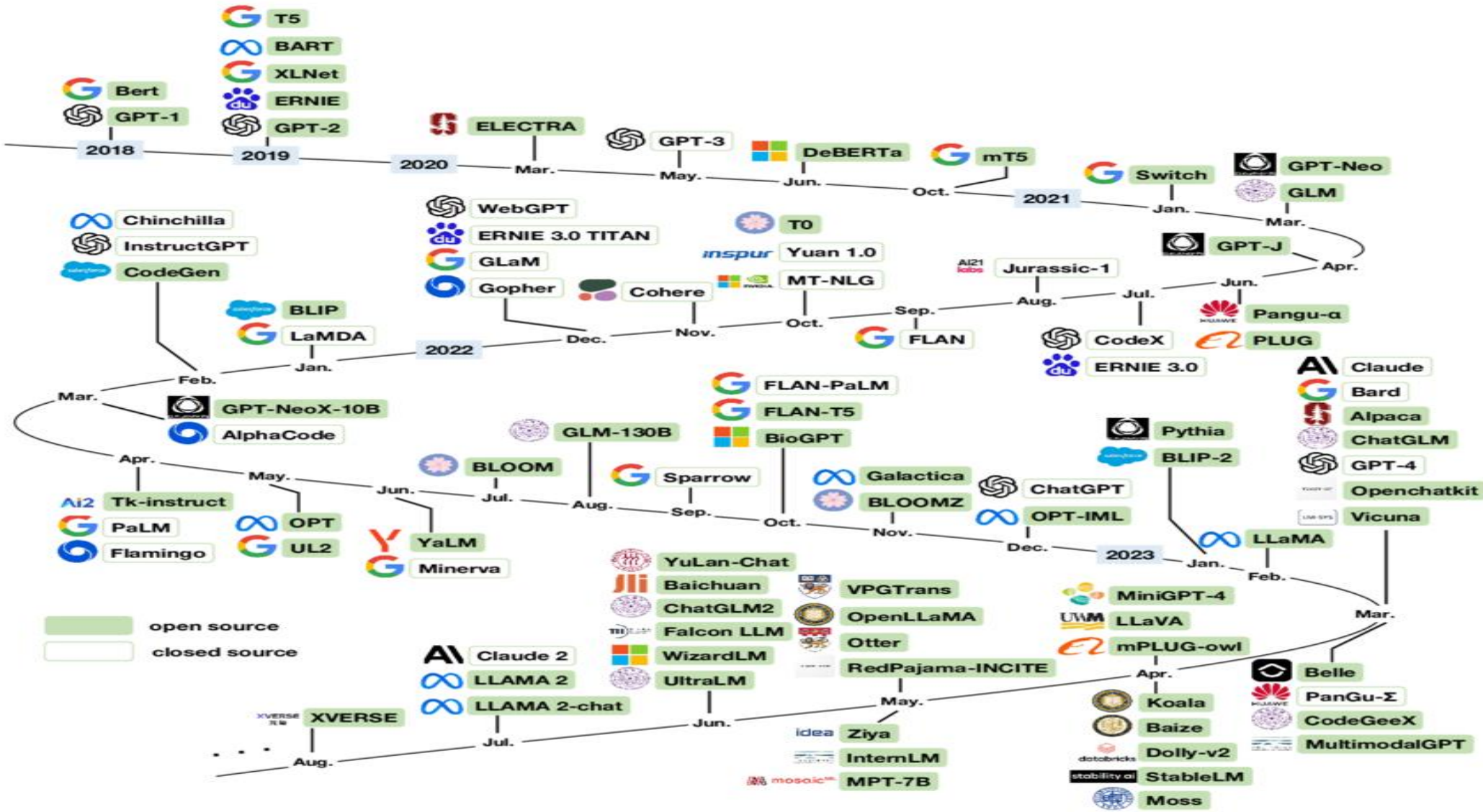
Investopedia

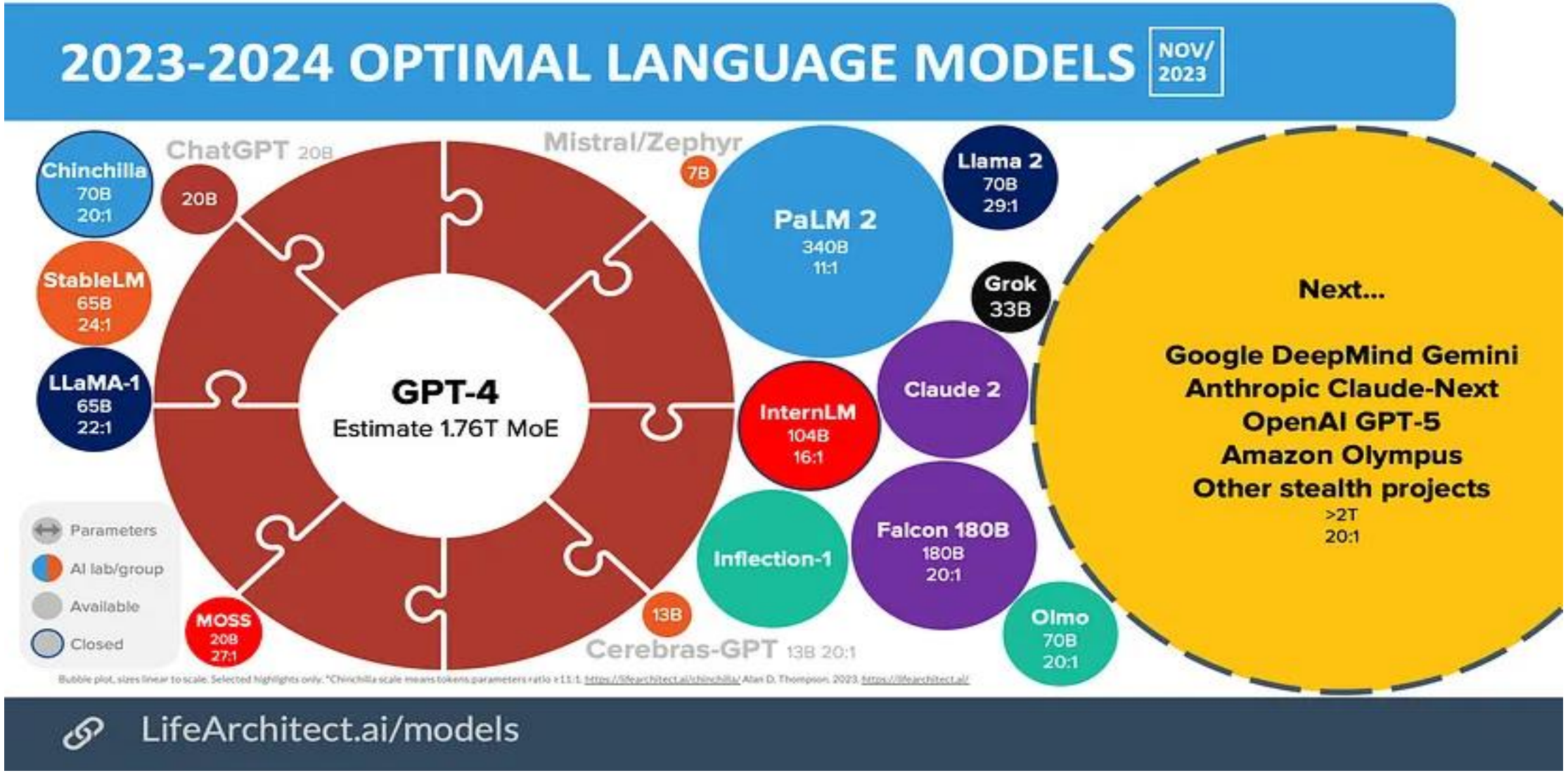
Kmit

Language Model History

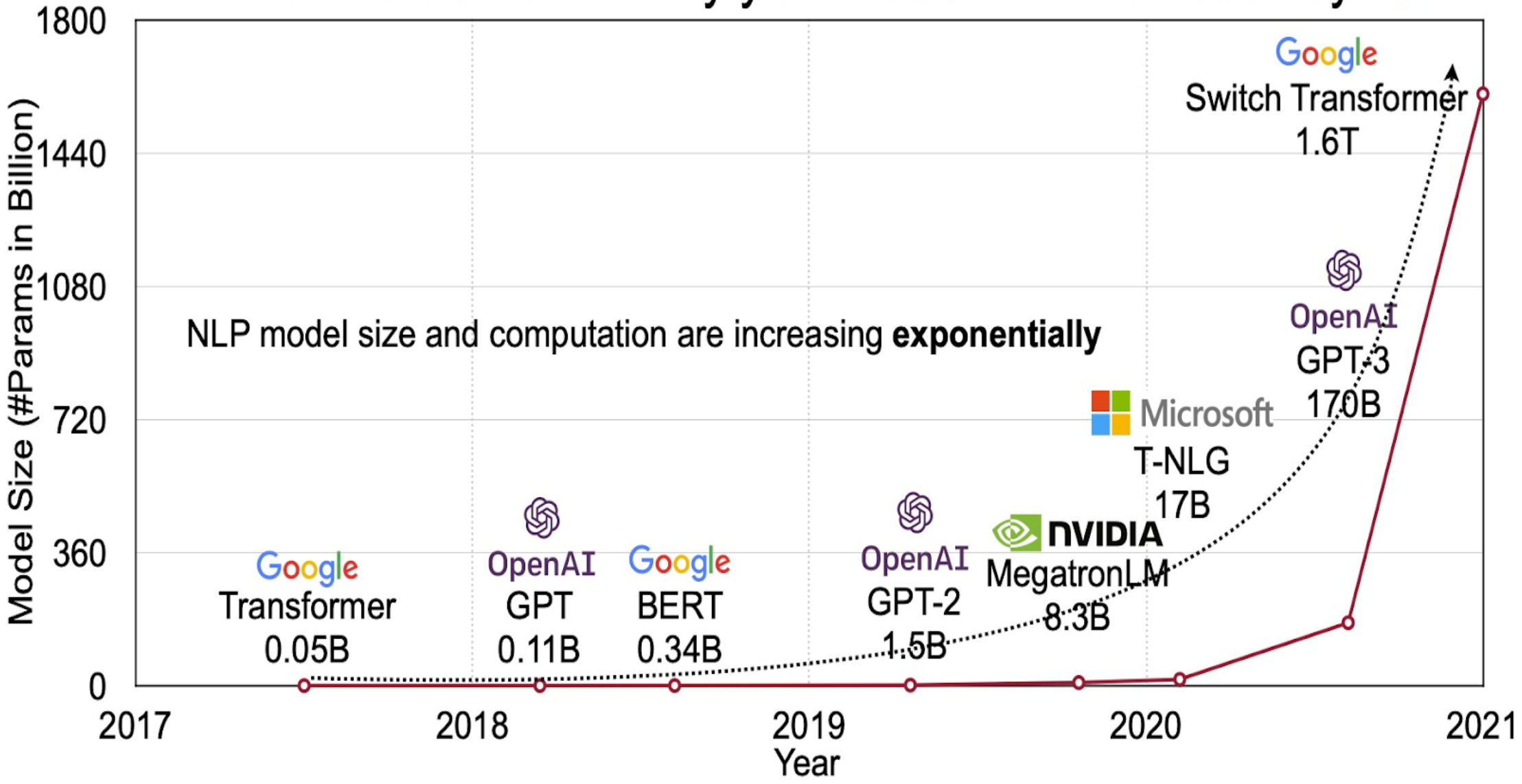



Analytics Vidhya
Evolution of Large Language Models
1967
Eliza
1970
SHRDLU
1980
XCALIBU
1988
RNN
1997
LSTM
2017
Transformers
2018
BERT
GPT
2019
GPT-2
RoBERTa
XLNet
2020
GPT-3
2021
GPT-3.5
2022
PaLM
InstructGPT
ChatGPT
2023
LLaMa
Falcon
GPT-4
LIMA
PaLM 2
BARD
Dolly 2
Guanaco

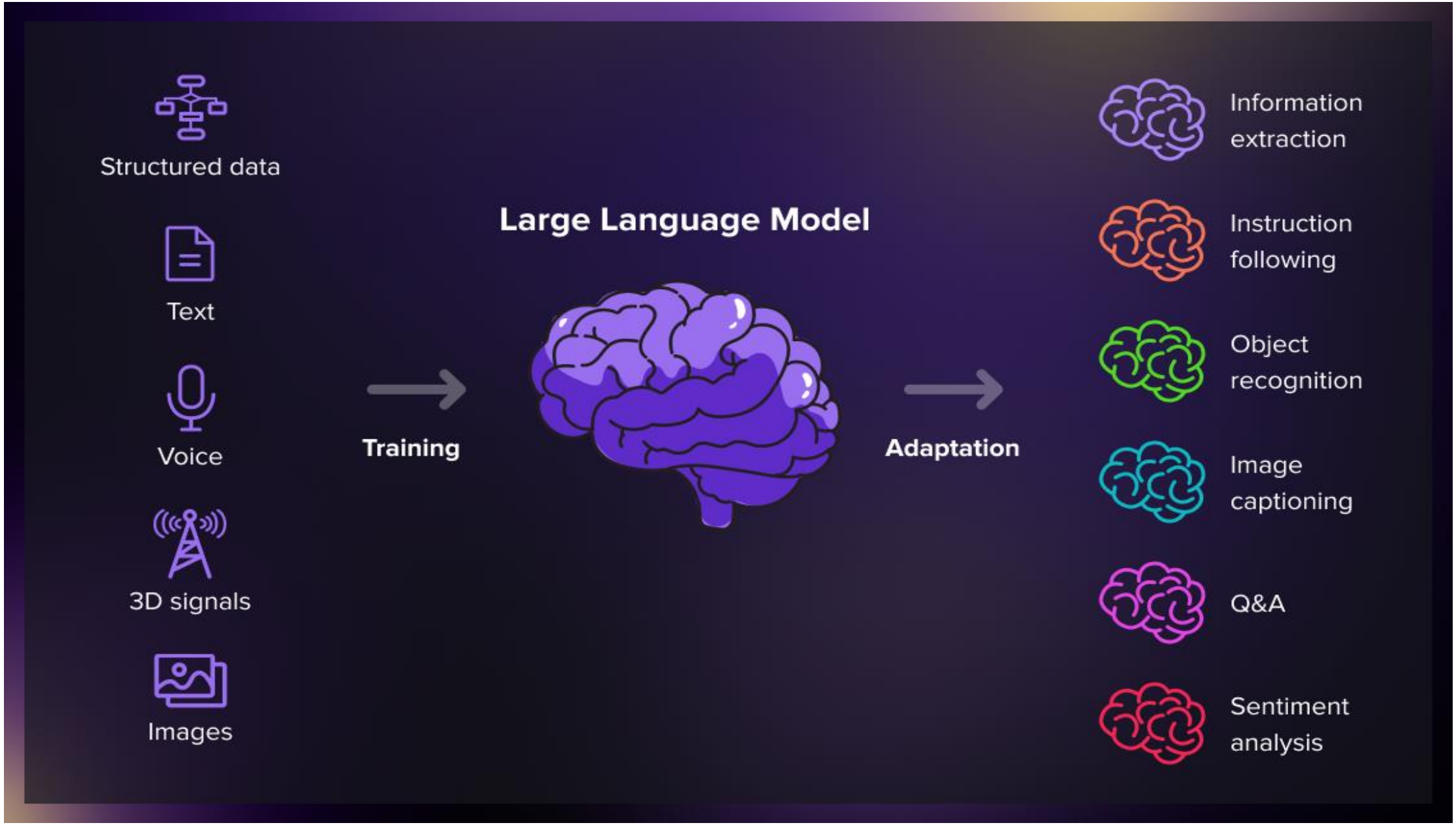

T5
BART
XLNet
ERNIE
GPT-2
Bert
GPT-1
2018
2019
2020
ELECTRA
Mar.
GPT-3
May.
DeBERTa
Jun.
mT5
Oct.
2021
Switch
Jan.
GPT-Neo
GLM
Mar.
GPT-J
Apr.
Jun.
Jurassic-1
Aug.
Jul.
Pangu-α
PLUG
Chinchilla
InstructGPT
CodeGen
WebGPT
ERNIE 3.0 TITAN
GLaM
Gopher
Cohere
T0
Yuan 1.0
MT-NLG
Sep.
Oct.
Nov.
Dec.
FLAN
CodeX
ERNIE 3.0
BLIP
LaMDA
Jan.
2022
Feb.
Mar.
GPT-NeoX-10B
AlphaCode
Apr.
May.
Jun.
Jul.
Aug.
Sep.
Oct.
Nov.
Dec.
2023
Jan.
Feb.
Mar.
Tk-instruct
PaLM
Flamingo
OPT
UL2
YaLM
Minerva
BLOOM
GLM-130B
Sparrow
FLAN-PaLM
FLAN-T5
BioGPT
Galactica
BLOOMZ
ChatGPT
OPT-IML
Pythia
BLIP-2
LLaMA
Claude
Bard
Alpaca
ChatGLM
GPT-4
Openchatkit
Vicuna
YuLan-Chat
Baichuan
ChatGLM2
Falcon LLM
WizardLM
UltraLM
VPGTrans
OpenLLaMA
Otter
RedPajama-INCITE
MiniGPT-4
LLaVA
mPLUG-owl
Koala
Baize
Dolly-v2
StableLM
Moss
Belle
PanGu-Σ
CodeGeeX
MultimodalGPT
Claude 2
LLAMA 2
LLAMA 2-chat
XVERSE
Ziya
InternLM
MPT-7B
open source
closed source

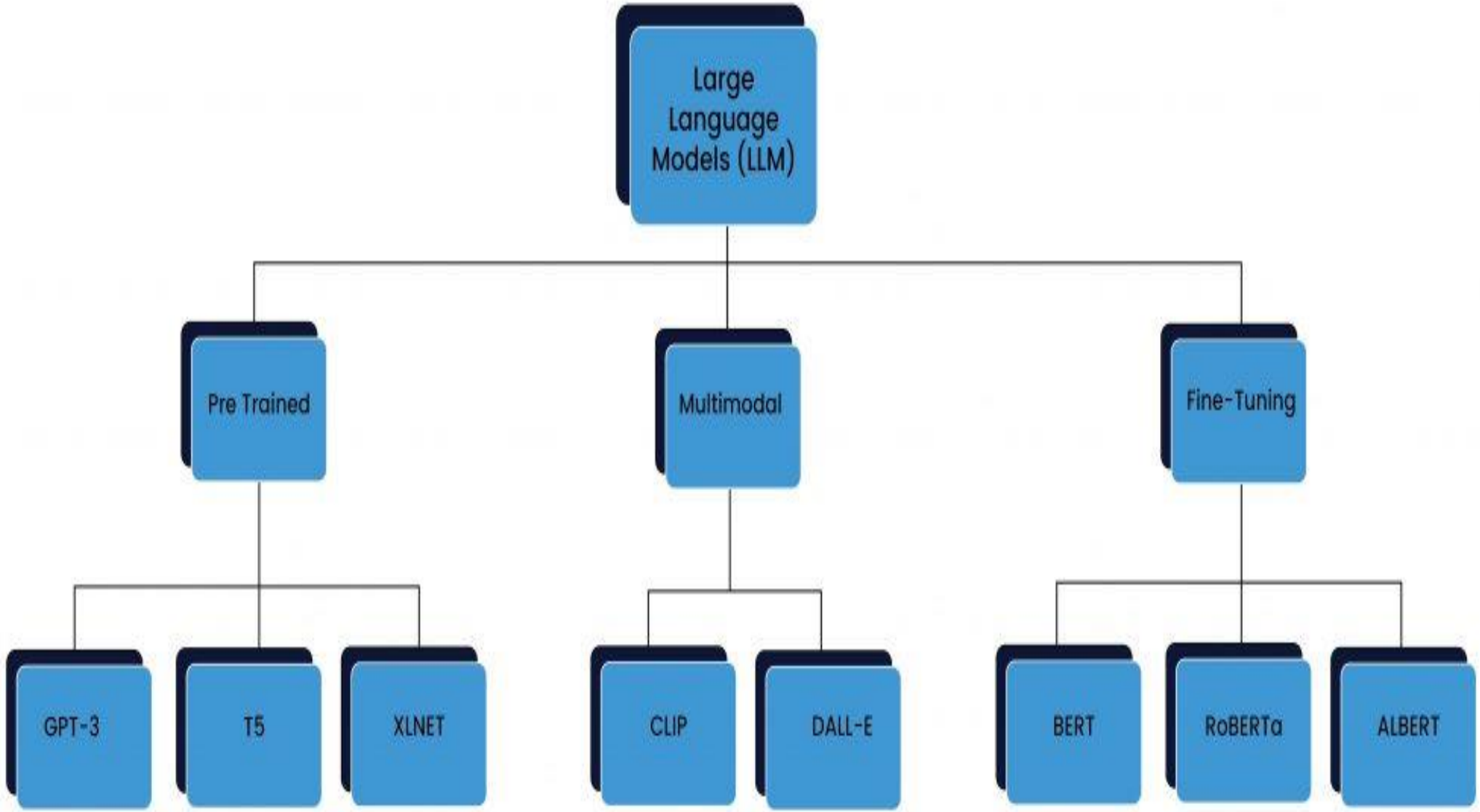

2023-2024 OPTIMAL LANGUAGE MODELS
NOV/ 2023
ChatGPT 20B
20B
Chinchilla
70B
20:1
StableLM
65B
24:1
LLaMA-1
65B
22:1
GPT-4
Estimate 1.76T MoE
Mistral/Zephyr
7B
PaLM 2
340B
11:1
Llama 2
70B
29:1
Grok
33B
InternLM
104B
16:1
Claude 2
Inflection-1
Falcon 180B
180B
20:1
Olmo
70B
20:1
13B
Cerebras-GPT 13B 20:1
MOSS
20B
27:1
Next...
Google DeepMind Gemini
Anthropic Claude-Next
OpenAI GPT-5
Amazon Olympus
Other stealth projects
>2T
20:1
Parameters
AI lab/group
Available
Closed
Bubble plot, sizes linear to scale. Selected highlights only. *Chinchilla scale means tokens:parameters ratio >1:1. https://lifearchitect.ai/chinchilla/ Alan D. Thompson, 2023. https://lifearchitect.ai/
LifeArchitect.ai/models

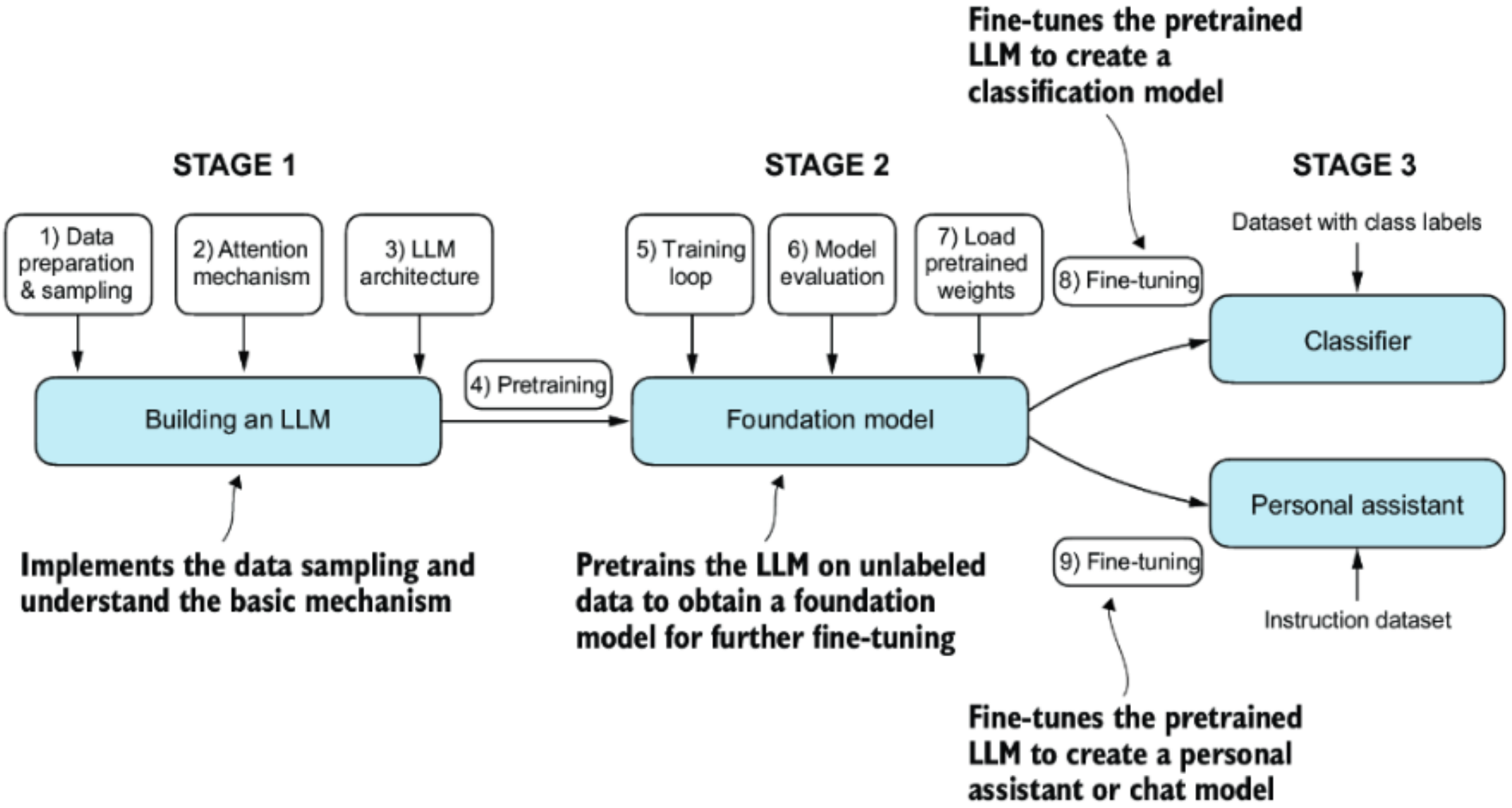
NLP's Moore's Law: Every year model size increases by 10x

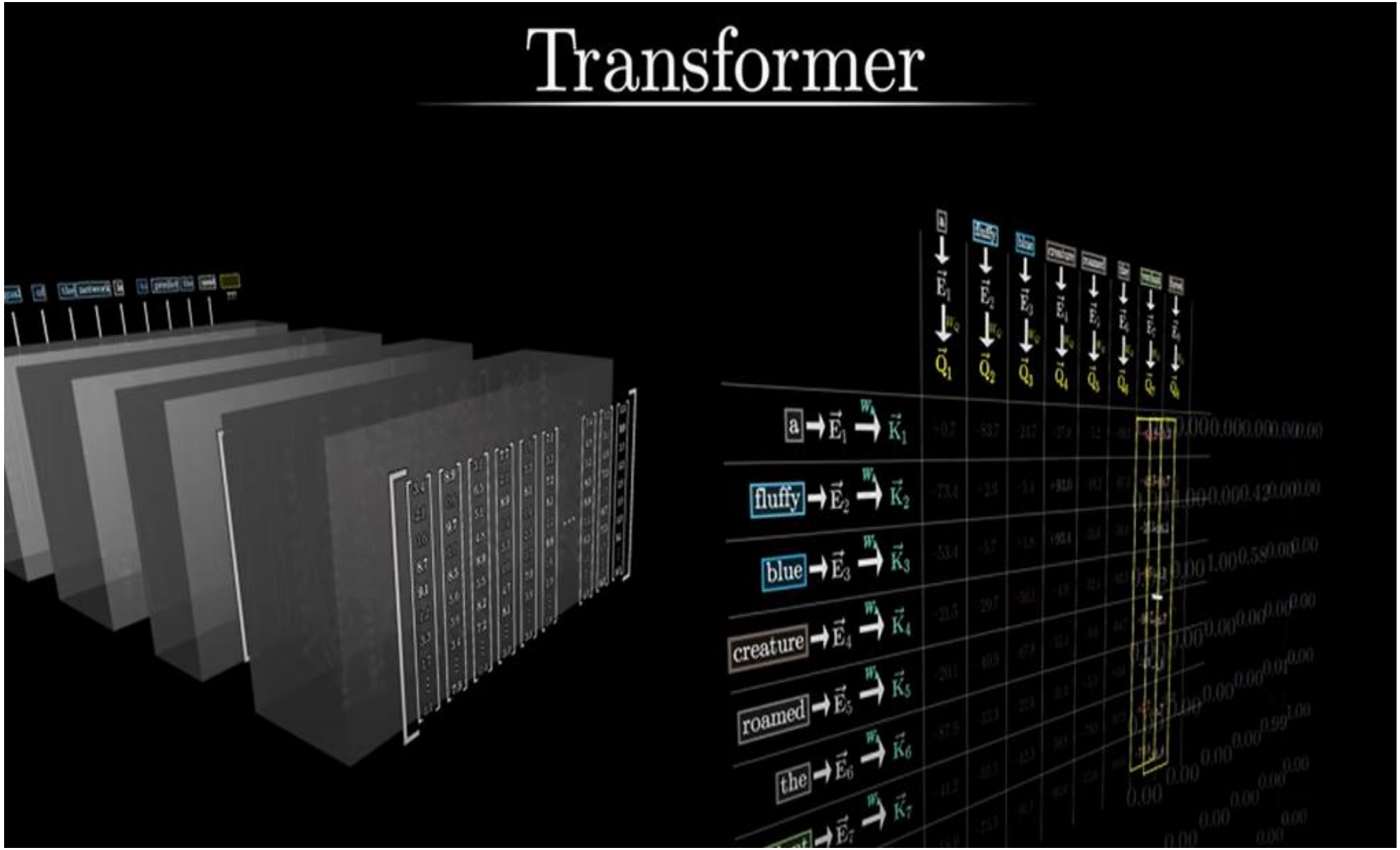


Parameters of transformer-based language models


Structured data
Text
Voice
3D signals
Images
Large Language Model
Training
Adaptation
Information extraction
Instruction following
Object recognition
Image captioning
Q&A
Sentiment analysis


Large Language Models (LLM)
Pre Trained
Multimodal
Fine-Tuning
GPT-3
T5
XLNET
CLIP
DALL-E
BERT
RoBERTa
ALBERT

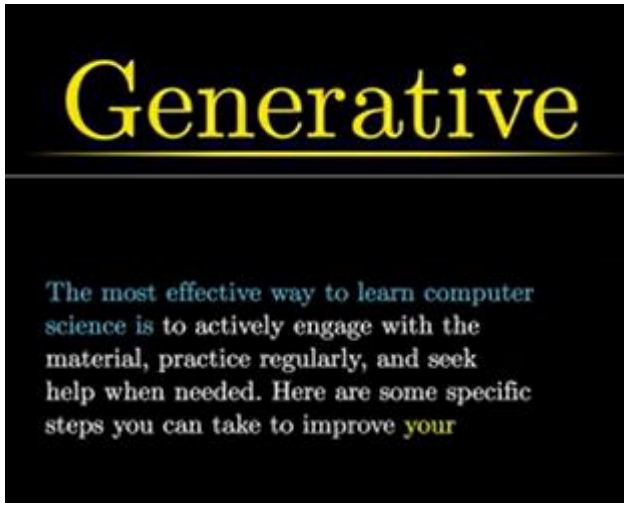

Fine-tunes the pretrained LLM to create a classification model
STAGE 1
STAGE 2
STAGE 3
1) Data preparation & sampling
2) Attention mechanism
3) LLM architecture
5) Training loop
6) Model evaluation
7) Load pretrained weights
8) Fine-tuning
Dataset with class labels
Classifier
4) Pretraining
Building an LLM
Foundation model
Personal assistant
9) Fine-tuning
Instruction dataset
Implements the data sampling and understand the basic mechanism
Pretrains the LLM on unlabeled data to obtain a foundation model for further fine-tuning
Fine-tunes the pretrained LLM to create a personal assistant or chat model

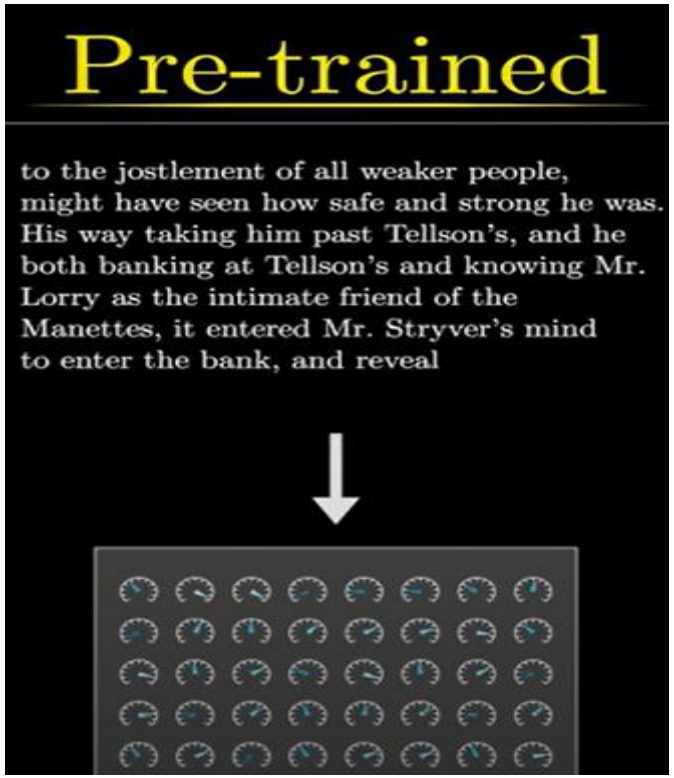

Transformer
a
fluffy
blue
creature
roamed
the

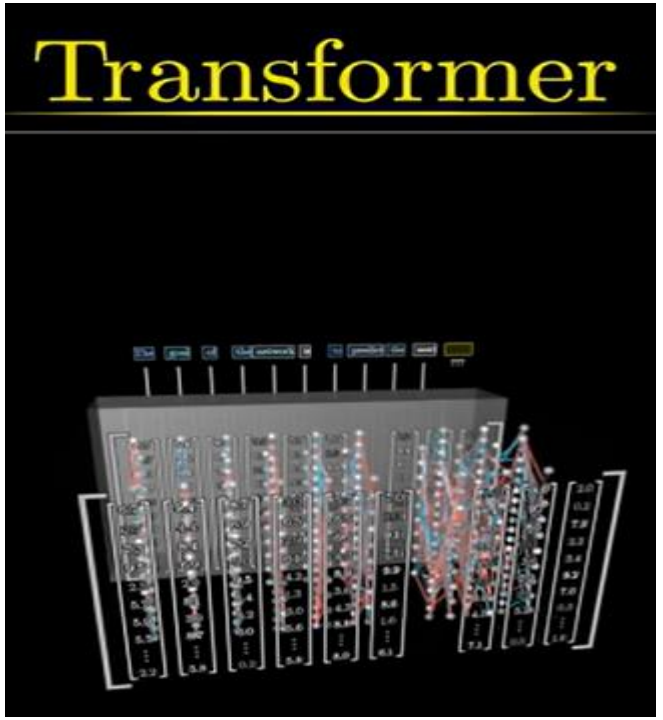
What is a Transformer?

A **Transformer** is a deep learning model introduced in **"Attention Is All You Need" (Vaswani et al., 2017)**. It **replaces recurrent layers (RNNs, LSTMs)** with **self-attention mechanisms**, allowing for **parallel processing** and **long-range dependencies**.

Kmit







Generative → Can **generate human-like text**.

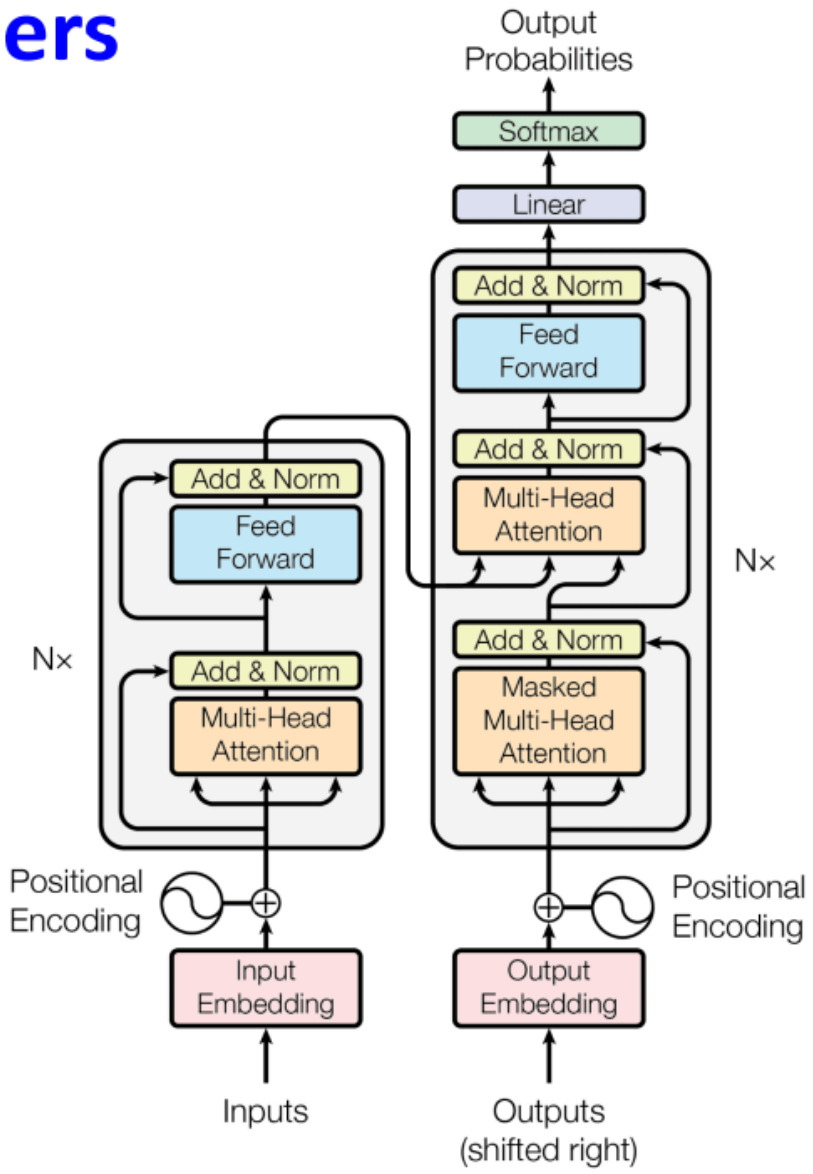
Pre-trained → Learns from **huge datasets** before fine-tuning.

Transformer → Uses **self-attention for efficient language understanding**.

Transformers

- Tokenizaton
- Input Embeddings
- Position Encodings
- Residuals
- Query
- Key
- Value
- Add & Norm
- Encoder
- Decoder

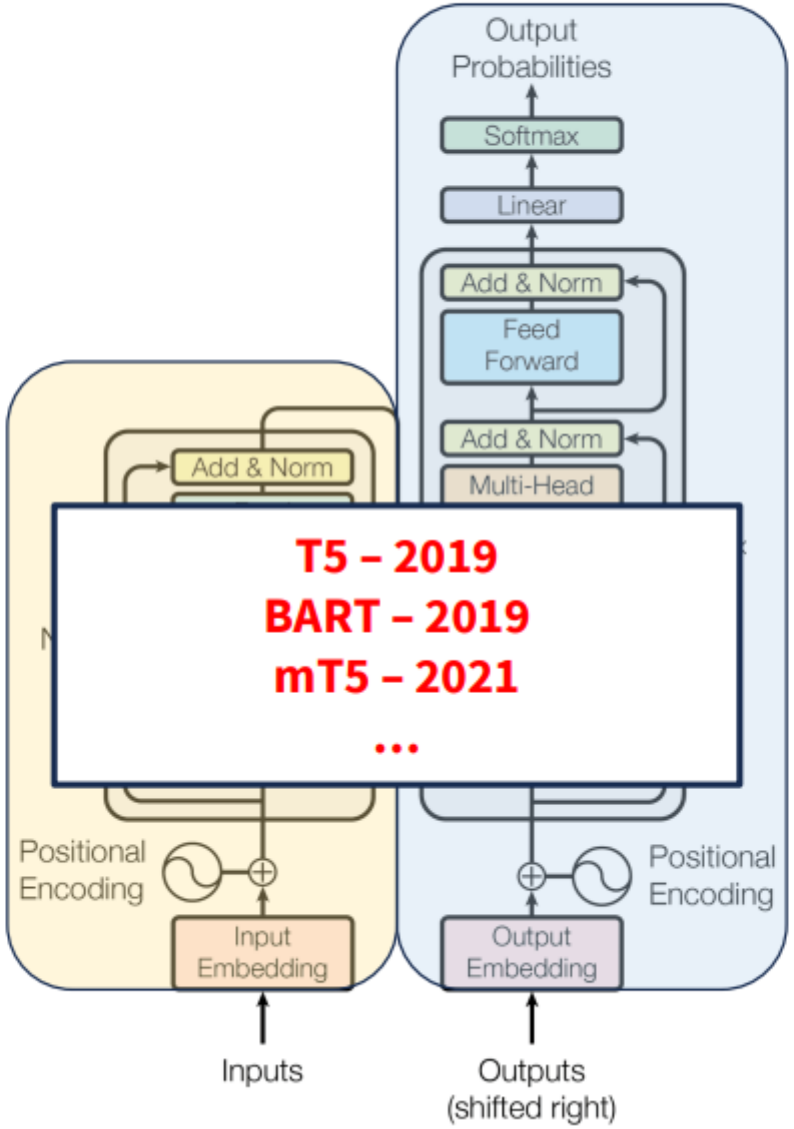
- Attention
- Self Attention
- Multi Head Attention
- Masked Attention
- Encoder Decoder Attention
- Output Probabilities / Logits
- Softmax
- Encoder-Decoder models
- Decoder only models



The LLM Era – Paradigm Shift in Machine Learning

BERT – 2018
DistilBERT – 2019
RoBERTa – 2019
ALBERT – 2019
ELECTRA – 2020
DeBERTa – 2020
…

Representation



GPT – 2018
GPT-2 – 2019
GPT-3 – 2020
GPT-Neo – 2021
GPT-3.5 (ChatGPT) – 2022
LLaMA – 2023
GPT-4 – 2023
…

Generation

Transformers

ENCODER ONLY

aka

auto-encoding models

TASKS

- Sentence classification
- Named entity recognition
- Extractive question-answering
- Masked language modeling

EXAMPLES

BERT, RoBERTa, distilBERT

DECODER ONLY

aka

auto-regressive models

TASKS

- Text generation
- Causal language modeling

EXAMPLES

GPT-2, GPT Neo, GPT-3

ENCODER-DECODER

aka

sequence-to-sequence models

TASKS

- Translation
- Summarization
- Generative question-answering

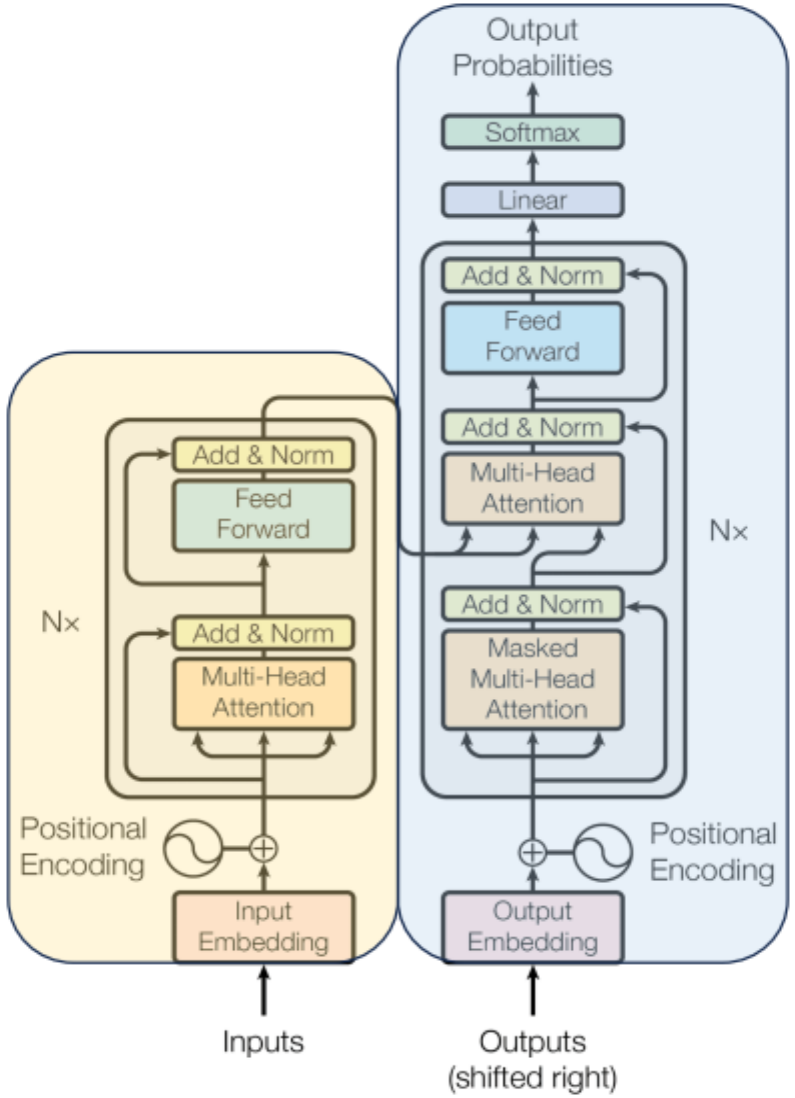
EXAMPLES

BART, T5, Marian

2018 – The Inception of the LLM Era

BERT
Oct 2018

Representation



GPT
Jun 2018

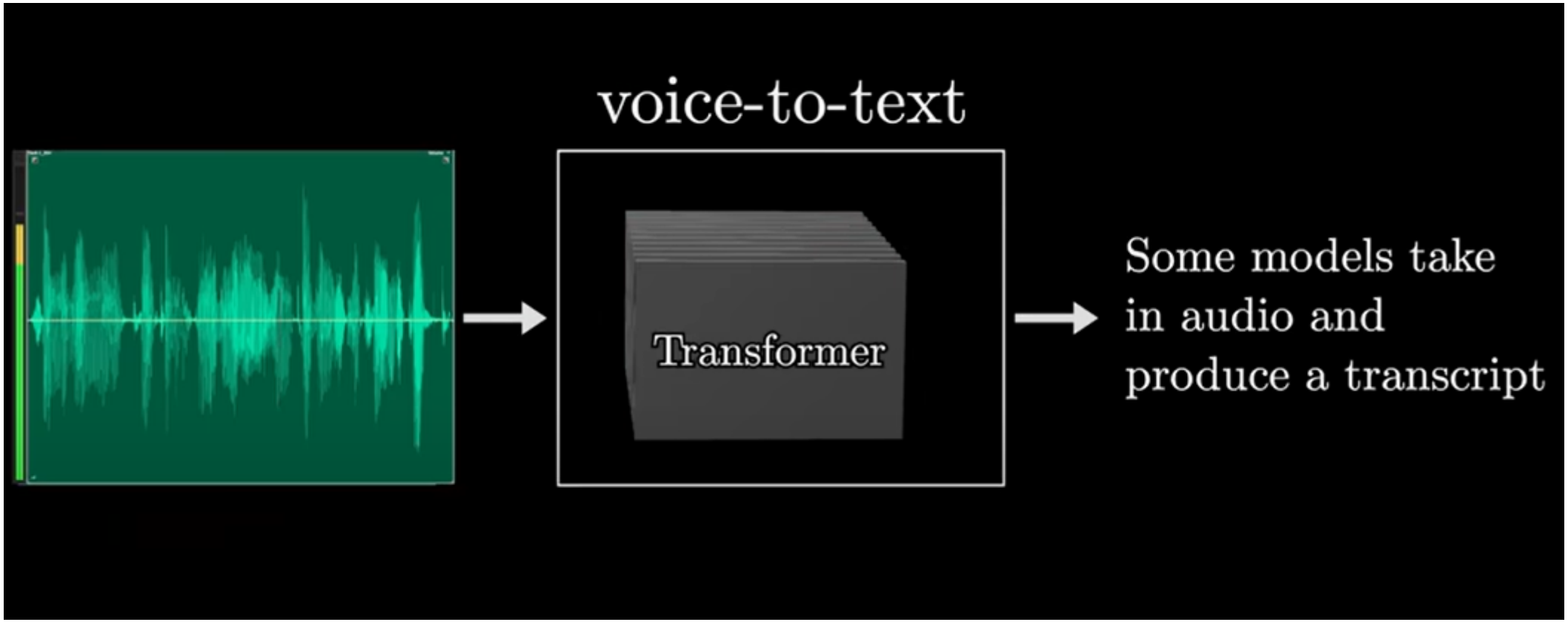
Generation

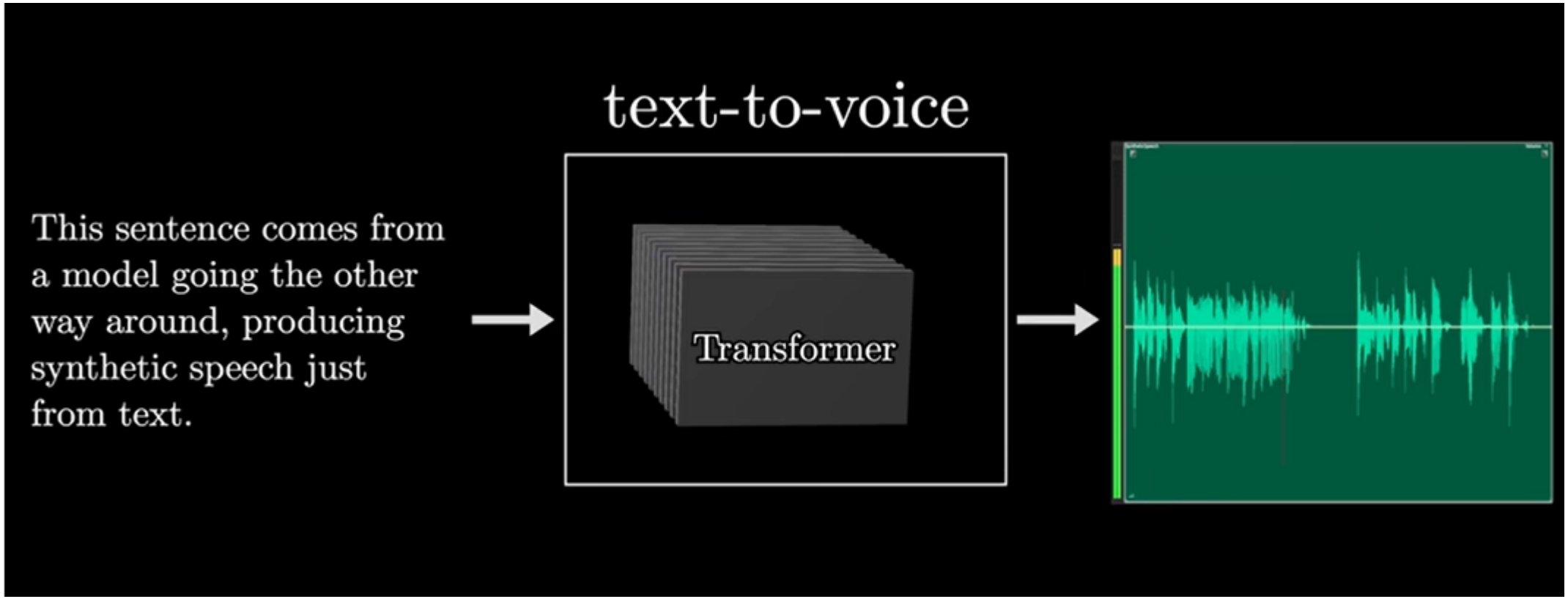
The LLM Era – Paradigm Shift in Machine Learning

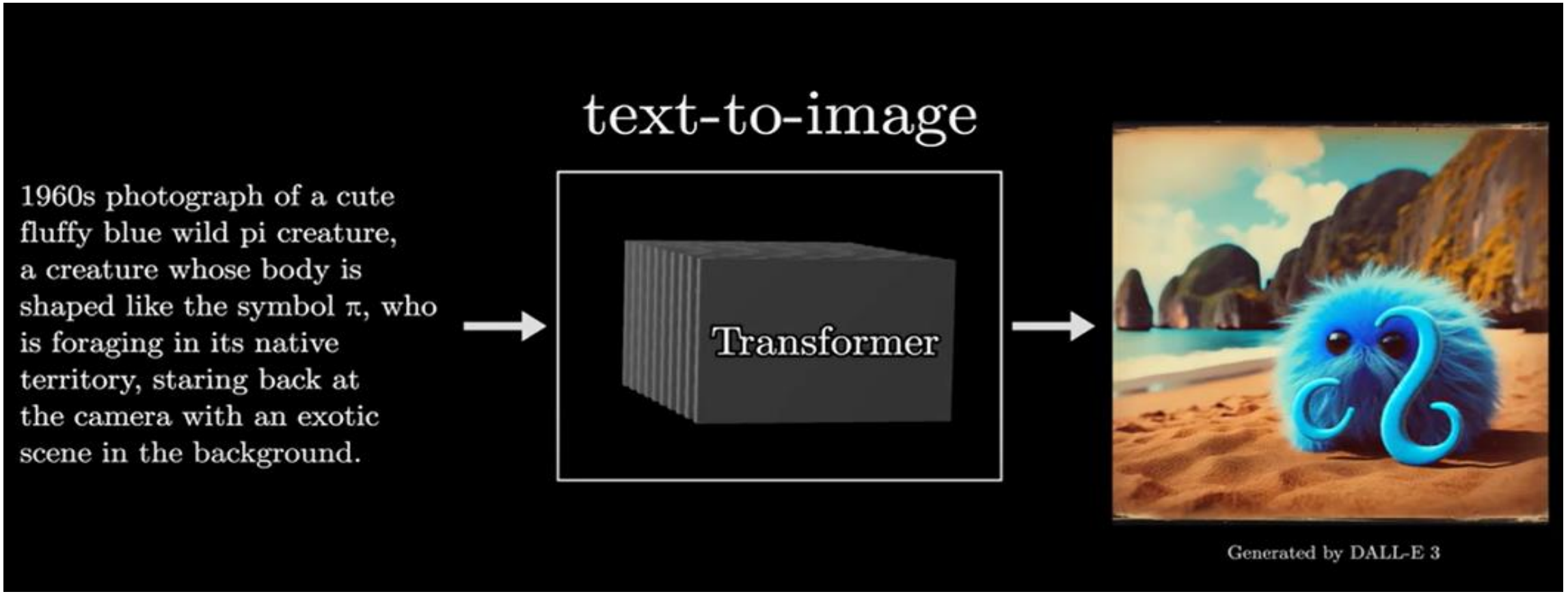
- **What has caused this paradigm shift?**
  - **Problem in recurrent networks**
    - Information is effectively lost during encoding of long sequences
    - Sequential nature disables parallel training and favors late timestep inputs

The LLM Era – Paradigm Shift in Machine Learning

- **What has caused this paradigm shift?**
  - **Problem in recurrent networks**
    - Information is effectively lost during encoding of long sequences
    - Sequential nature disables parallel training and favors late timestep inputs
  - **Solution: Attention mechanism**
    - Handling long-range dependencies
    - Parallel training
    - Dynamic attention weights based on inputs


voice-to-text
Transformer
Some models take in audio and produce a transcript

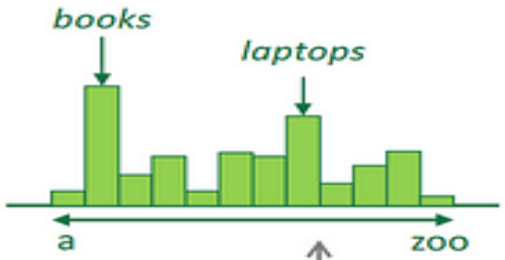

text-to-voice
This sentence comes from a model going the other way around, producing synthetic speech just from text.
Transformer


text-to-image
1960s photograph of a cute fluffy blue wild pi creature, a creature whose body is shaped like the symbol π, who is foraging in its native territory, staring back at the camera with an exotic scene in the background.
Transformer
Generated by DALL-E 3

Recurrent Neural Networks

To increase the context available to our language model, we can use a recurrent neural network, a type of stateful NN. The network for each word maintains a state influenced by both the current word and previous hidden state. This architecture removes the fixes size of the context alleviating the problems of simpler neural language models.

$\hat{y}^{(4)} = P(x^{(5)}|\text{the students opened their})$



output distribution

$$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

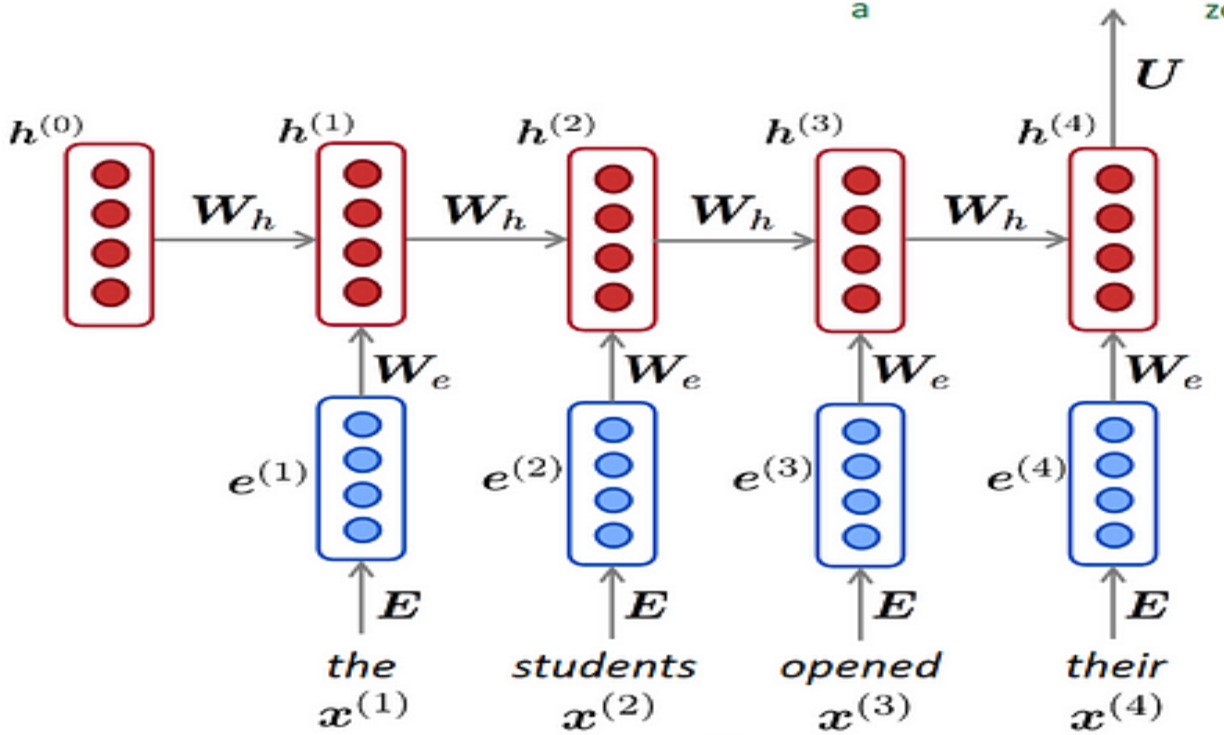
$h^{(0)}$ is the initial hidden state

word embeddings

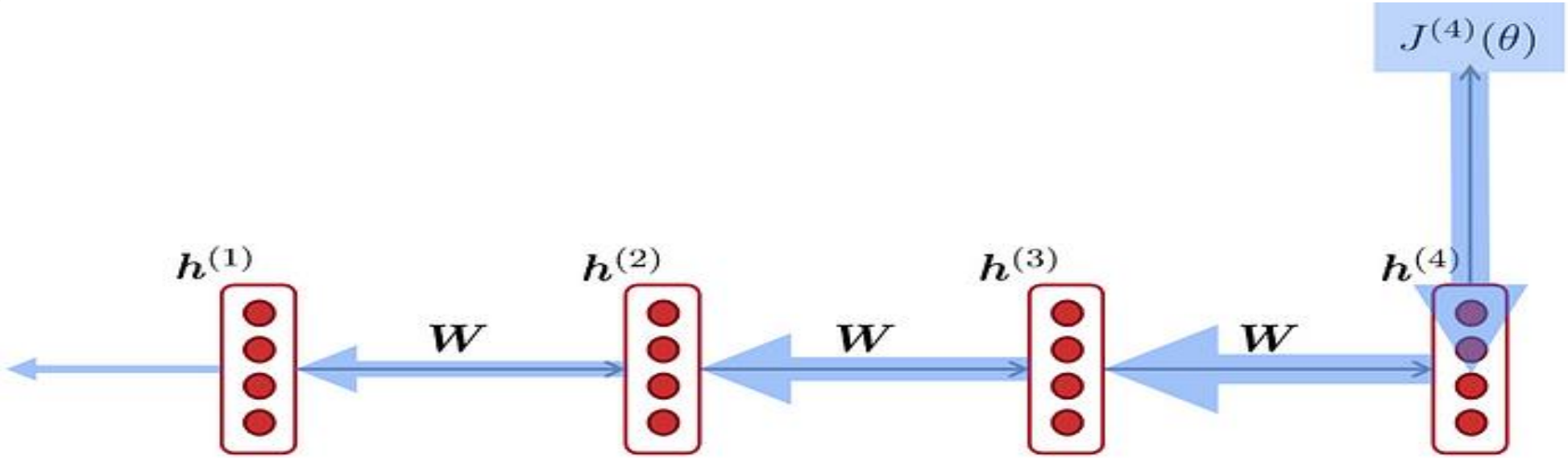
$$e^{(t)} = Ex^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



***Note**: this input sequence could be much longer, but this slide doesn't have space!*



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

LSTM

One of the architecture proposed to solve the vanishing gradient problem is called Long Short-Term Memory, and it works by having both and hidden state and a memory cell and three gates the form read (in the hidden state), write and delete of the cell. They are called the output, input, and forget gate, respectively. This architecture implements a dedicated way to maintain long-term dependencies, which is never forgetting the memory cell.

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

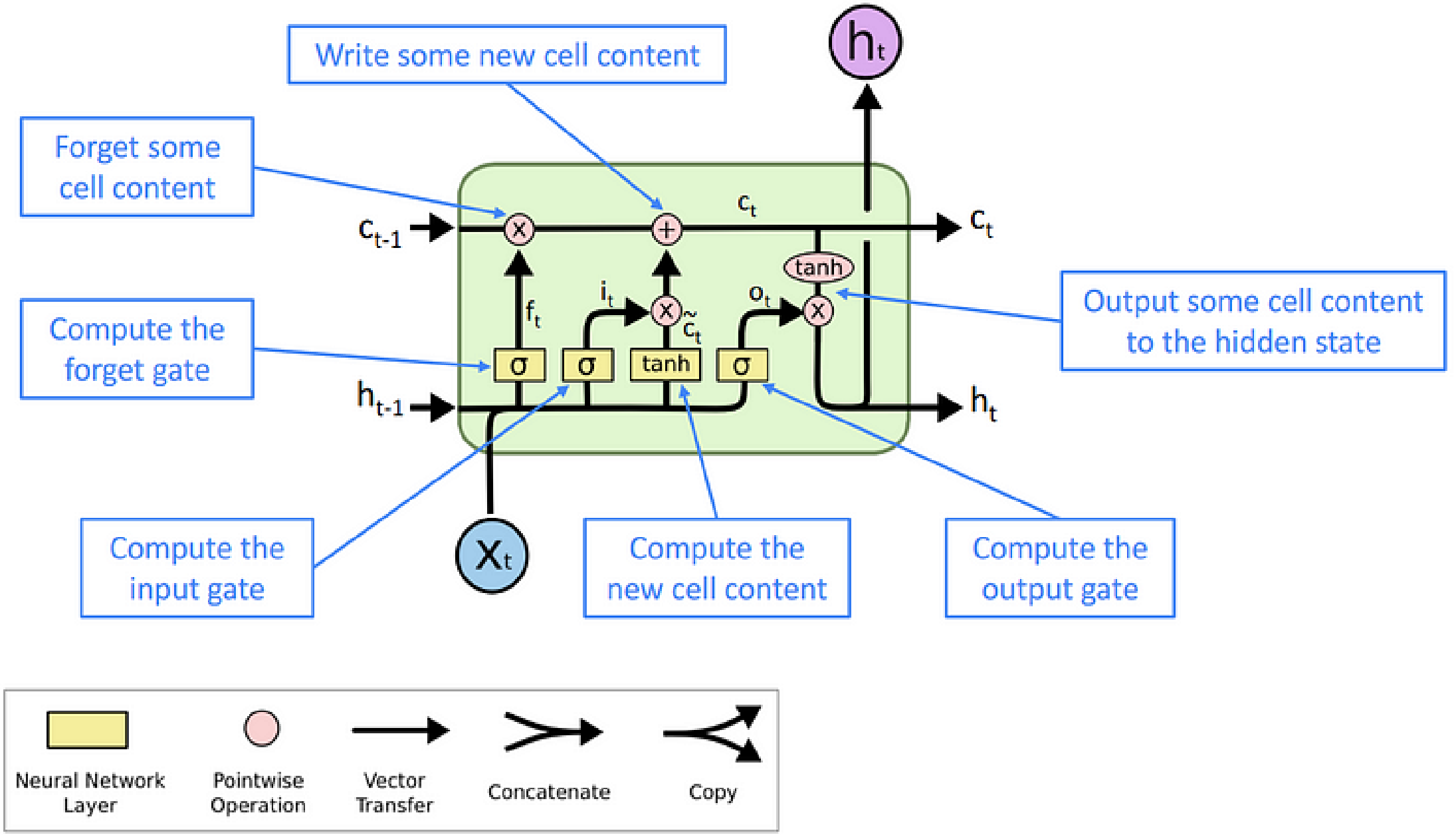
Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$
\begin{aligned}
f^{(t)} &= \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right) \\
i^{(t)} &= \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right) \\
o^{(t)} &= \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right) \\
\tilde{c}^{(t)} &= \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right) \\
c^{(t)} &= f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)} \\
h^{(t)} &= o^{(t)} \circ \tanh c^{(t)}
\end{aligned}
$$

All these are vectors of same length n

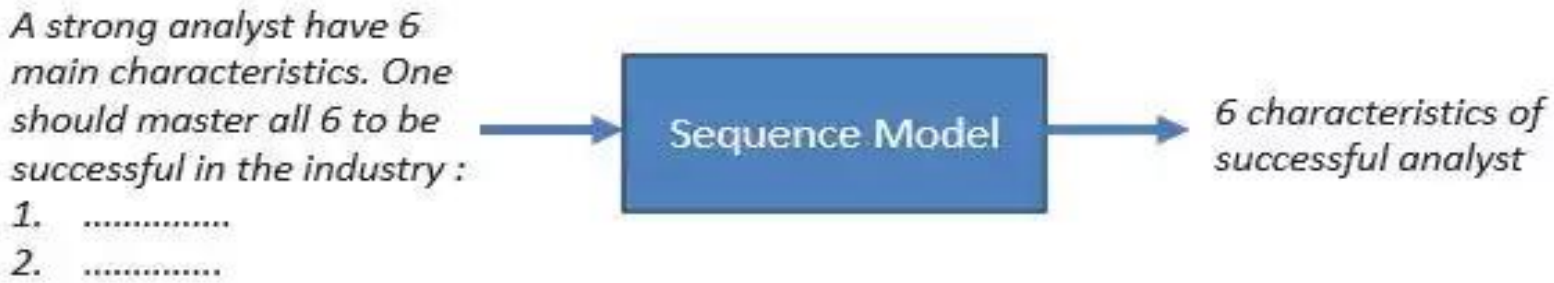
Gates are applied using element-wise product


Write some new cell content
Forget some cell content
Compute the forget gate
Compute the input gate
Compute the new cell content
Compute the output gate
Output some cell content to the hidden state
h_t
c_{t-1}
c_t
h_{t-1}
x_t
f_t
i_t
$\tilde{c}_t$
o_t
σ
tanh
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

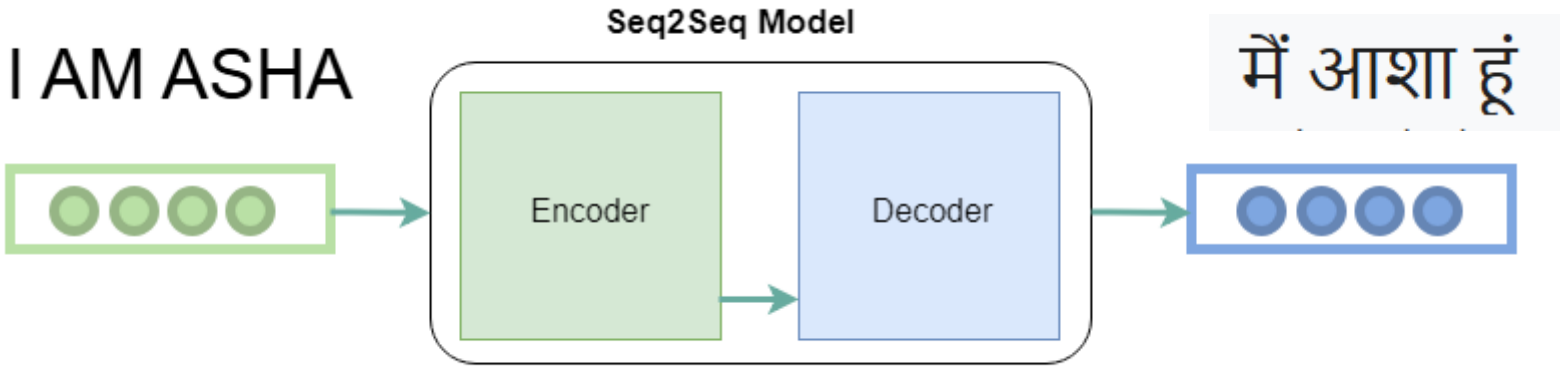
- Seq2Seq (Sequence-to-Sequence) models are a type of neural network, an exceptional Recurrent Neural Network architecture, designed to transform one data sequence into another.
- They are handy for tasks such as solving complex language problems like machine translation, question answering, creating chatbots, text summarization, etc.

Machine Language Translation

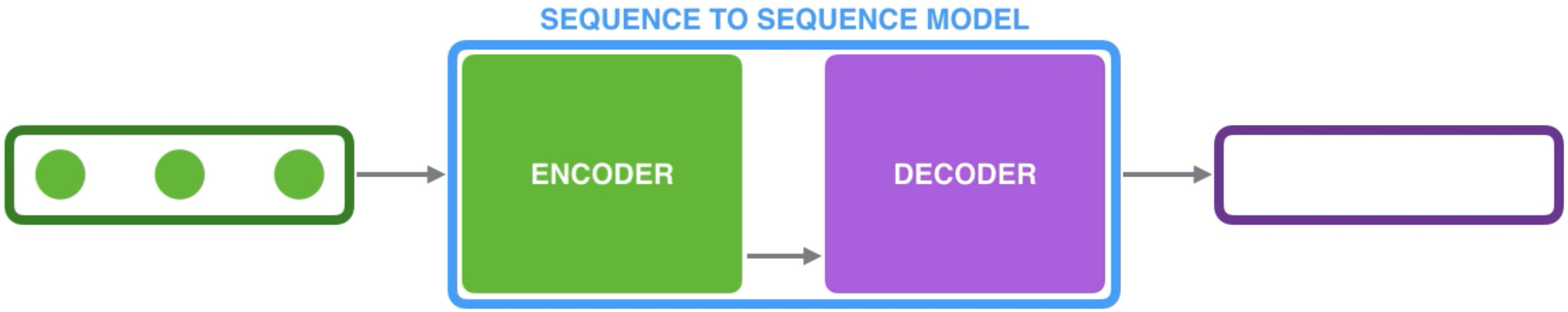
Text Summarization



Chatbot

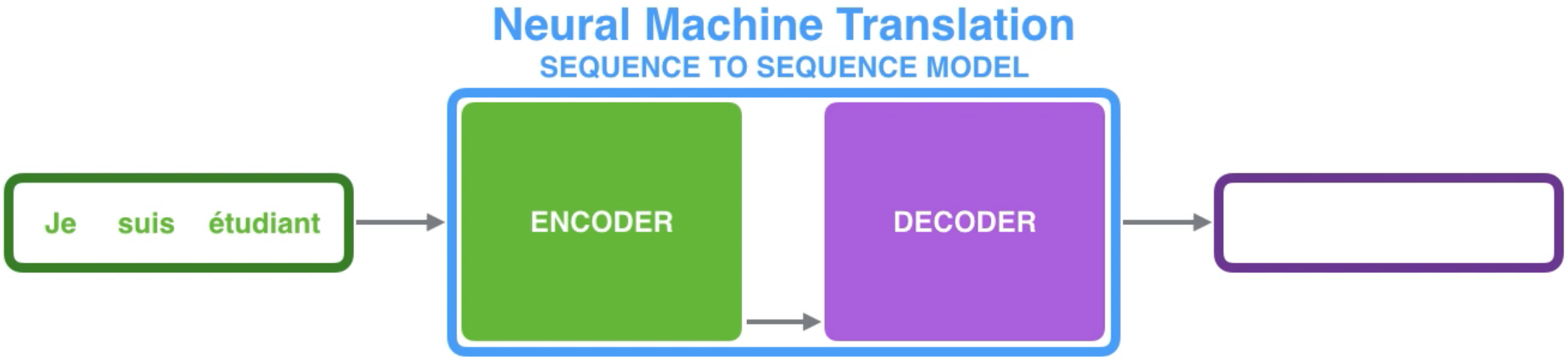


Encoder and Decoder Stack in seq2seq model

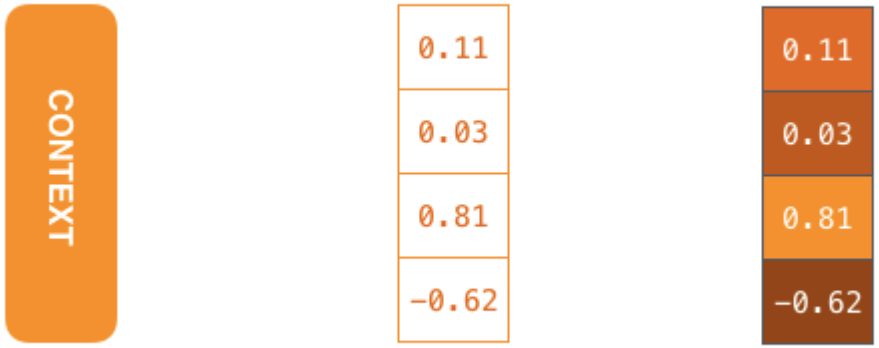


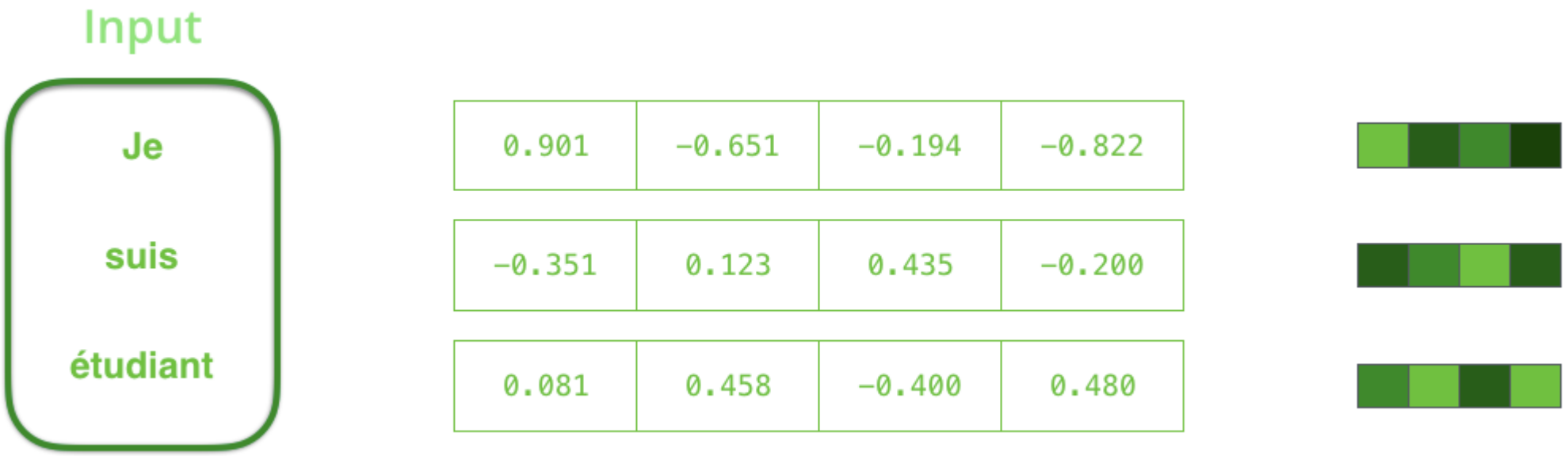
The model is composed of an encoder and a decoder.

The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context). After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item.



The context is a vector (an array of numbers, basically) in the case of machine translation. The encoder and decoder tend to both be recurrent neural networks

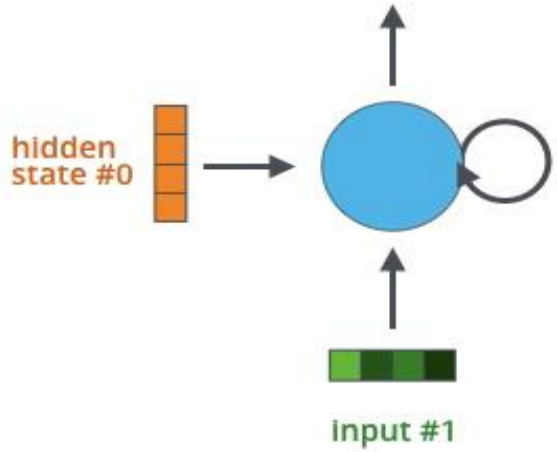

CONTEXT
0.11
0.03
0.81
−0.62
0.11
0.03
0.81
−0.62



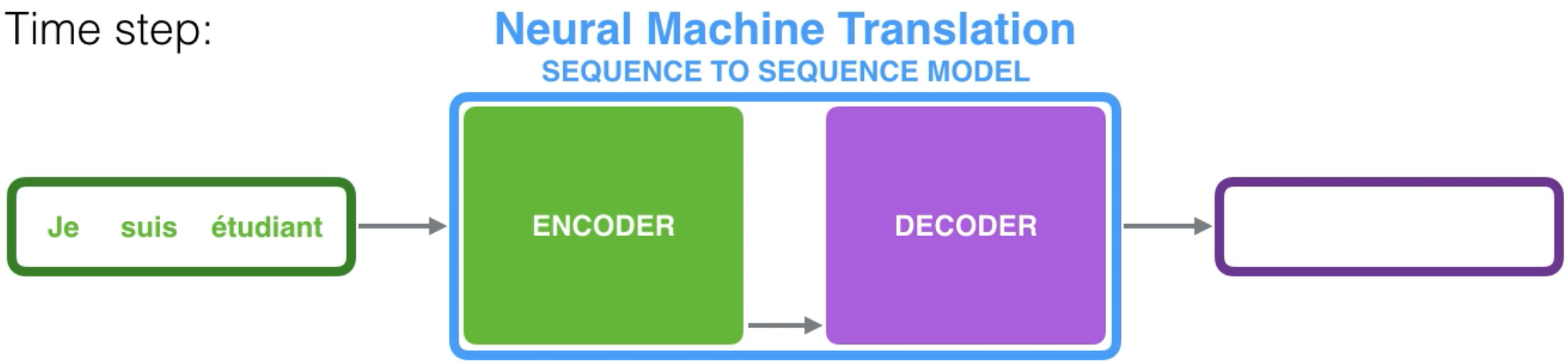
By design, a RNN takes two inputs at each time step: an input (in the case of the encoder, one word from the input sentence), and a hidden state. The word, however, needs to be represented by a vector. To transform a word into a vector, we turn to the class of methods called “word embedding” algorithms. These turn words into vector spaces that capture a lot of the meaning/semantic information of the words

Recurrent Neural Network

Time step #1:
An RNN takes two input vectors:



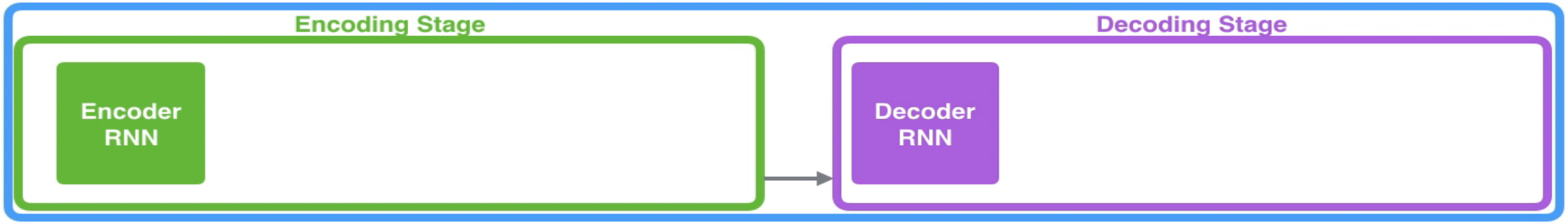
Each pulse for the encoder or decoder is that RNN processing its inputs and generating an output for that time step.
Since the encoder and decoder are both RNNs, each time step one of the RNNs does some processing, it updates its hidden state based on its inputs and previous inputs it has seen.



When we look at the hidden states for the encoder. Notice how the last hidden state is actually the context we pass along to the decoder.

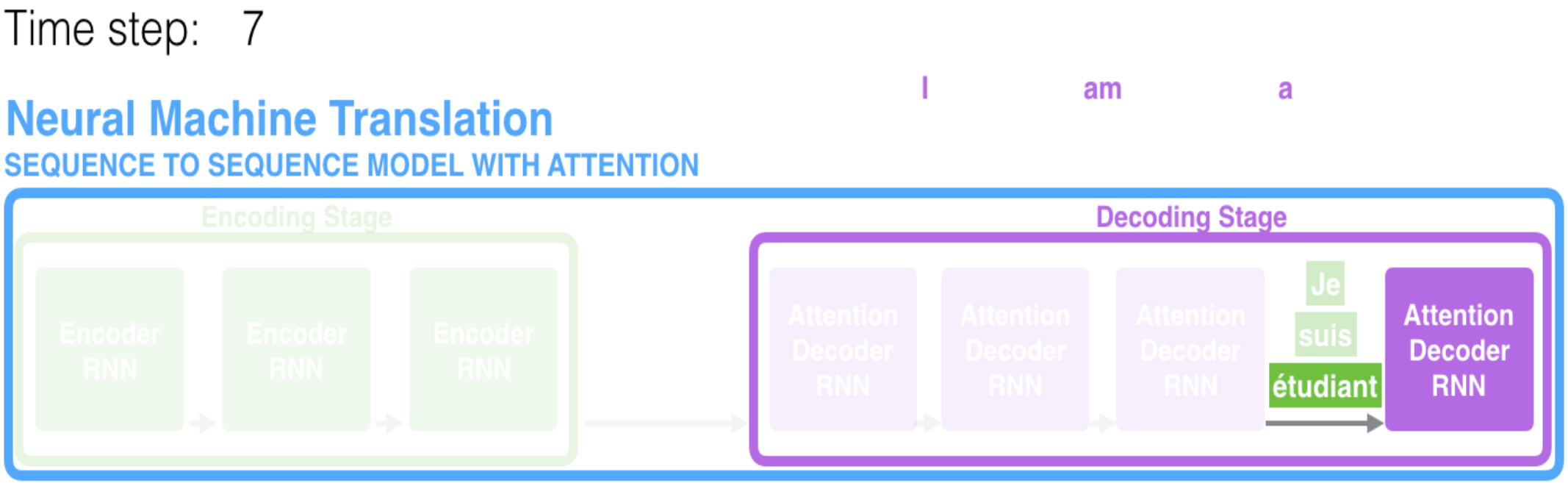
Neural Machine Translation

SEQUENCE TO SEQUENCE MODEL



This is called an “unrolled” view where instead of showing the one decoder, we show a copy of it for each time step as we can look at the inputs and outputs of each time step.

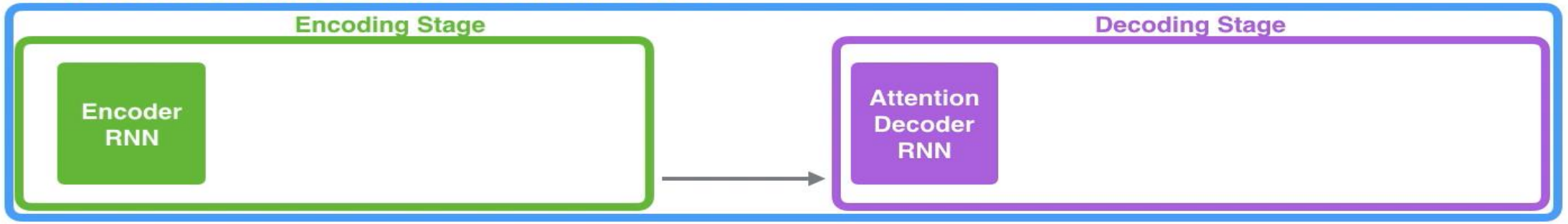
The context vector turned out to be a bottleneck for these types of models. It made it challenging for the models to deal with long sentences. A solution was proposed in Bahdanau et al., 2014 and Luong et al., 2015. These papers introduced and refined a technique called “Attention”, which highly improved the quality of machine translation systems. Attention allows the model to focus on the relevant parts of the input sequence as needed.



At time step 7, the attention mechanism enables the decoder to focus on the word "étudiant" ("student" in french) before it generates the English translation. This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without attention.

Neural Machine Translation

SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



Je suis étudiant

An attention model differs from a classic sequence-to-sequence model in two main ways:

1) First, the encoder passes a lot more data to the decoder. Instead of passing the last hidden state of the encoding stage, the encoder passes *all* the hidden states to the decoder

2) Second, an attention decoder does an extra step before producing its output. In order to focus on the parts of the input that are relevant to this decoding time step, the decoder does the following:

- Look at the set of encoder hidden states it received – each encoder hidden state is most associated with a certain word in the input sentence
- Give each hidden state a score (let's ignore how the scoring is done for now)
- Multiply each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Kmit

Attention at time step 4

1.The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.

2.The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.

3.Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.

4.We concatenate h4 and C4 into one vector.

5.We pass this vector through a feedforward neural network (one trained jointly with the model).

6.The output of the feedforward neural networks indicates the output word of this time step.

7.Repeat for the next time steps

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

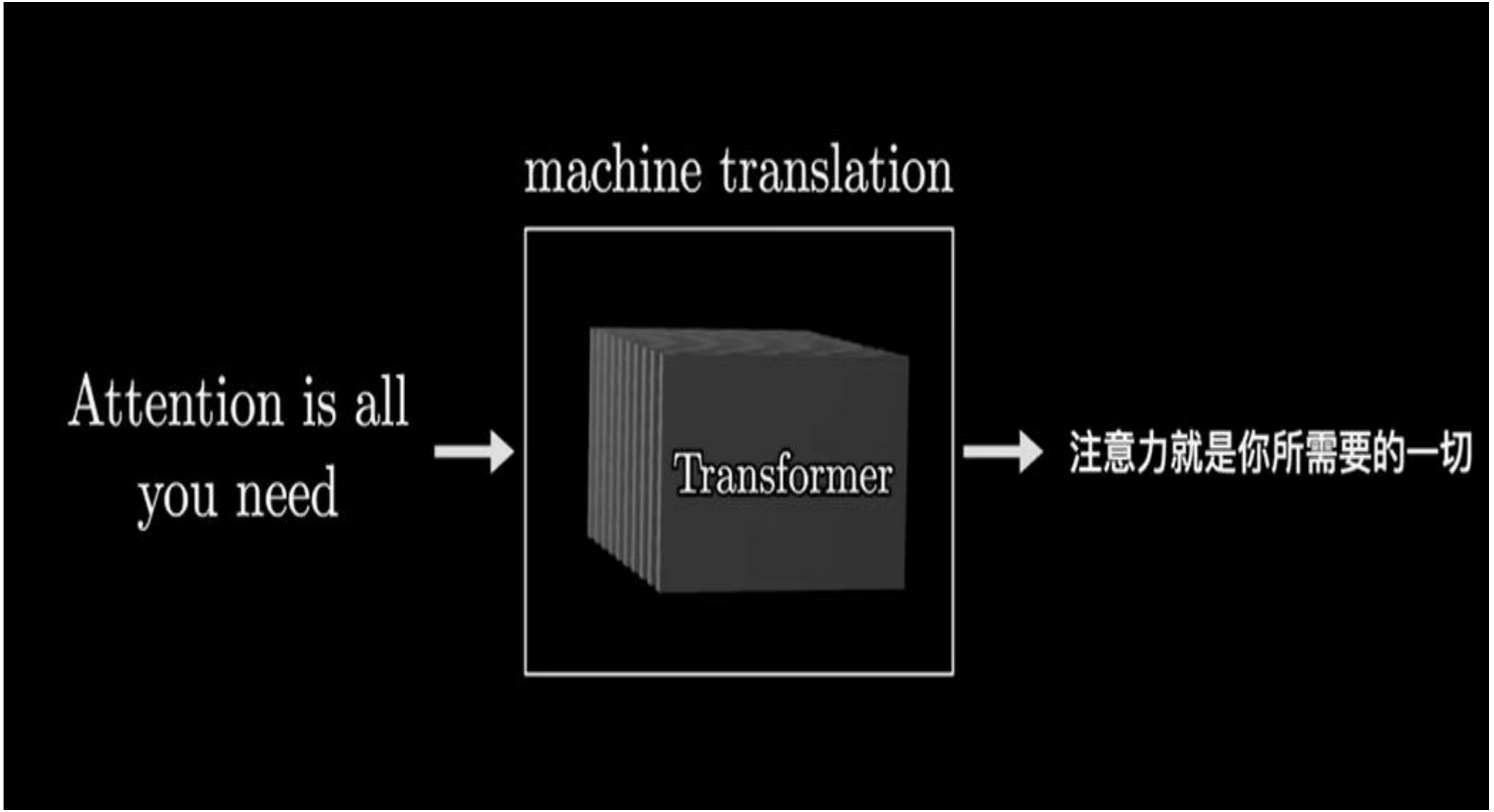
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

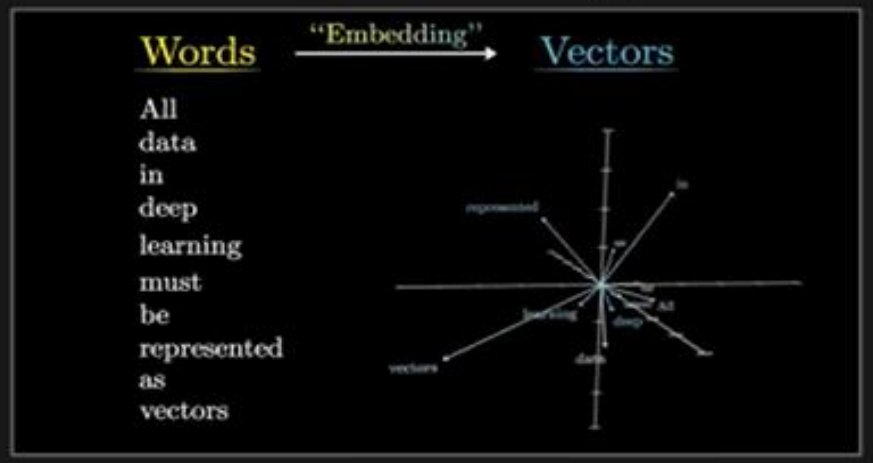
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

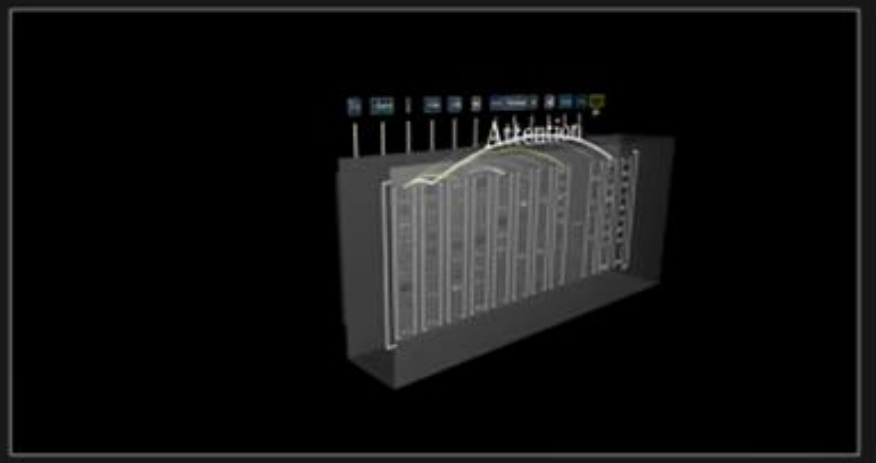
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.


machine translation
Attention is all you need
Transformer
注意力就是你所需要的一切

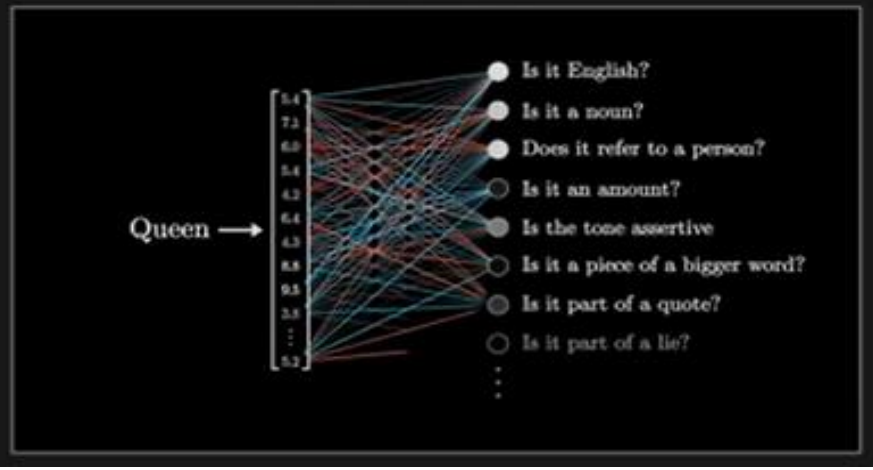
Embedding



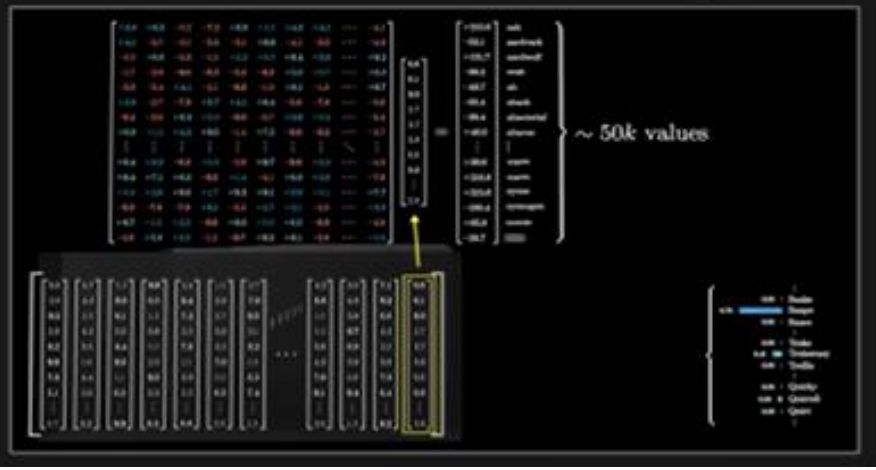
Attention

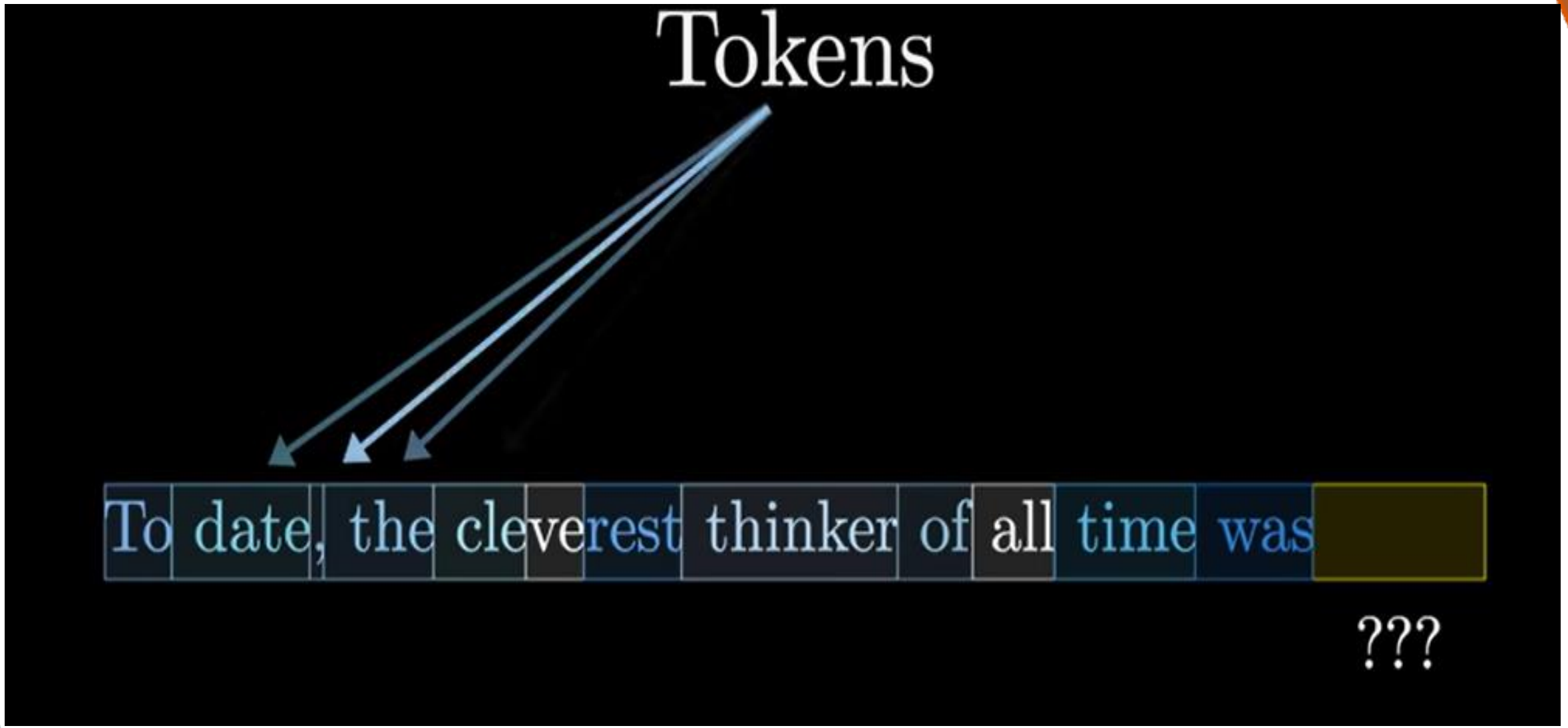


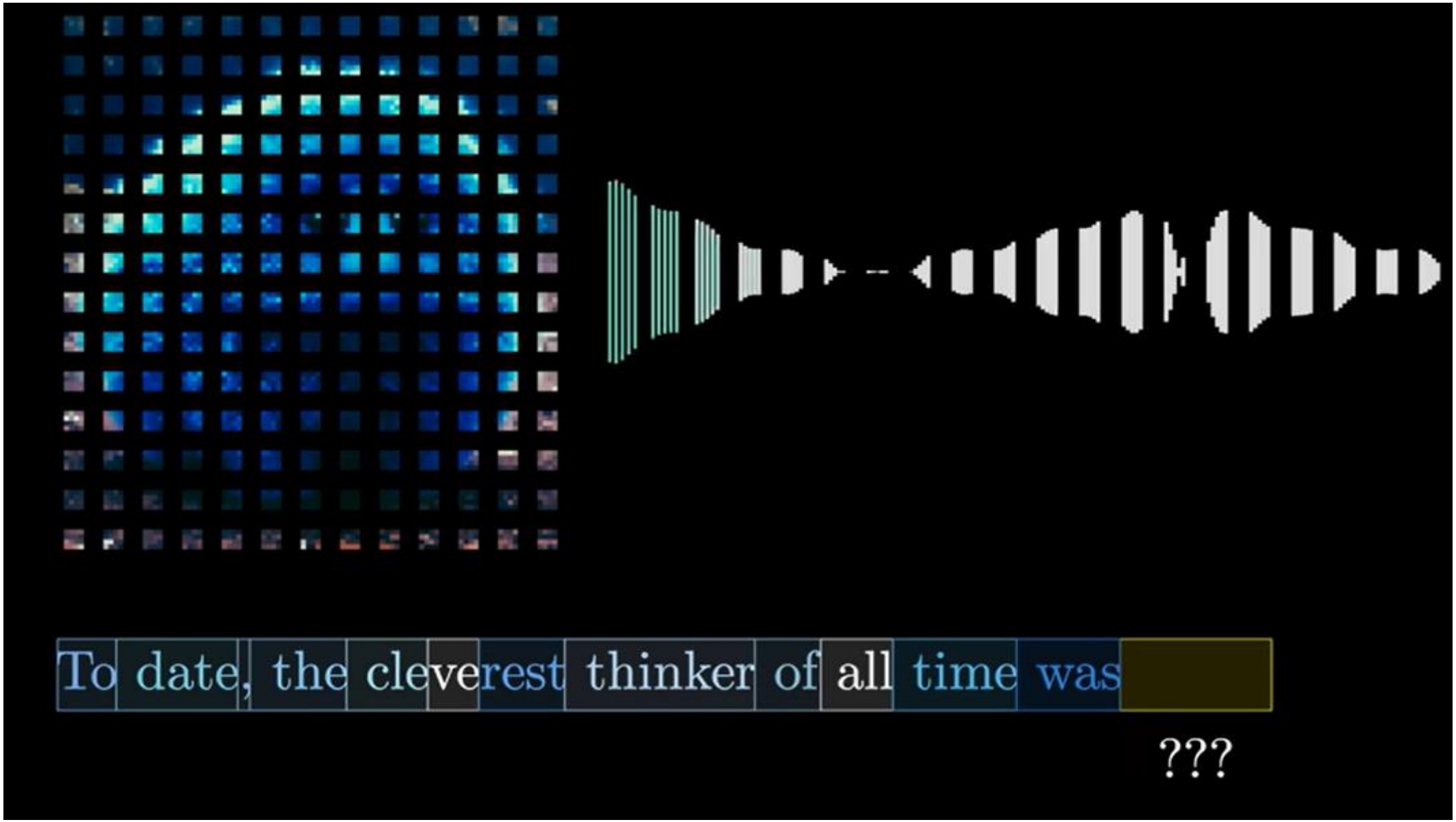
MLPs

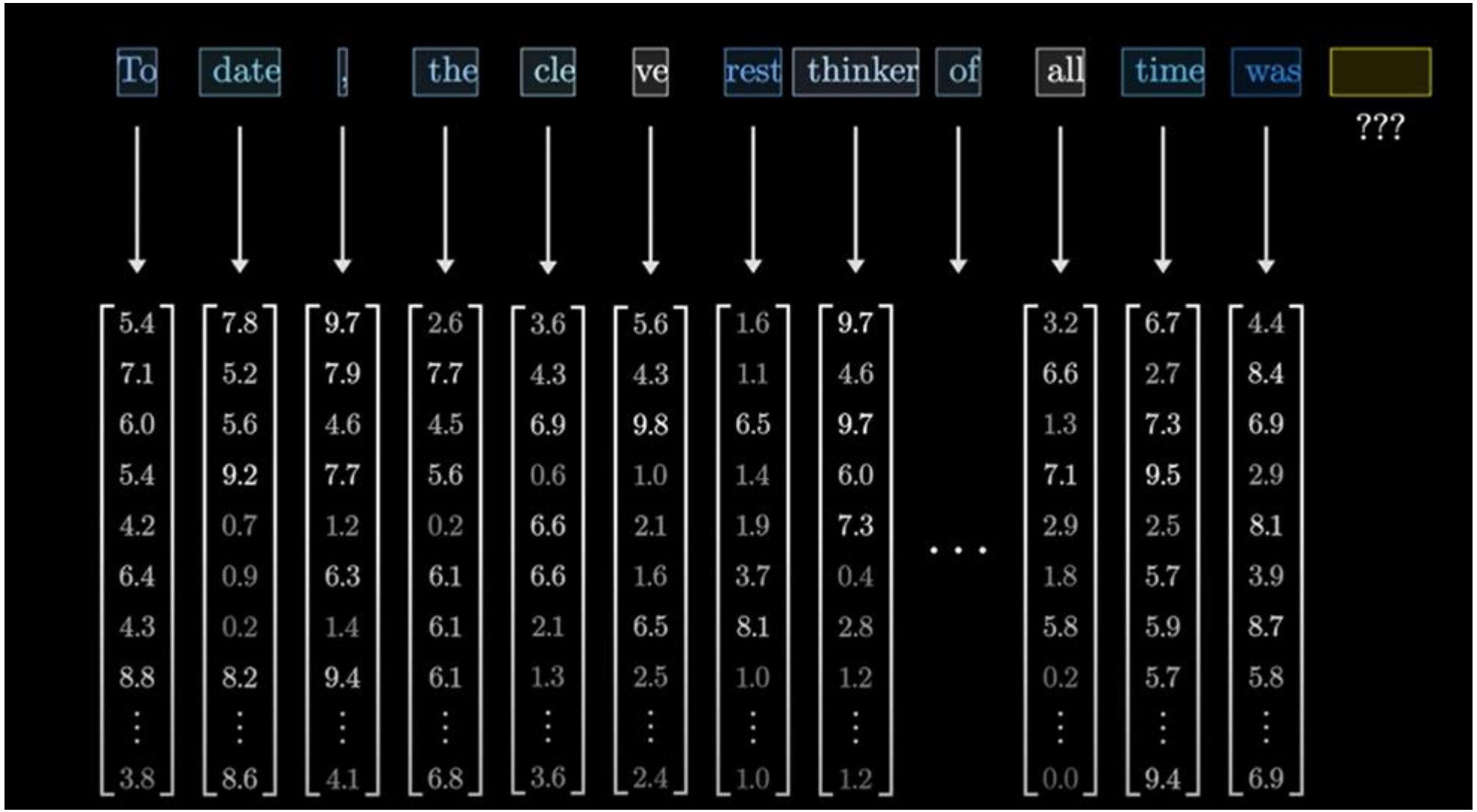


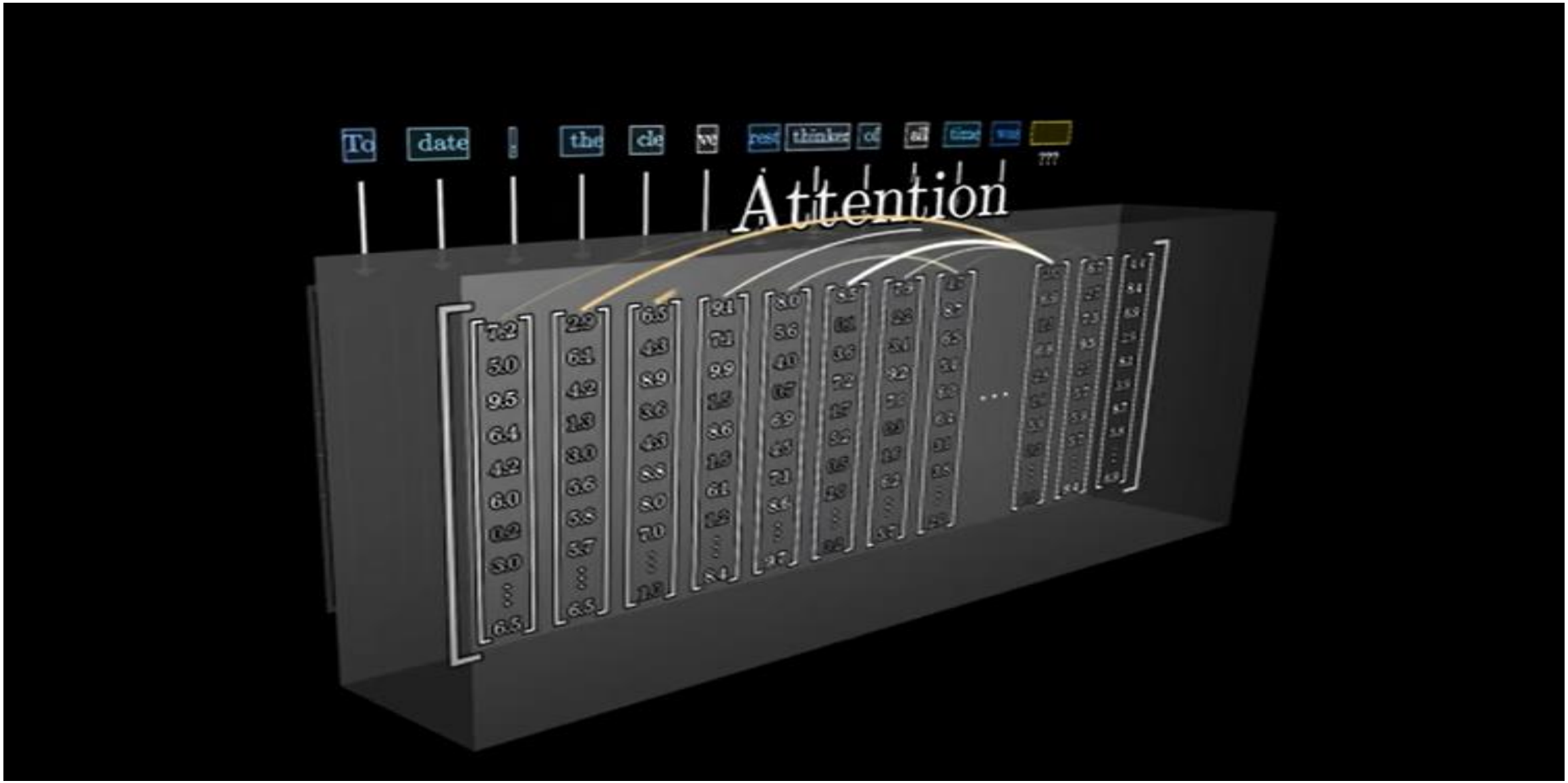
Unembedding

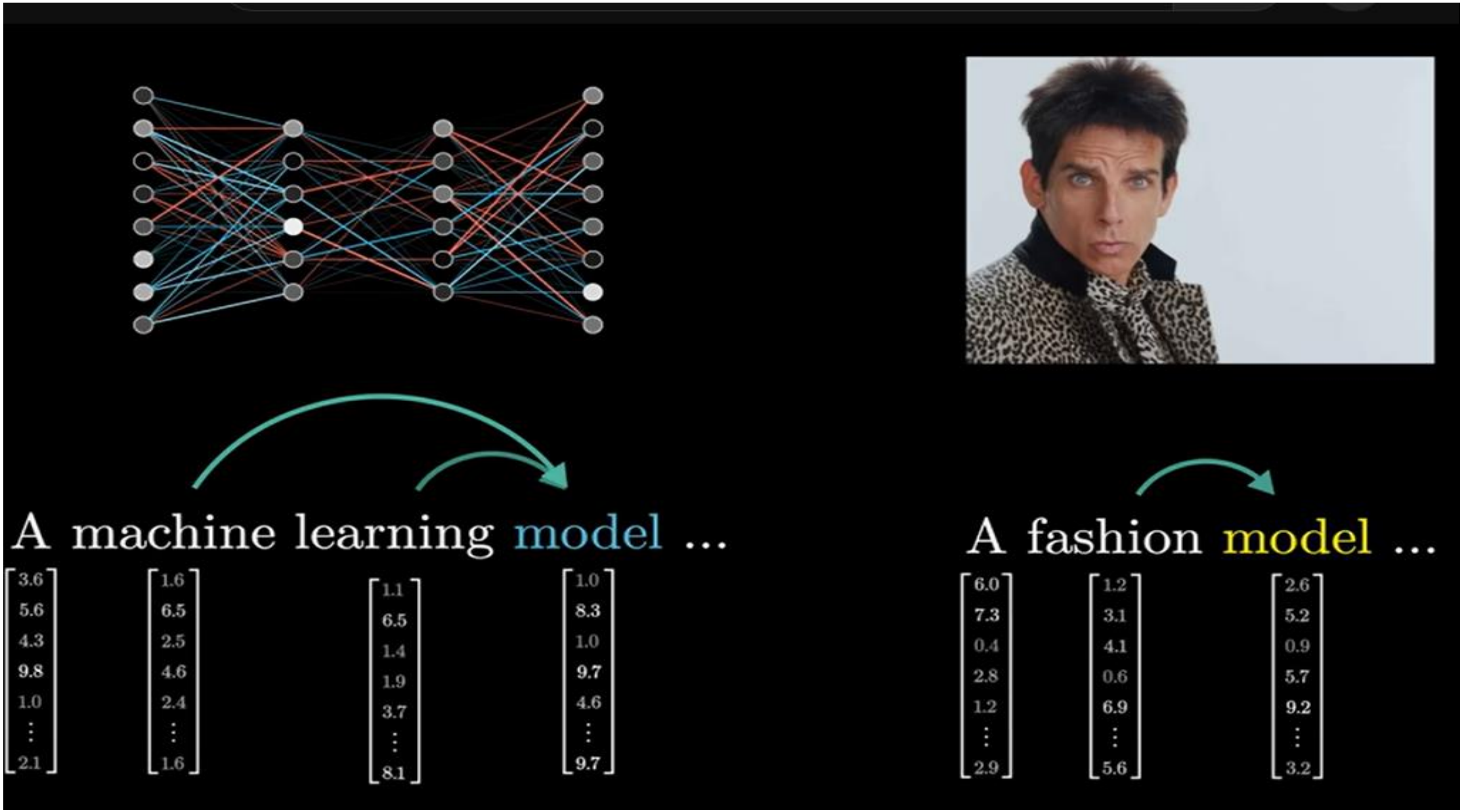


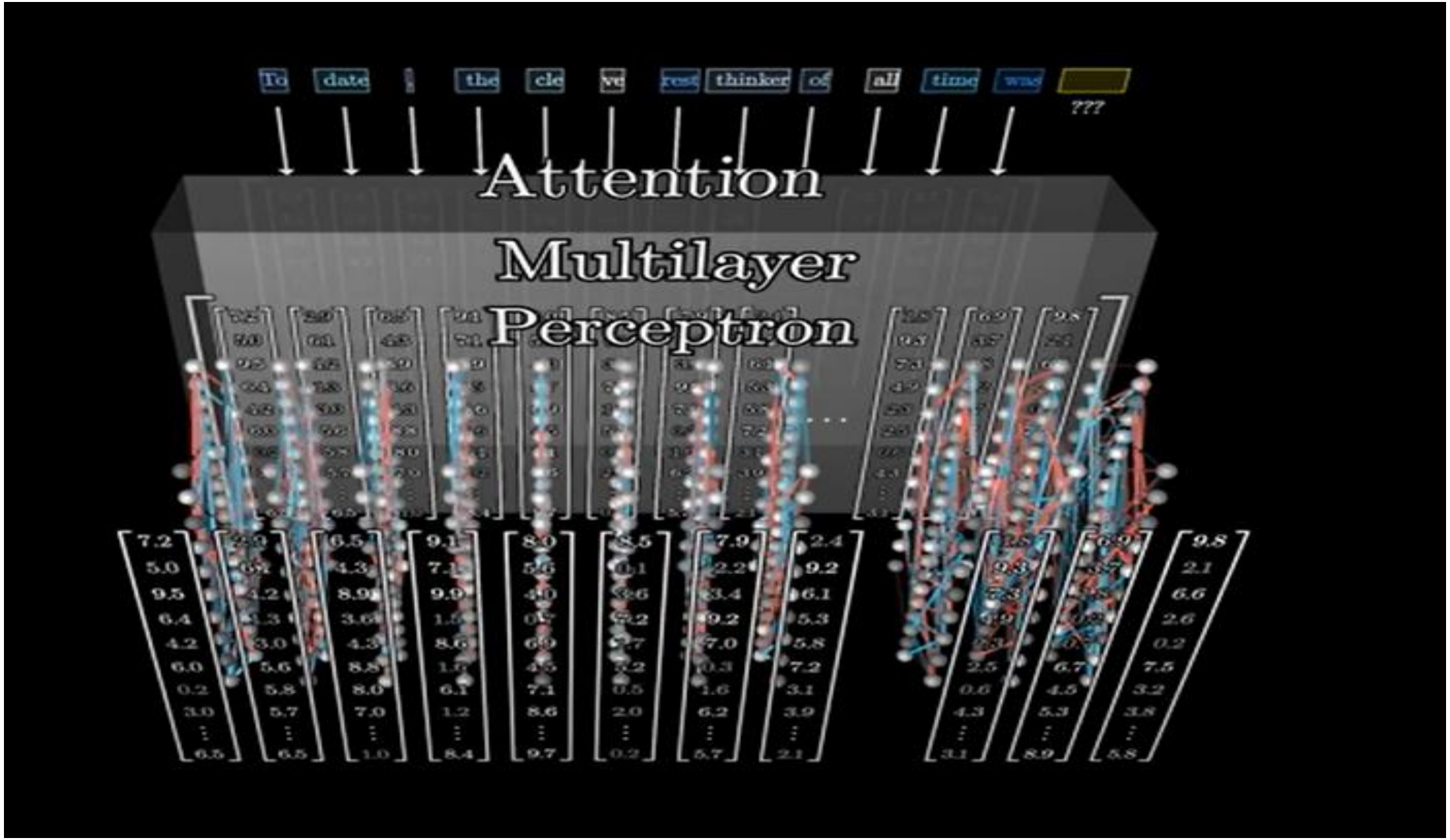

Tokens
To date, the cleverest thinker of all time was
???

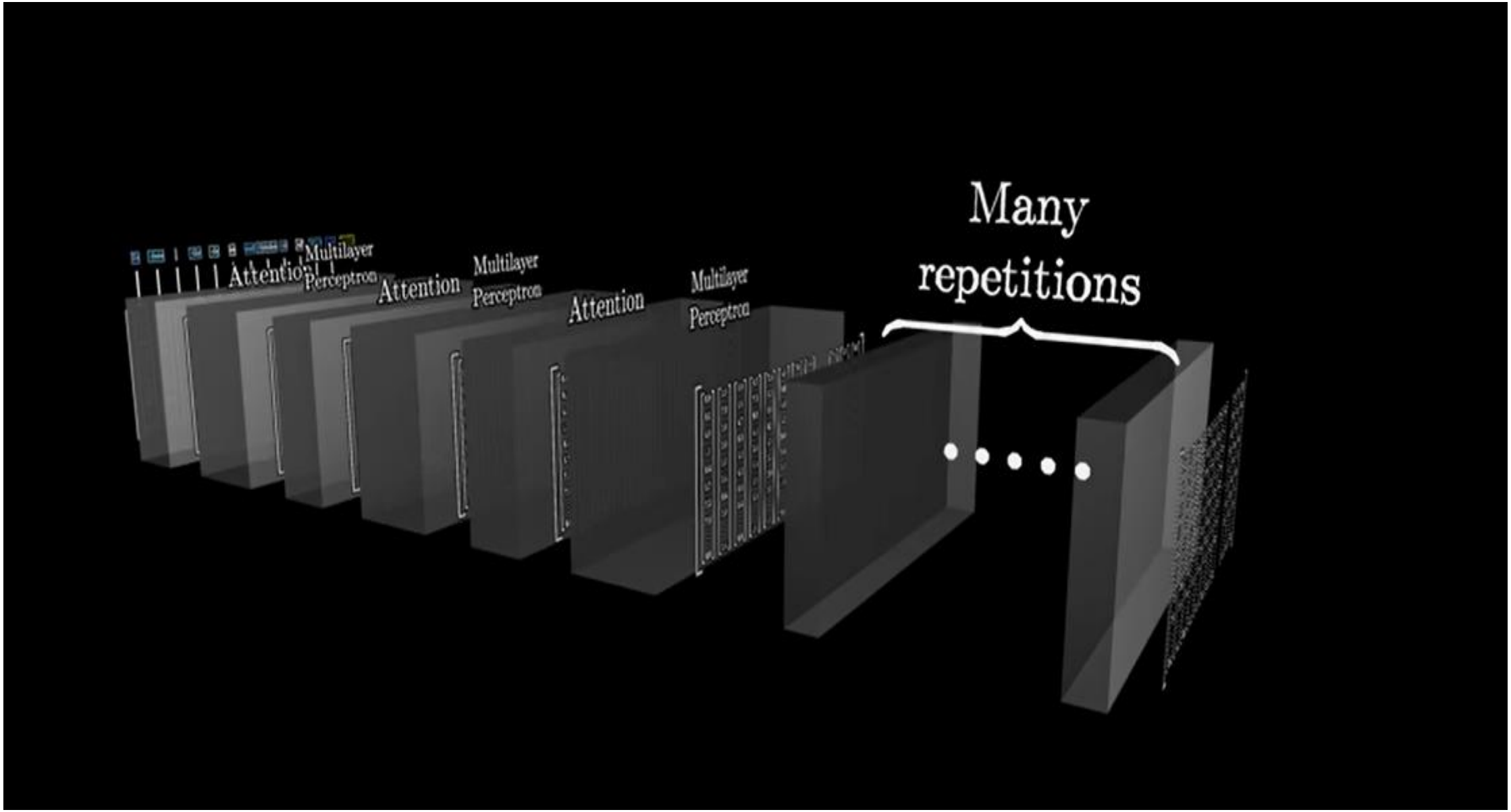

To date, the cleverest thinker of all time was
???


To
date
!
the
cle
ve
rest
thinker
of
all
time
was
???
5.4 7.8 9.7 2.6 3.6 5.6 1.6 9.7 3.2 6.7 4.4
7.1 5.2 7.9 7.7 4.3 4.3 1.1 4.6 6.6 2.7 8.4
6.0 5.6 4.6 4.5 6.9 9.8 6.5 9.7 1.3 7.3 6.9
5.4 9.2 7.7 5.6 0.6 1.0 1.4 6.0 7.1 9.5 2.9
4.2 0.7 1.2 0.2 6.6 2.1 1.9 7.3 2.9 2.5 8.1
6.4 0.9 6.3 6.1 6.6 1.6 3.7 0.4 1.8 5.7 3.9
4.3 0.2 1.4 6.1 2.1 6.5 8.1 2.8 5.8 5.9 8.7
8.8 8.2 9.4 6.1 1.3 2.5 1.0 1.2 0.2 5.7 5.8
3.8 8.6 4.1 6.8 3.6 2.4 1.0 1.2 0.0 9.4 6.9


To
date
the
???
Attention


A machine learning model ...
A fashion model ...


To
date
the
thinker
of
all
time
???
Attention
Multilayer
Perceptron


Attention
Multilayer Perceptron
Attention
Multilayer Perceptron
Attention
Multilayer Perceptron
Many repetitions

Attention Mechanism

Assign attention weight to each word, to know how much "attention" the model should pay to each word (i.e., for each word, the network learns a "context")

Originally developed for language translation:
Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

Hidden state in a regular RNN (RNN #2)

1st translated word

Attention weight

Bidirectional RNN (RNN #1)

1st input word

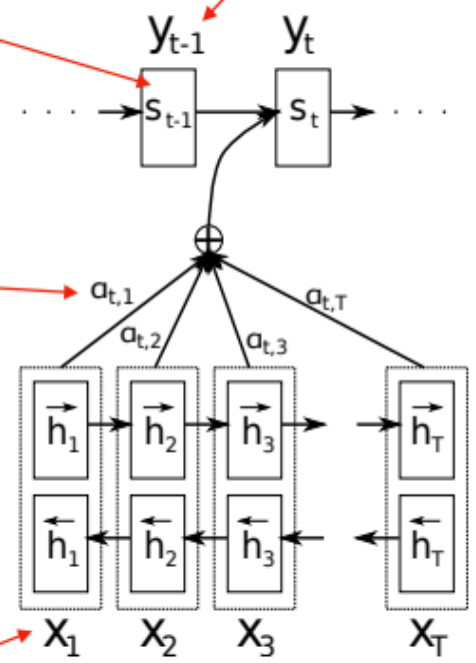
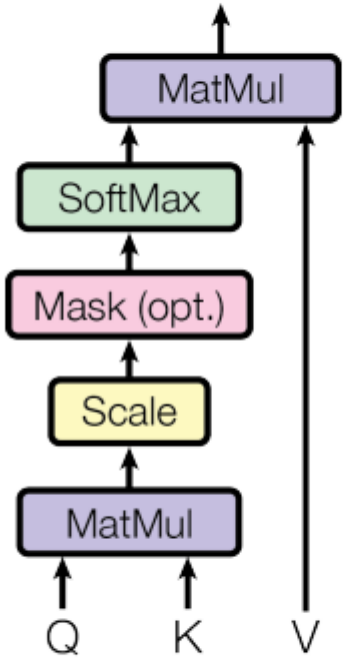


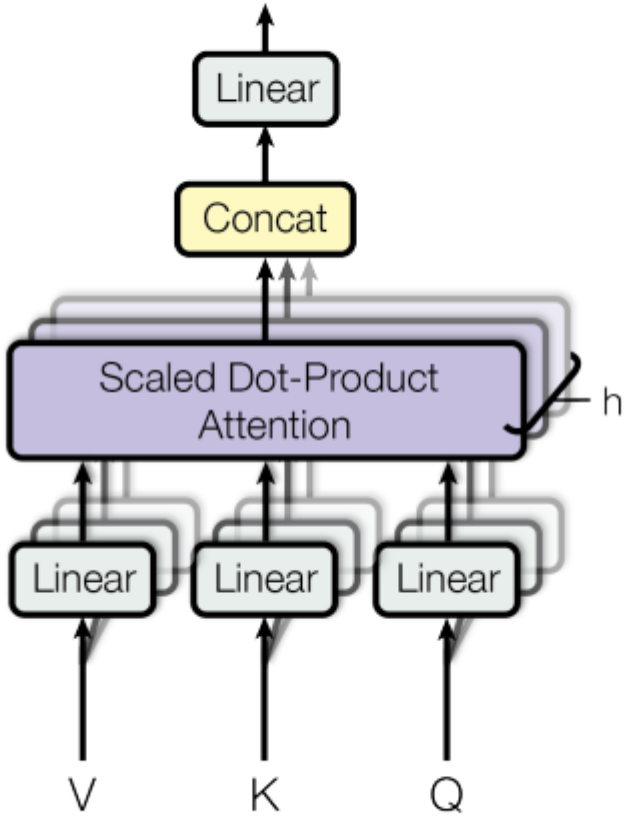
Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

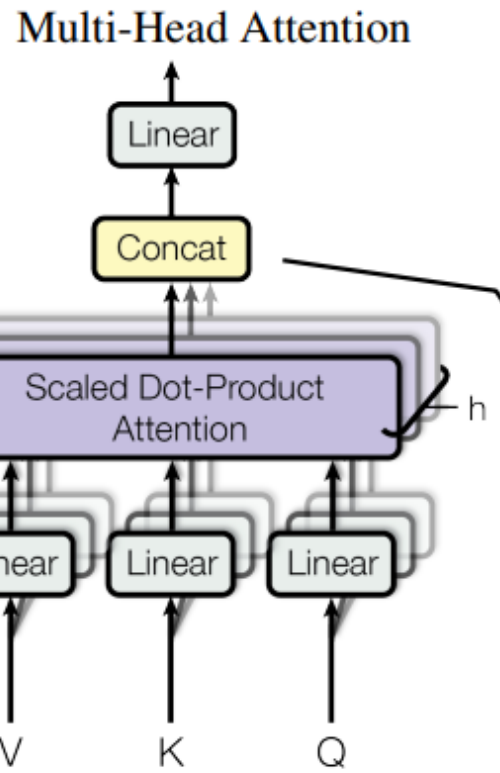
Kmit

Scaled Dot-Product Attention



Multi-Head Attention





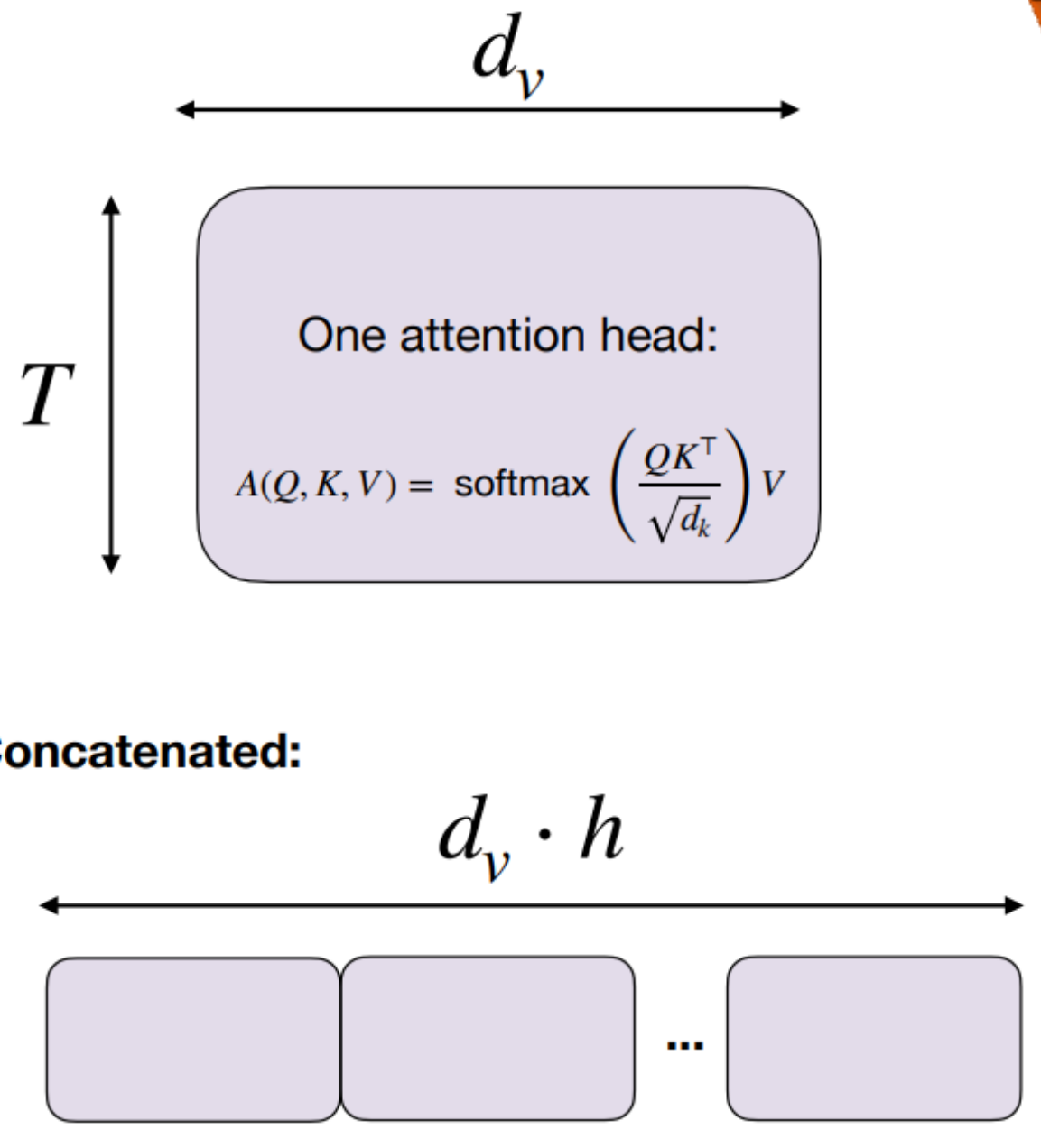
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Input sequence dim.
in original transformer:

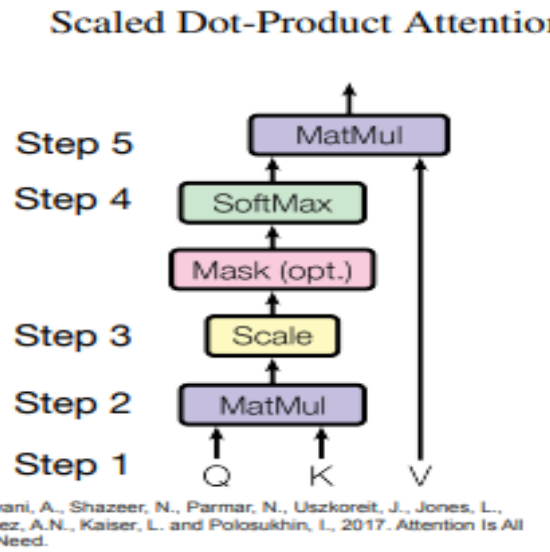
$$T \times d_e = T \times 512$$

and

$$d_v = 512/h = 64$$

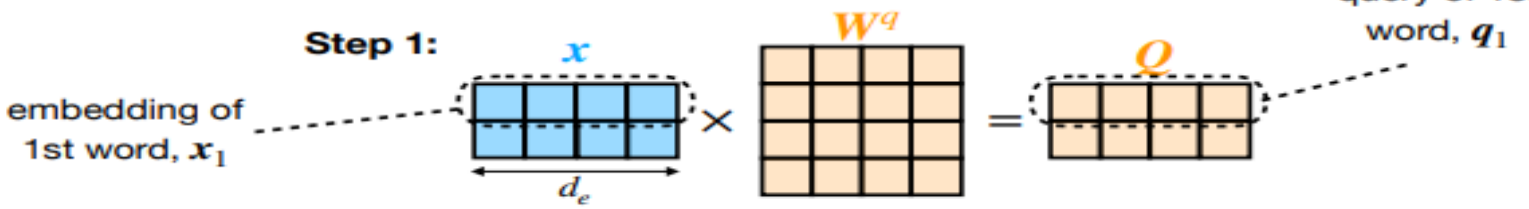


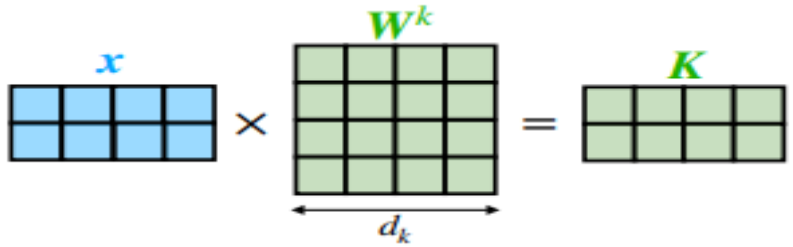
Scaled Dot-Product Attention Recap

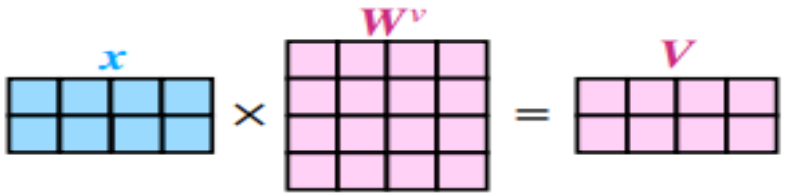


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

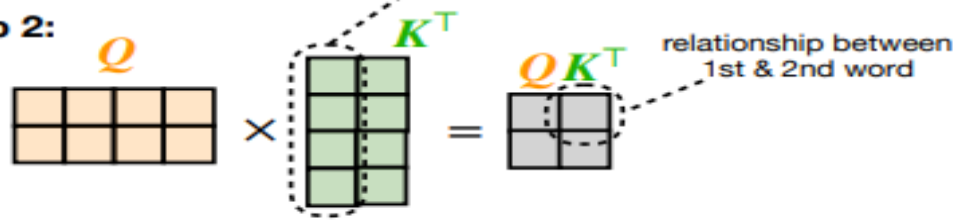
Step 1:



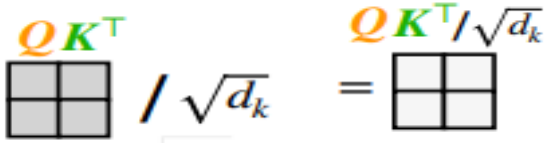




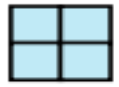
Step 2:



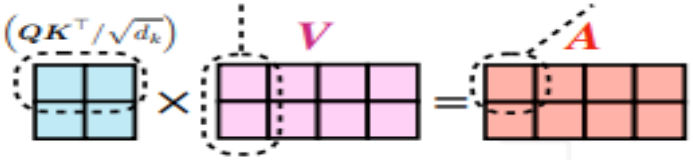
Step 3:

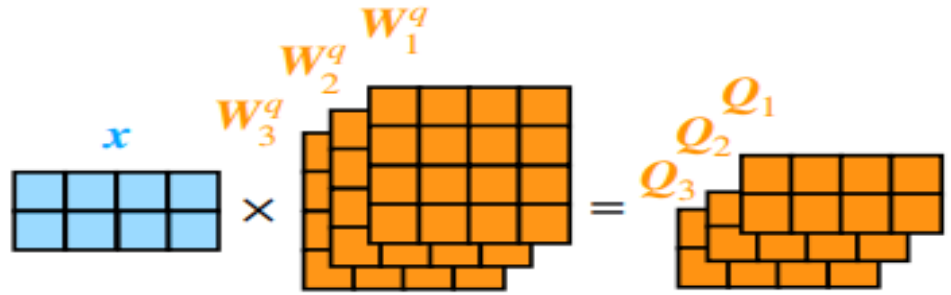


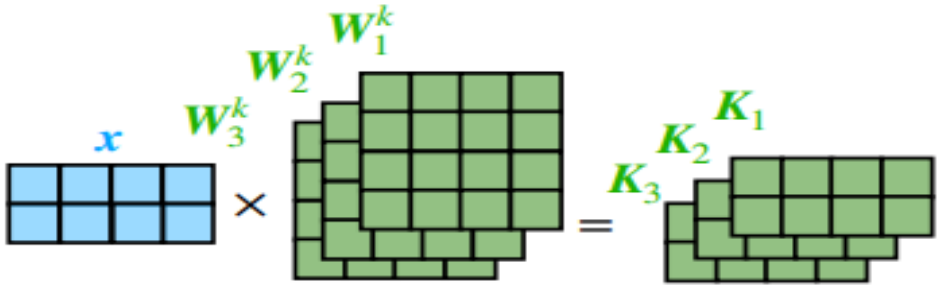
Step 4: $\text{softmax}\left(QK^\top/\sqrt{d_k}\right) =$

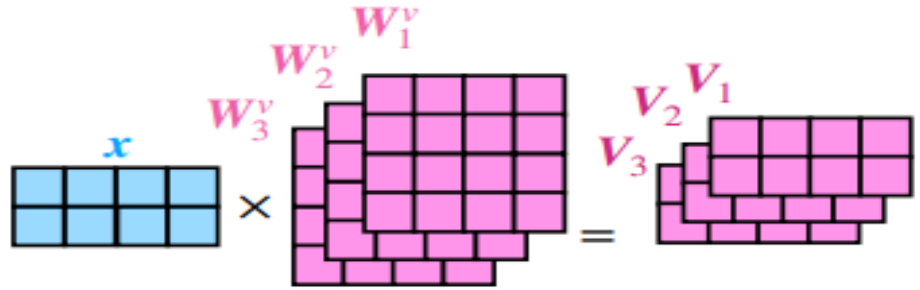


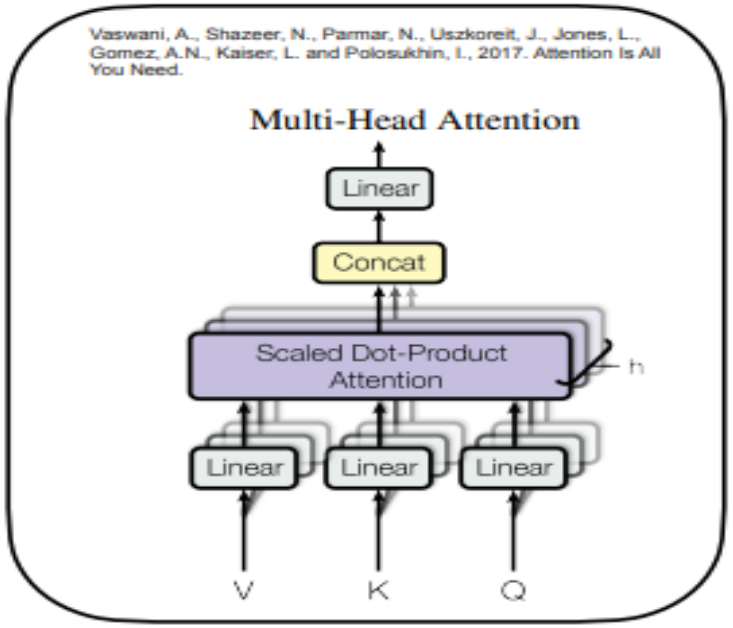
Step 5:

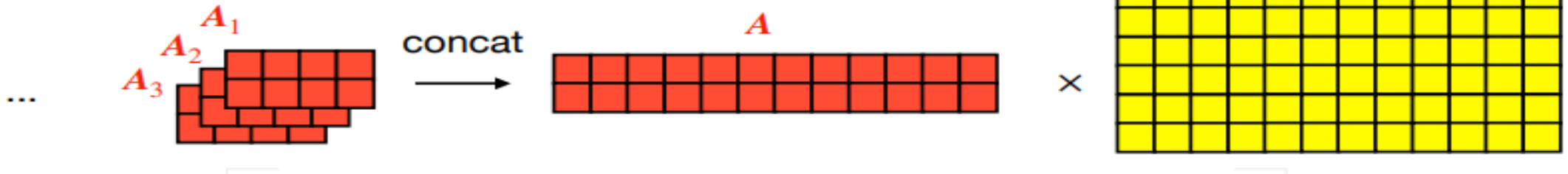



x
W_1^q
W_2^q
W_3^q
Q_1
Q_2
Q_3

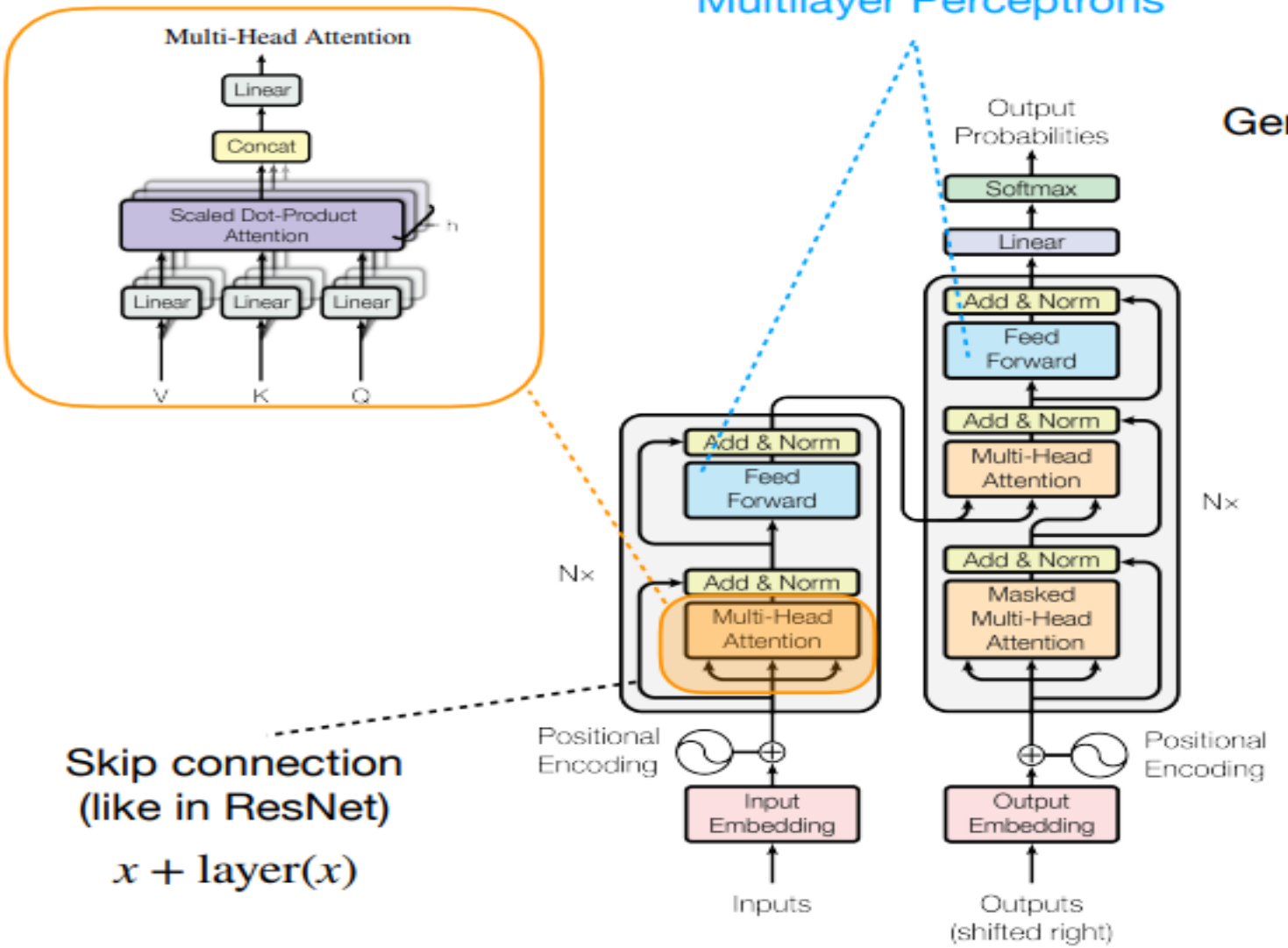

x
W_1^k
W_2^k
W_3^k
K_1
K_2
K_3


x
W_1^v
W_2^v
W_3^v
V_1
V_2
V_3


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q


...
A_1
A_2
A_3
concat
A
W_o

Multilayer Perceptrons



Generates output words one at a time

Skip connection (like in ResNet)

$$x + \text{layer}(x)$$

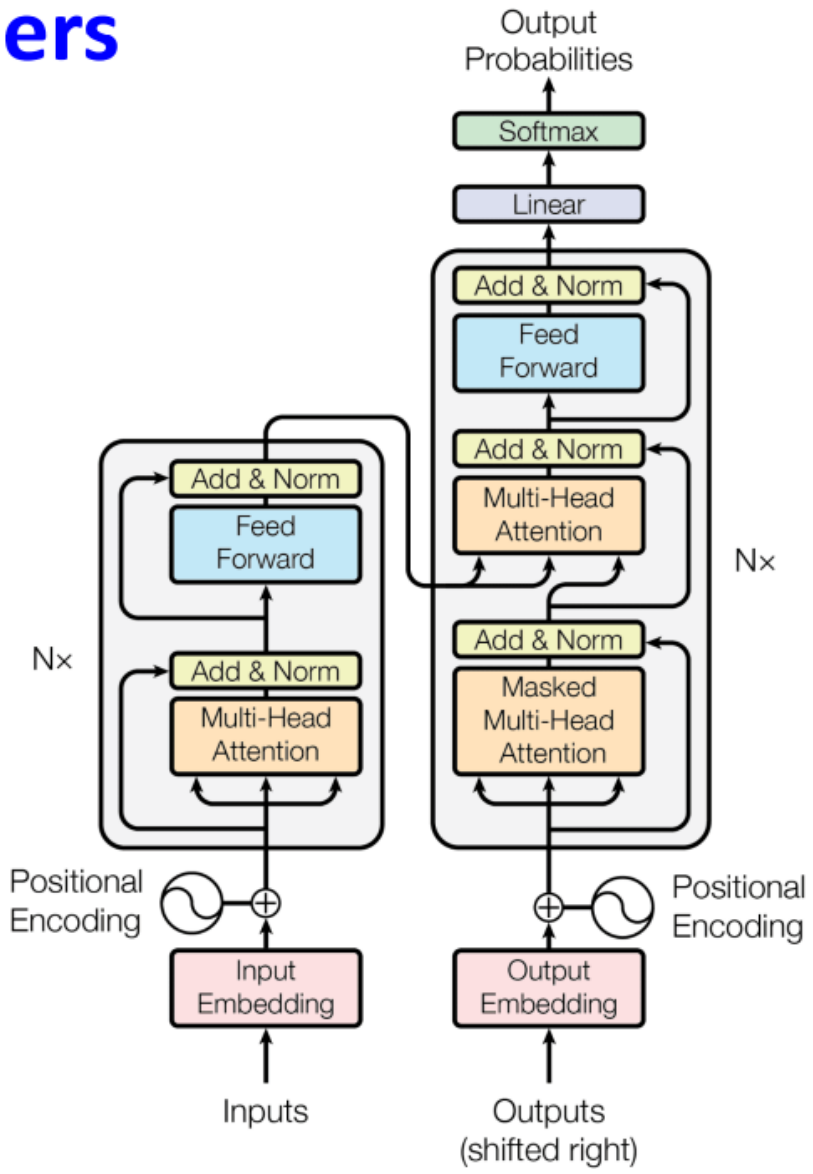
Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Transformers

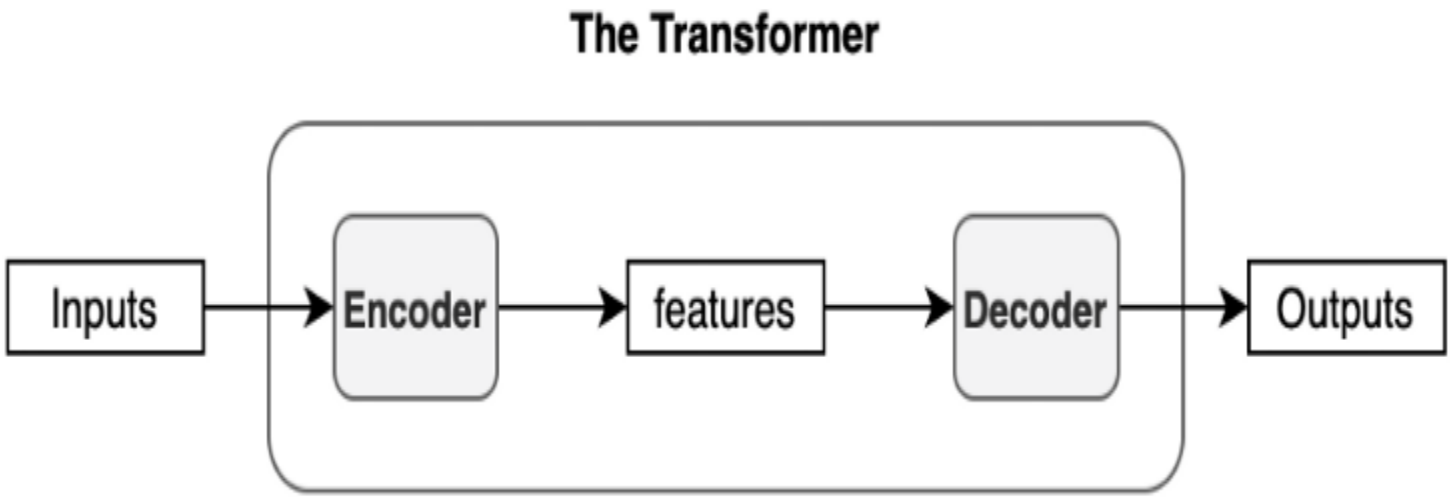
- Tokenizaton
- Input Embeddings
- Position Encodings
- Residuals
- Query
- Key
- Value
- Add & Norm
- Encoder
- Decoder

- Attention
- Self Attention
- Multi Head Attention
- Masked Attention
- Encoder Decoder Attention
- Output Probabilities / Logits
- Softmax
- Encoder-Decoder models
- Decoder only models

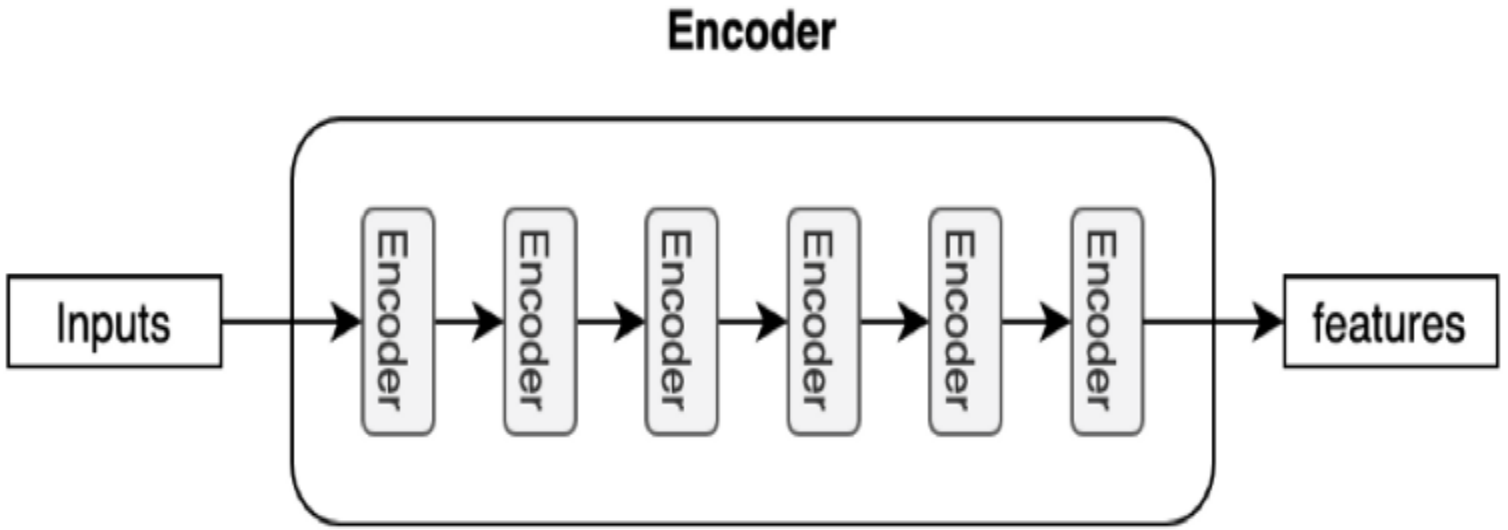


The original transformer published in the paper is a neural machine translation model. For example, we can train it to translate English into French sentences.

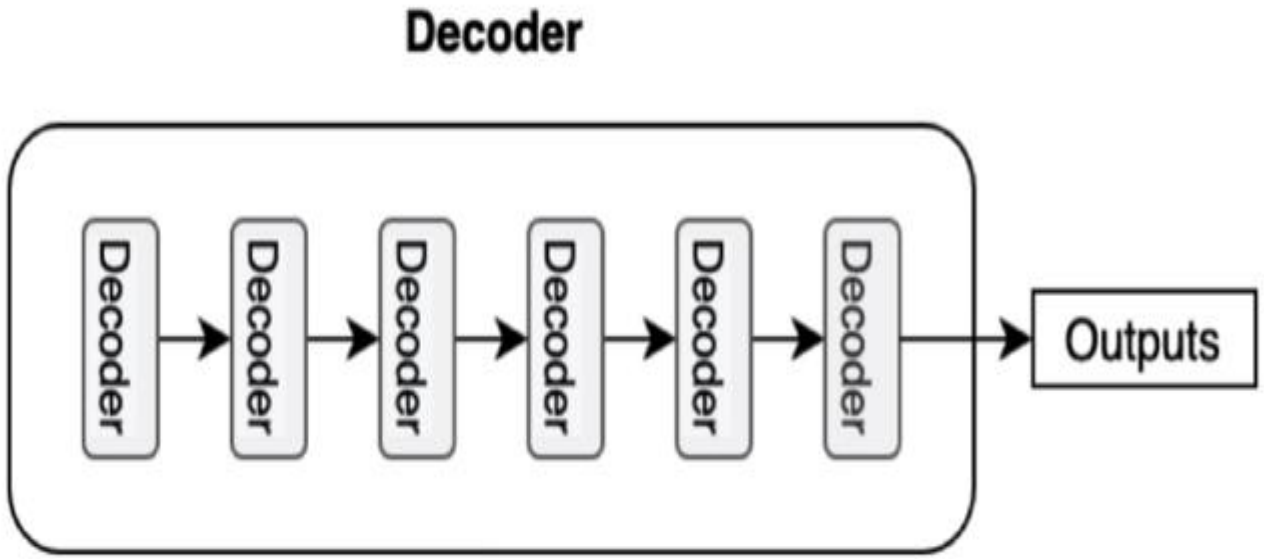
The transformer uses an encoder-decoder architecture. The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence (translation).



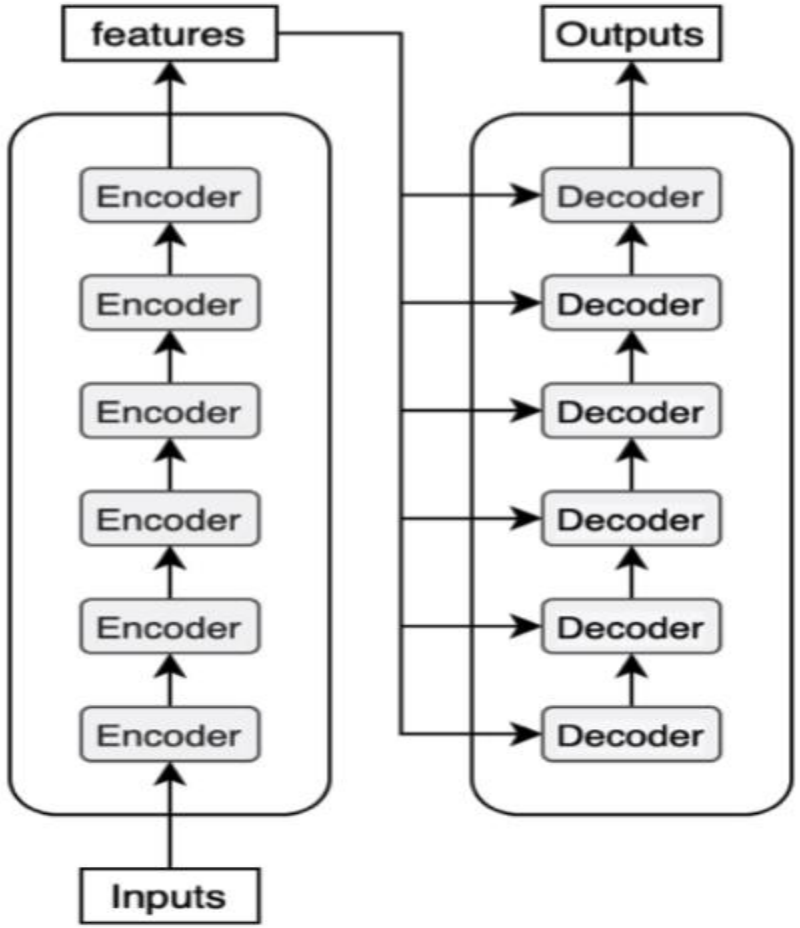
The encoder in the transformer consists of multiple encoder blocks. An input sentence goes through the encoder blocks, and the output of the last encoder block becomes the input features to the decoder.



The decoder also consists of multiple decoder blocks.

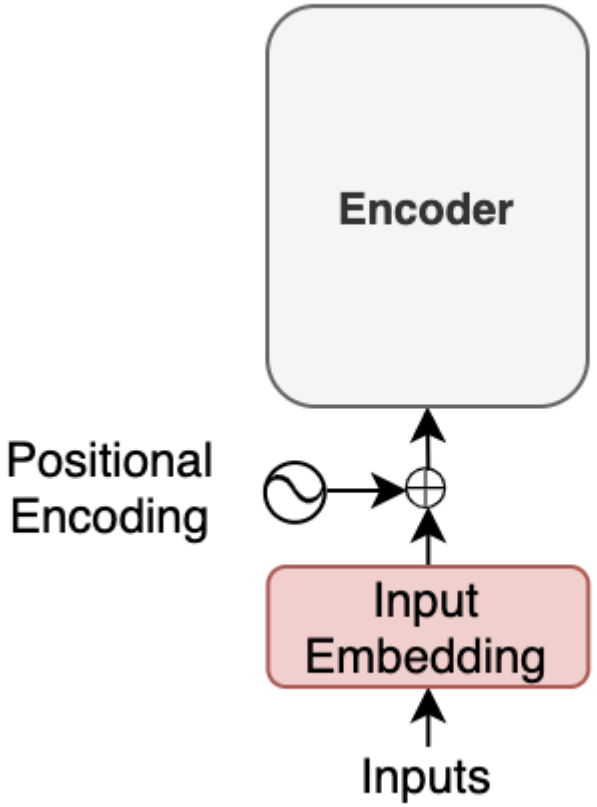


Each decoder block receives the features from the encoder. If we draw the encoder and the decoder vertically, the whole picture looks like the diagram from the paper.



Input Embedding and Positional Encoding

Like any neural translation model, we often tokenize an input sentence into distinct elements (tokens). A tokenized sentence is a fixed-length sequence.



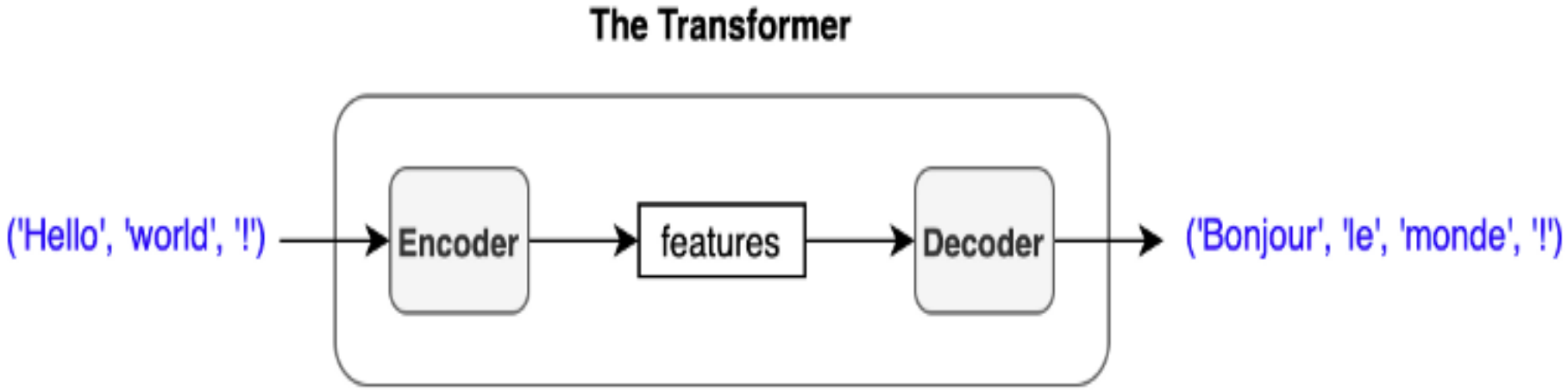
- with the transformer, we inject **positional encoding** into each embedding so that the model can know word positions without recurrence.

- The input to the transformer is not the characters of the input text but a sequence of embedding vectors.

- Each vector represents the semantics and position of a token. The encoder performs linear algebra operations on those vectors to extract the contexts for each token from the entire sentence and enrich the embedding vectors with helpful information for the target task through the multiple encoder blocks.

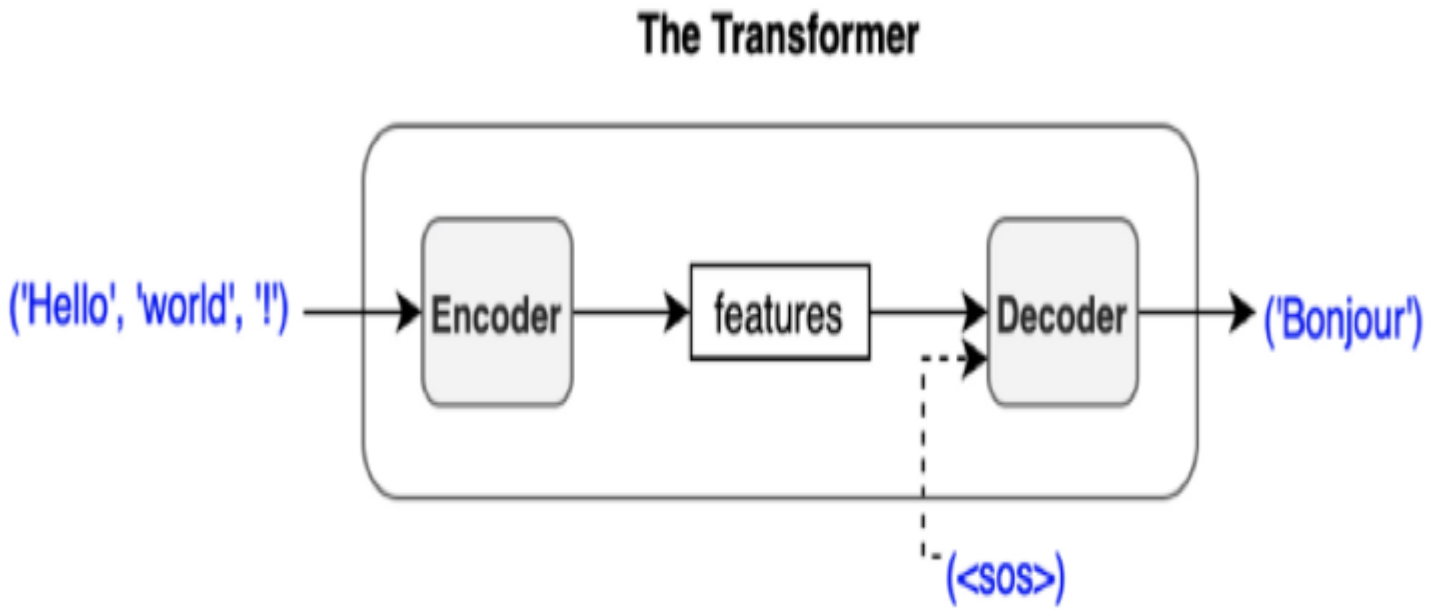
Softmax and Output Probabilities

The decoder uses input features from the encoder to generate an output sentence. The input features are nothing but enriched embedding vectors.



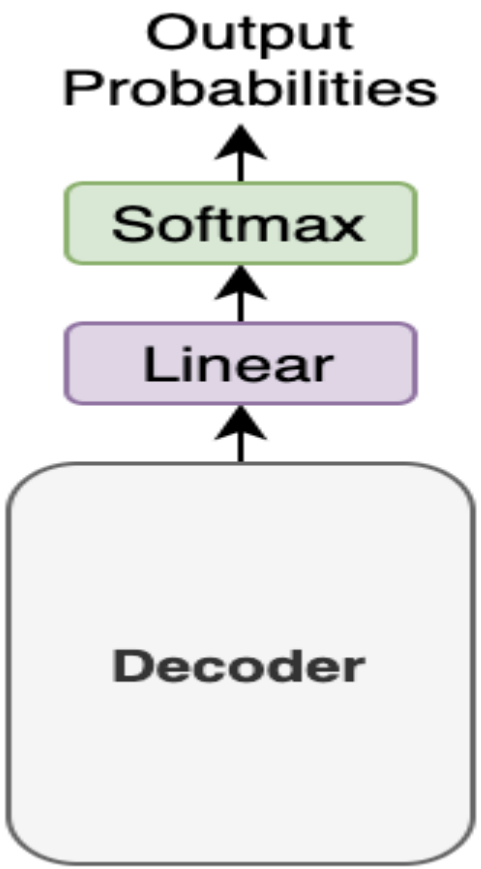
The decoder outputs one token at a time. An output token becomes the subsequent input to the decoder. In other words, a previous output from the decoder becomes the last part of the next input to the decoder. This kind of processing is called “**auto-regressive**”—a typical pattern for generating sequential outputs and not specific to the transformer. It allows a model to generate an output sentence of different lengths than the input.

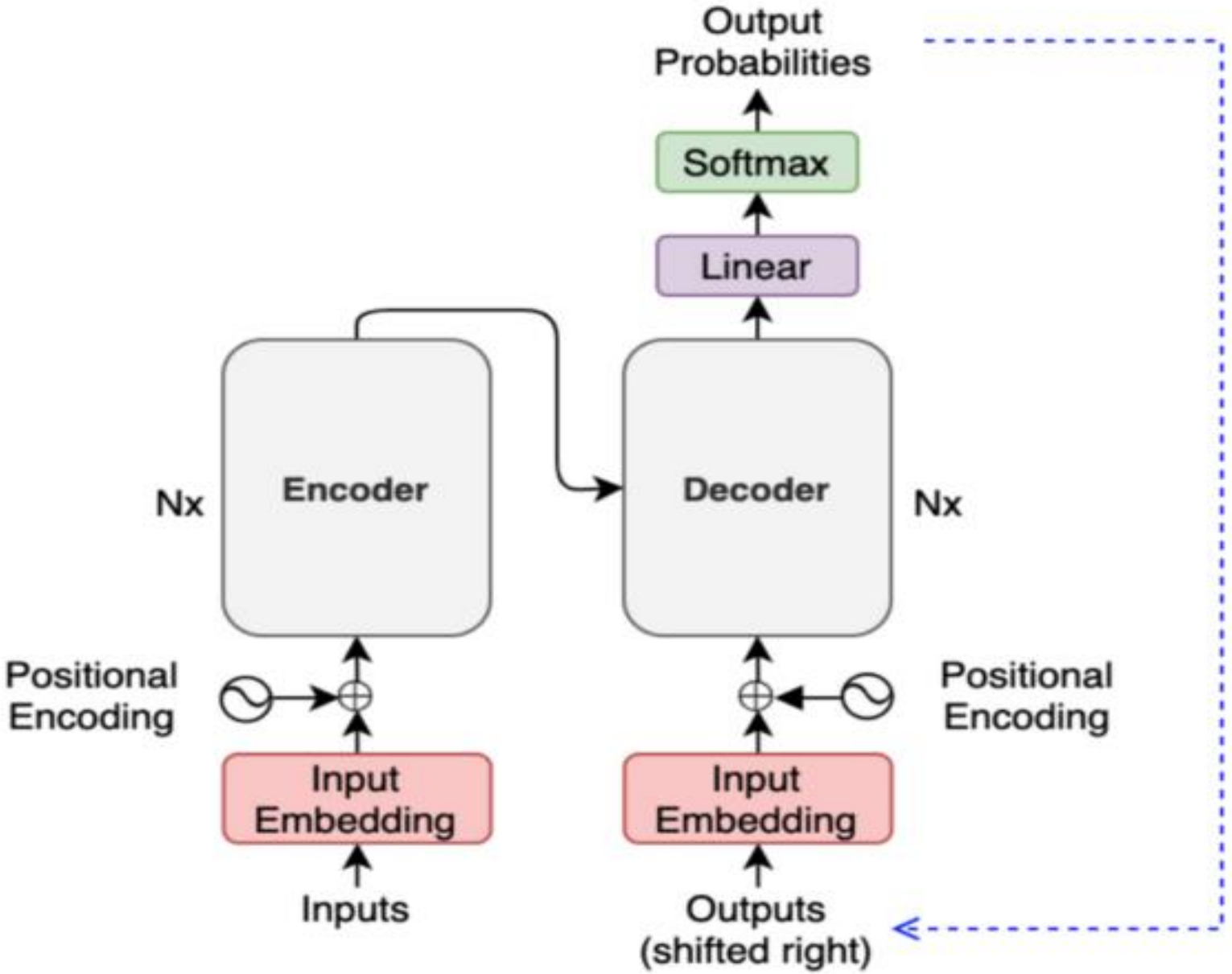
However, we have no previous output at the beginning of a translation. So, we pass the start-of-sentence marker <SOS> to the decoder to initiate the translation.



The decoder uses multiple decoder blocks to enrich <SOS> with the contextual information from the input features. In other words, the decoder transforms the embedding vector <SOS> into a vector containing information helpful to generating the first translated word (token).

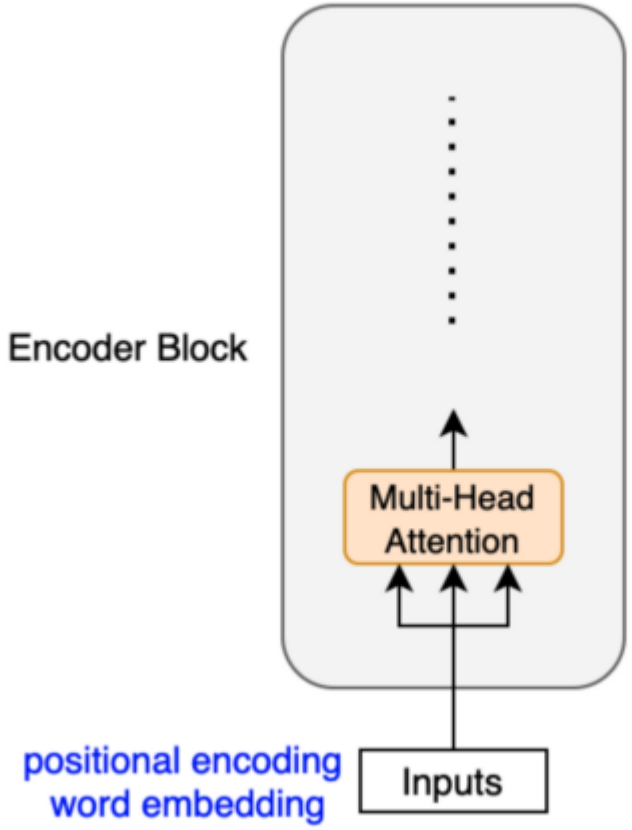
Then, the output vector from the decoder goes through a linear transformation that changes the dimension of the vector from the embedding vector size (512) into the size of vocabulary (say, 10,000). The softmax layer further converts the vector into 10,000 probabilities.


Output
Probabilities
Softmax
Linear
Decoder

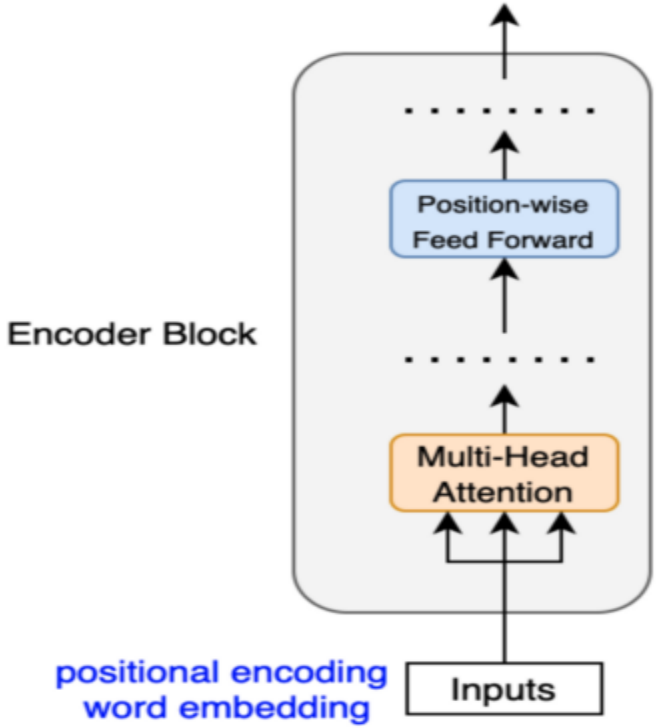

Output Probabilities
Softmax
Linear
Nx
Encoder
Decoder
Nx
Positional Encoding
Positional Encoding
Input Embedding
Input Embedding
Inputs
Outputs (shifted right)

Encoder Block Internals

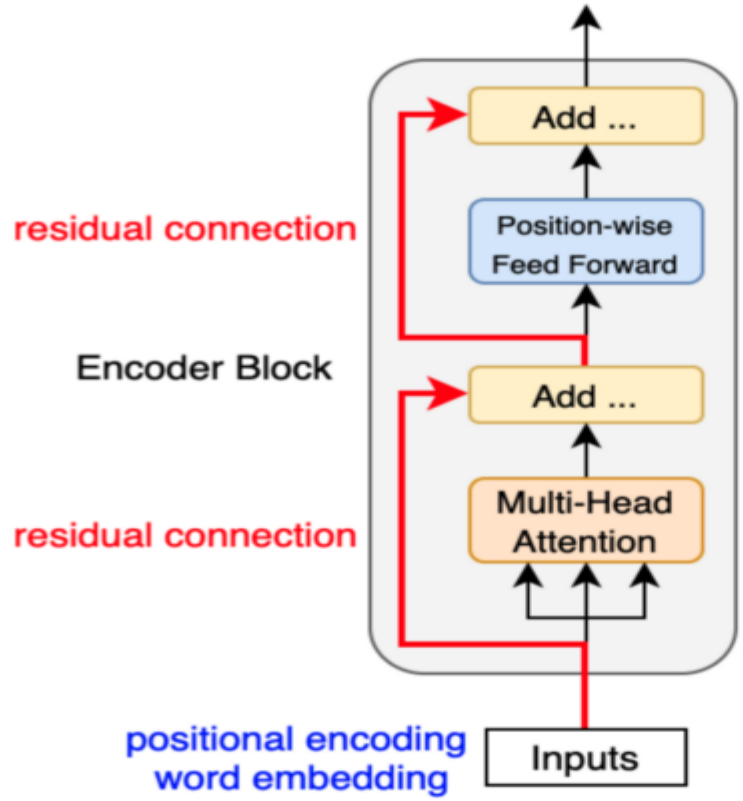
The encoder block uses the **self-attention mechanism** to enrich each token (embedding vector) with contextual information from the whole sentence.



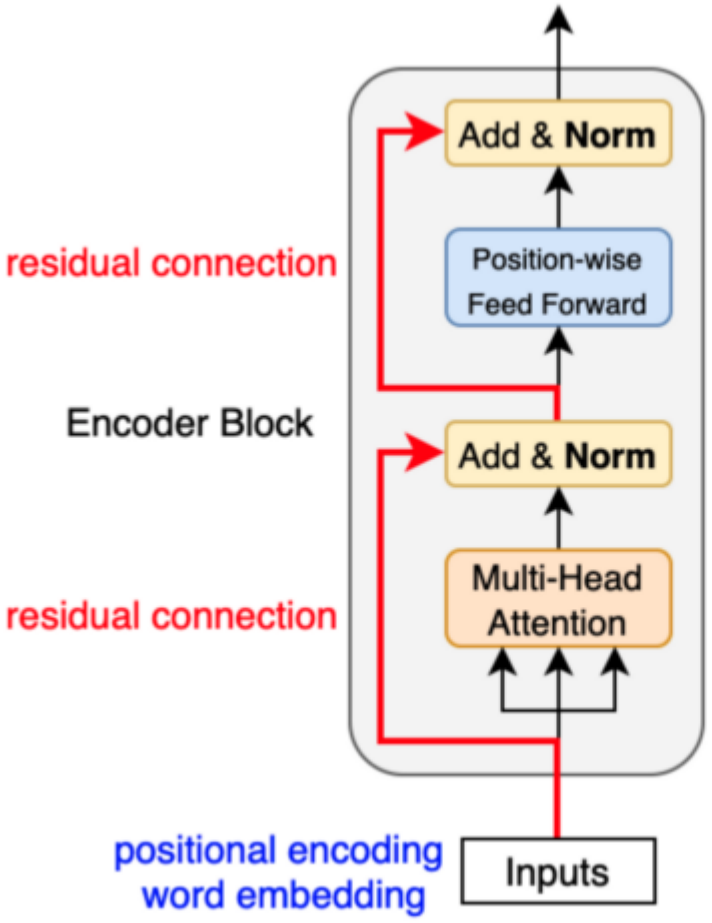
The **position-wise feed-forward network** (FFN) has a linear layer, ReLU, and another linear layer, which processes each embedding vector independently with identical weights. So, each embedding vector (with contextual information from the multi-head attention) goes through the position-wise feed-forward layer for further transformation.



Residual connections carry over the previous embeddings to the subsequent layers. As such, the encoder blocks enrich the embedding vectors with additional information obtained from the multi-head self-attention calculations and position-wise feed-forward networks.



After each residual connection, there is a **layer normalization**:

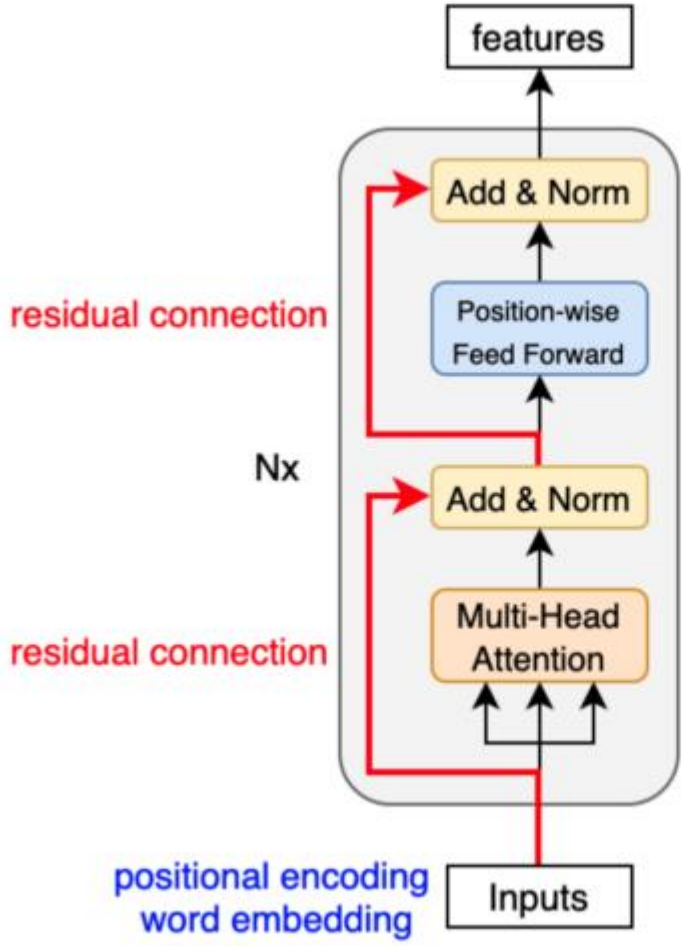


Like batch normalization, layer normalization aims to reduce the effect of covariant shift.

In other words, it prevents the mean and standard deviation of embedding vector elements from moving around, which makes training unstable and slow (i.e., we can’t make the learning rate big enough).

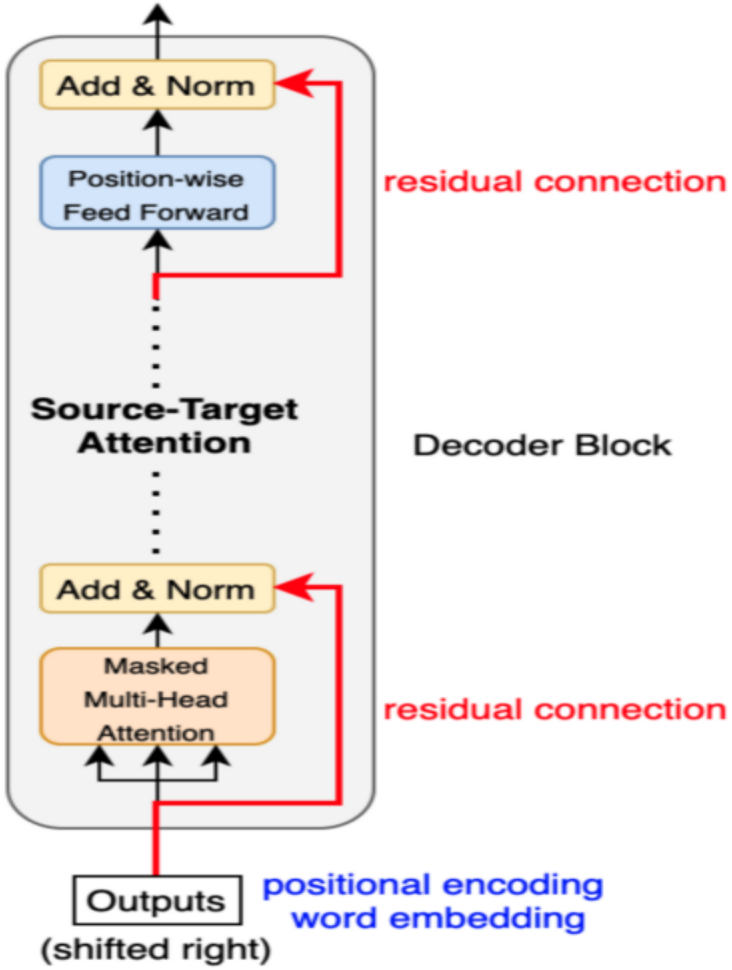
Unlike batch normalization, layer normalization works at each embedding vector (not at the batch level).

The transformer uses six stacked encoder blocks. The outputs from the last encoder block become the input features for the decoder.



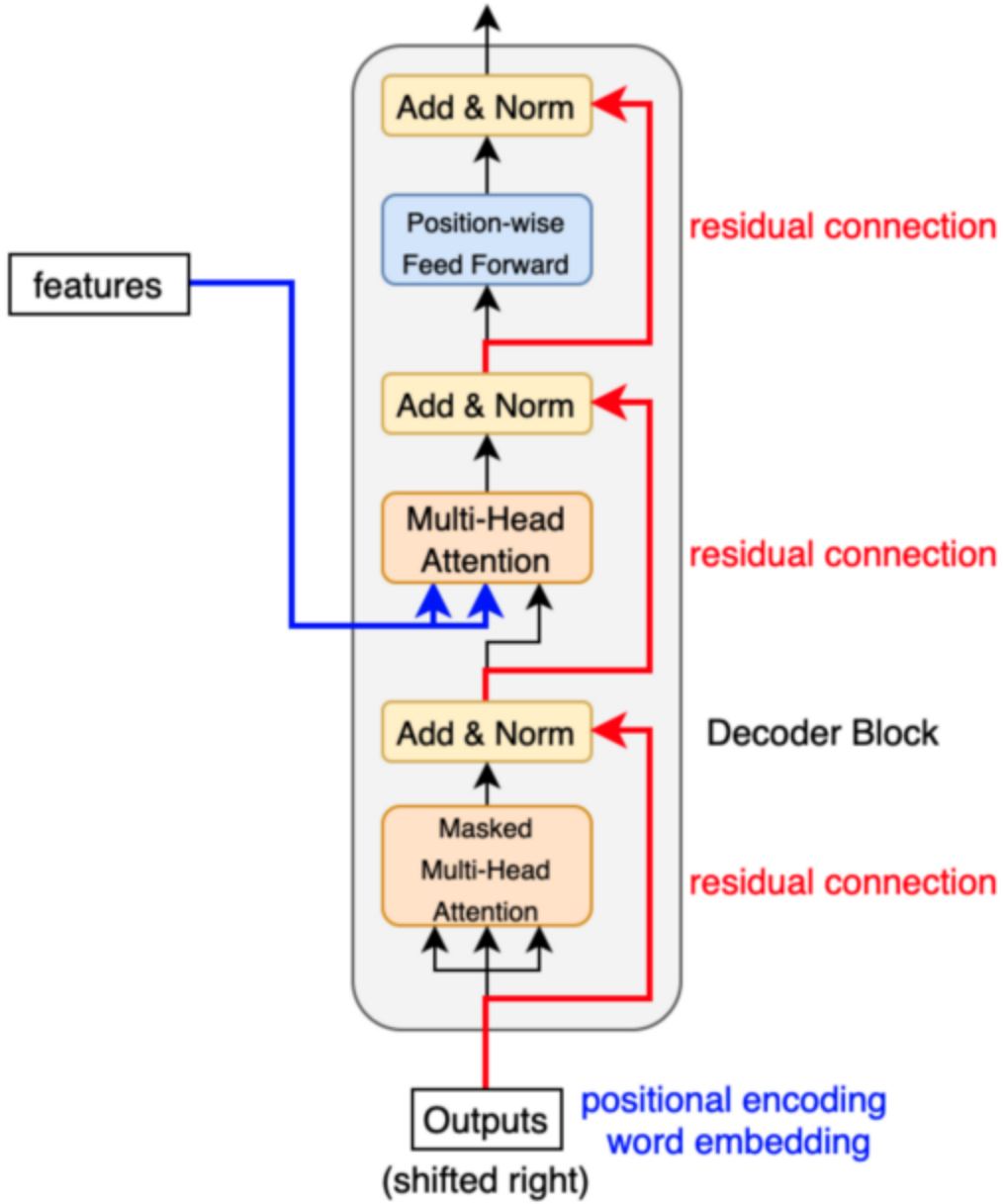
Decoder Block Internals

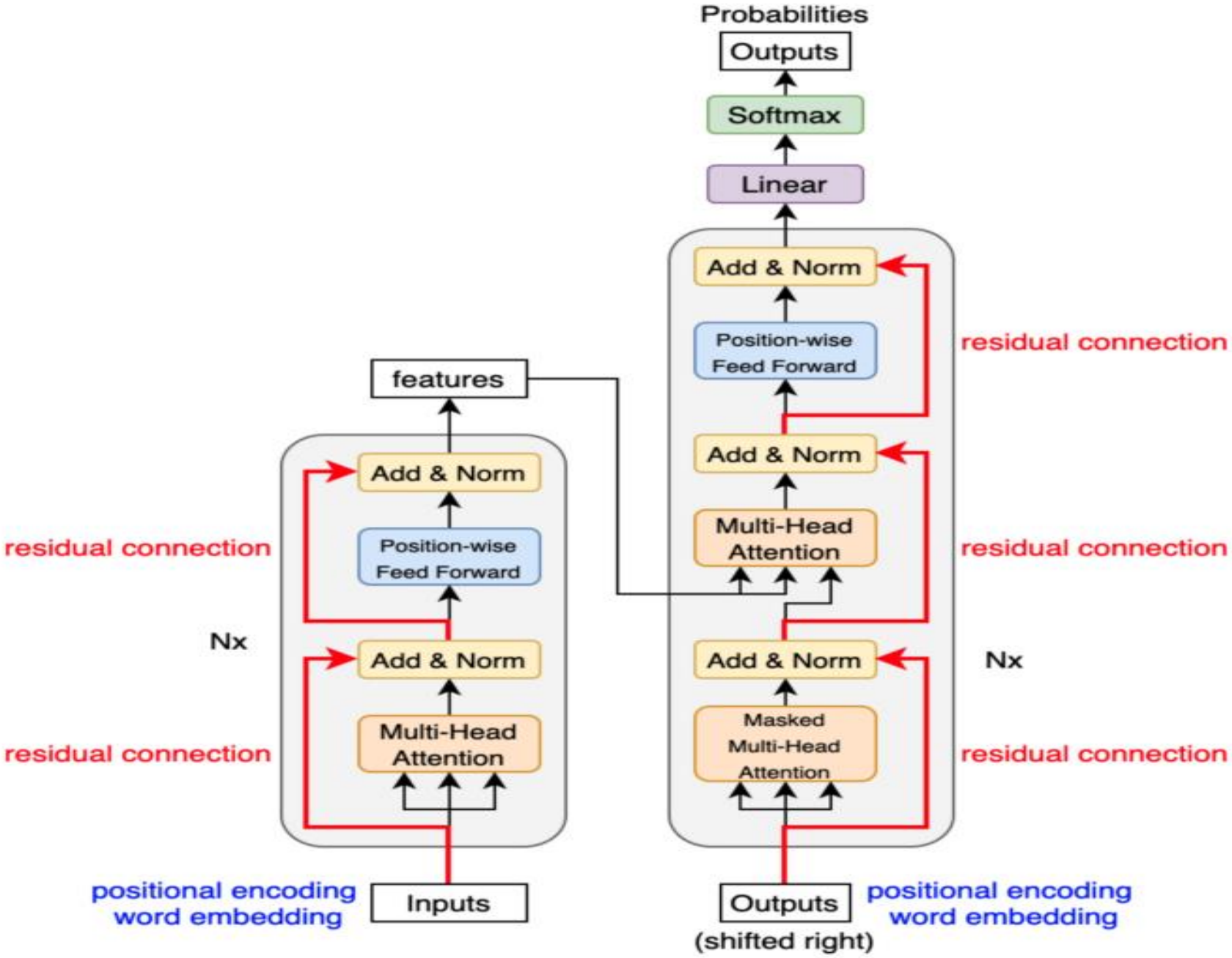
The decoder block is similar to the encoder block, except it calculates the source-target attention.

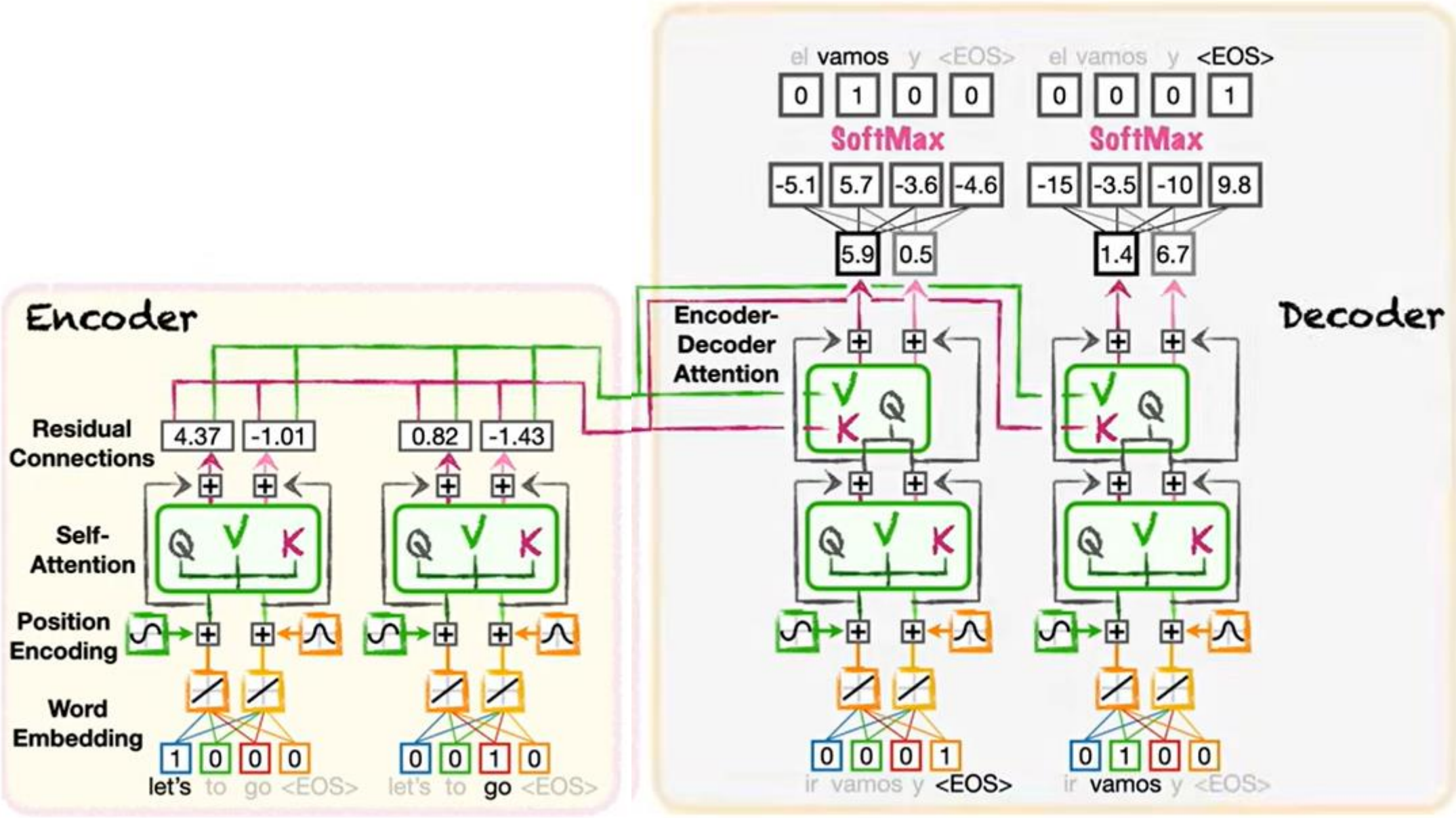


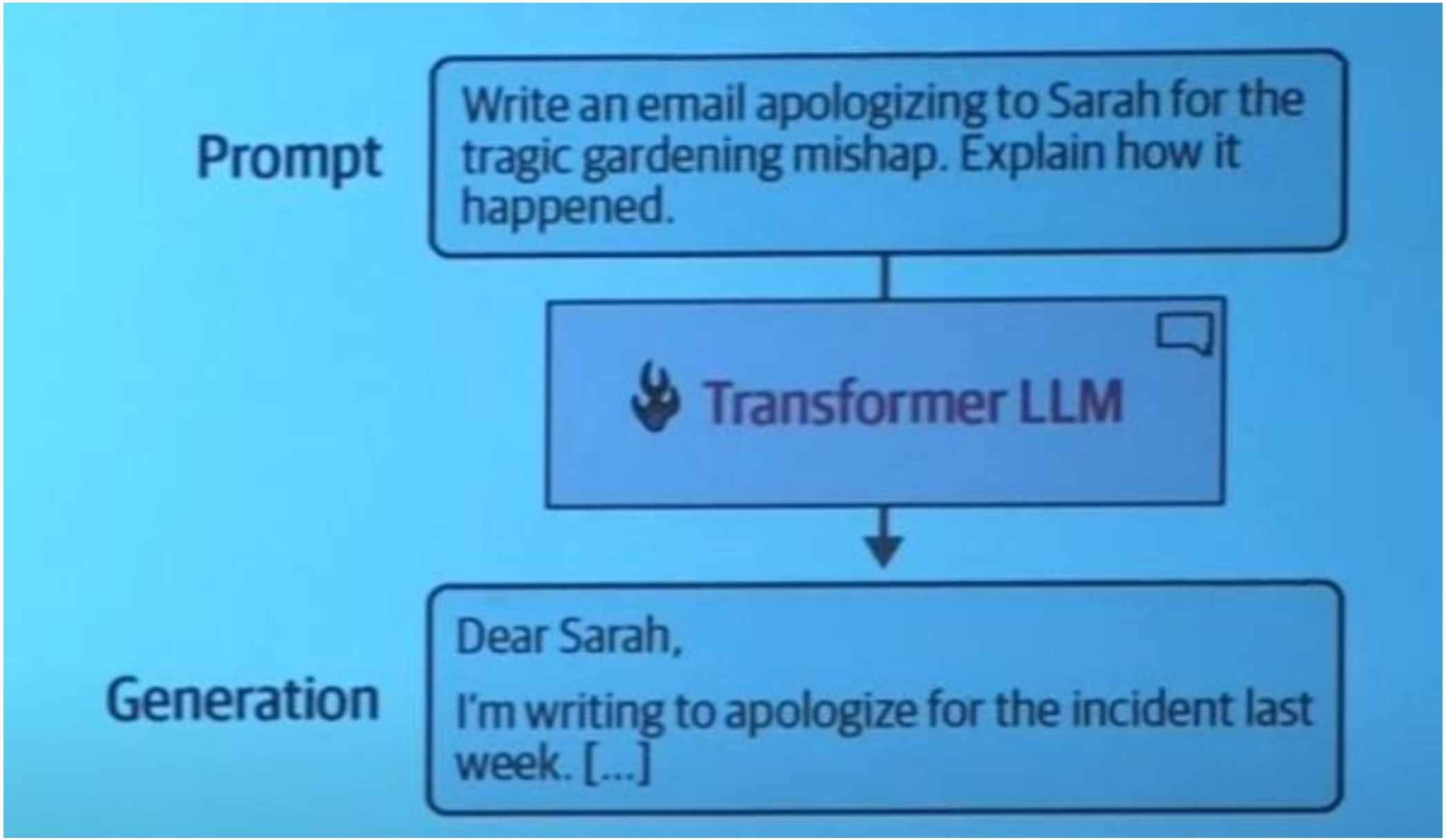
The input to the decoder is an output shifted right, which becomes a sequence of embeddings with positional encoding. So, we can think of the decoder block as another encoder generating enriched embeddings useful for translation outputs.

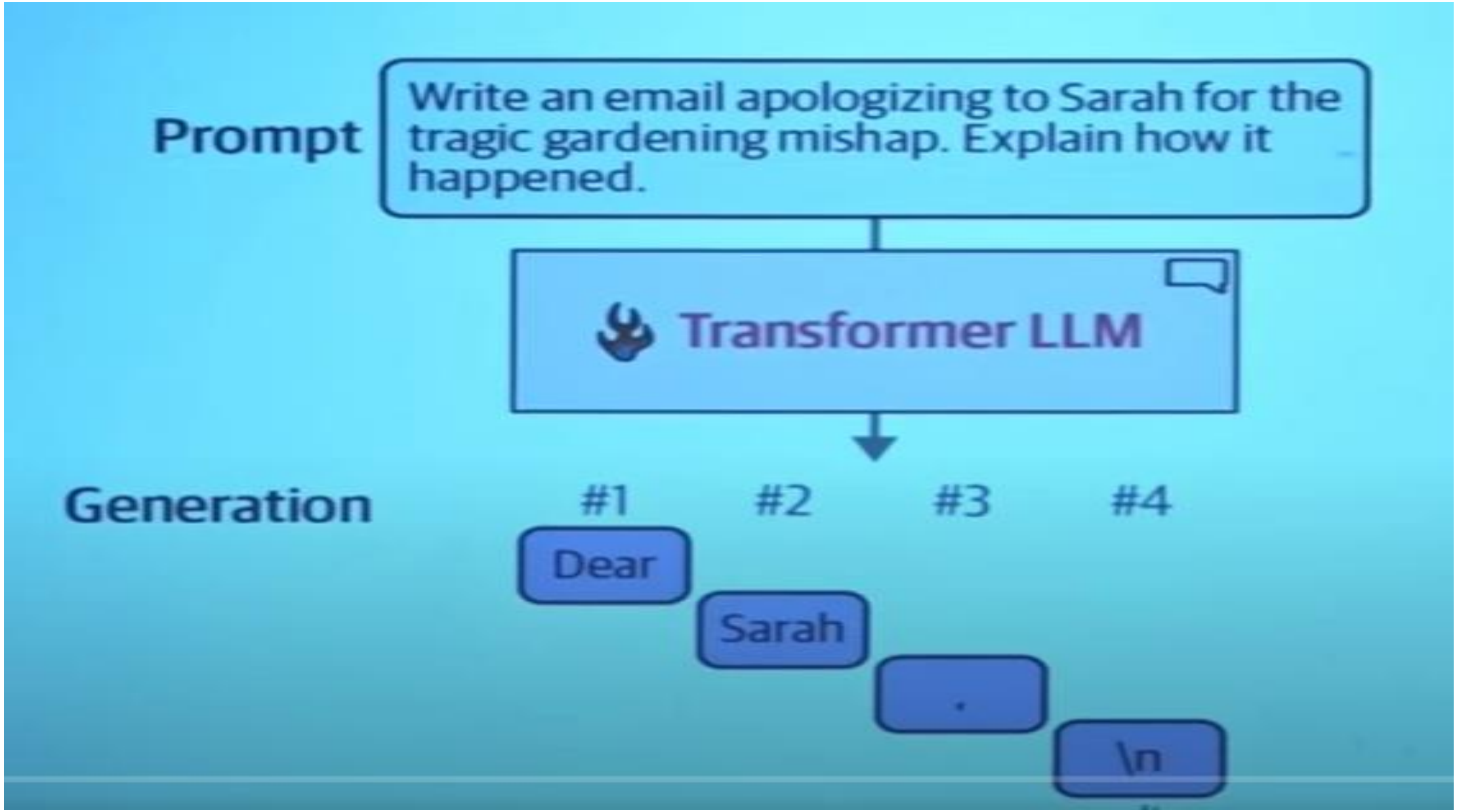
Masked multi-head attention means the multi-head attention receives inputs with masks so that the attention mechanism does not use information from the hidden (masked) positions. The paper mentions that they used the mask inside the attention calculation by setting attention scores to negative infinity (or a very large negative number). The softmax within the attention mechanisms effectively assigns zero probability to masked positions.

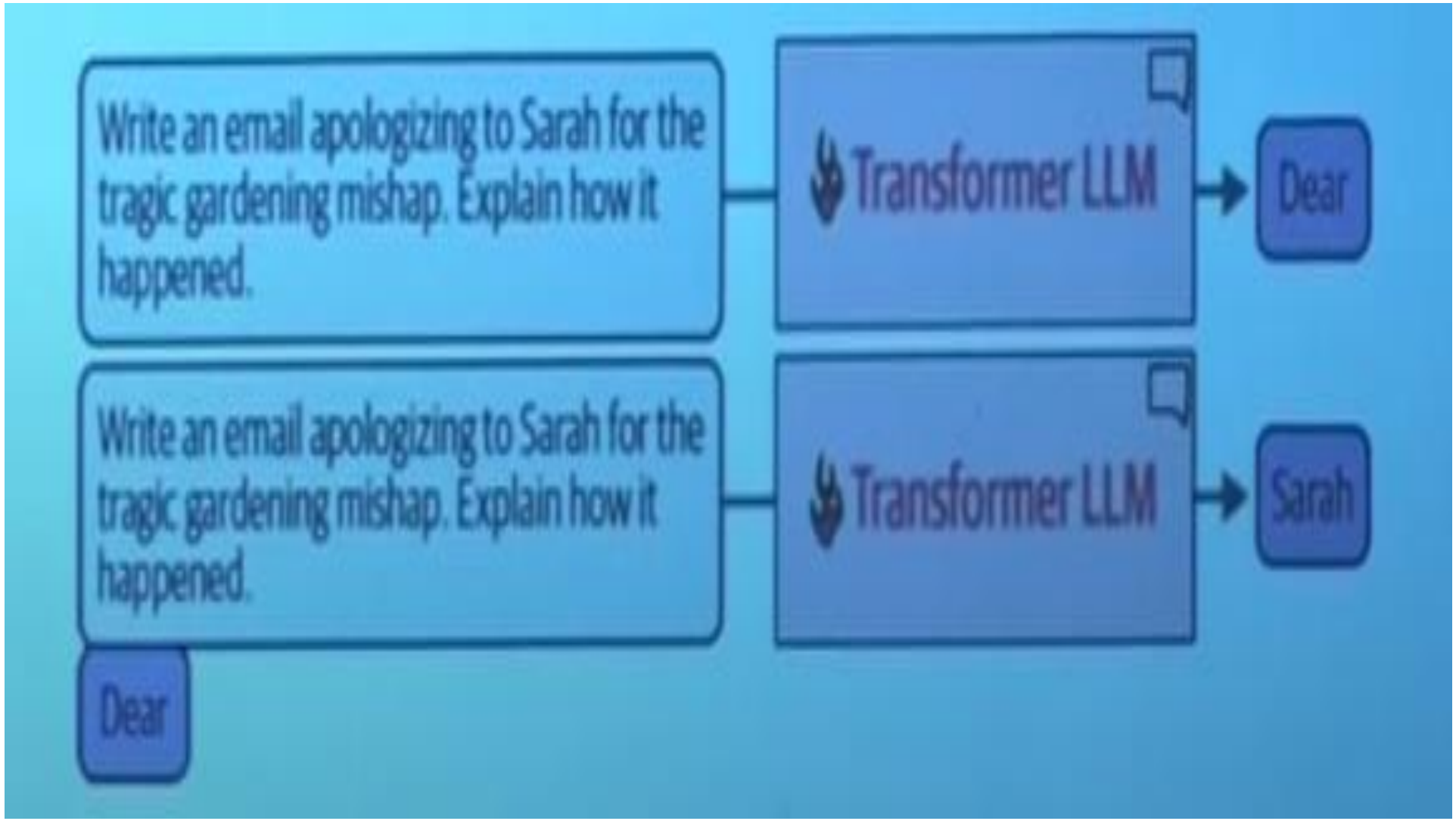

Add & Norm
Position-wise
Feed Forward
features
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Outputs
(shifted right)
residual connection
residual connection
Decoder Block
residual connection
positional encoding
word embedding

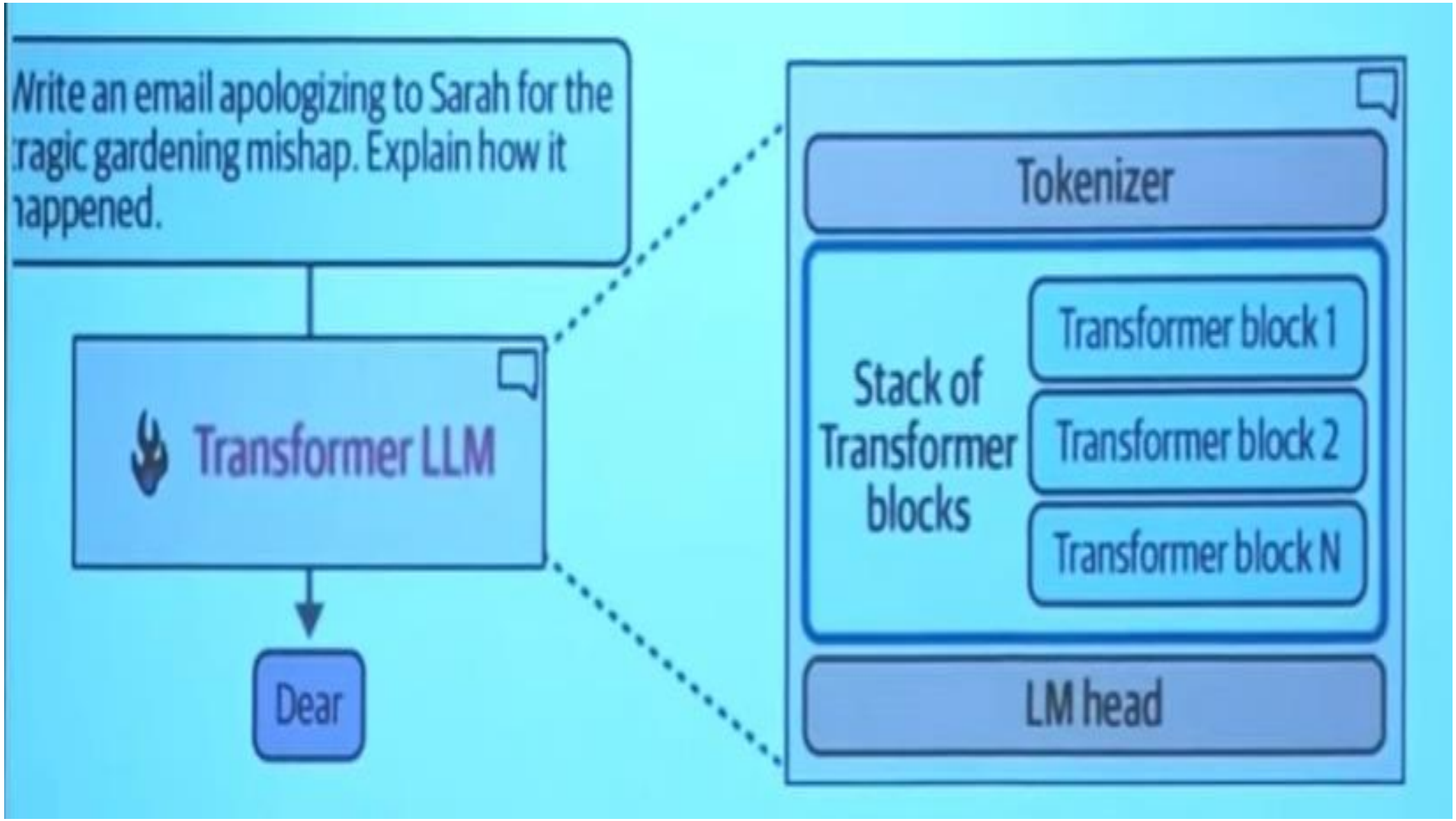

Probabilities
Outputs
Softmax
Linear
Add & Norm
Position-wise Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
residual connection
residual connection
residual connection
Nx
Nx
features
Add & Norm
Position-wise Feed Forward
Add & Norm
Multi-Head Attention
residual connection
residual connection
residual connection
positional encoding
word embedding
Inputs
Outputs
(shifted right)
positional encoding
word embedding

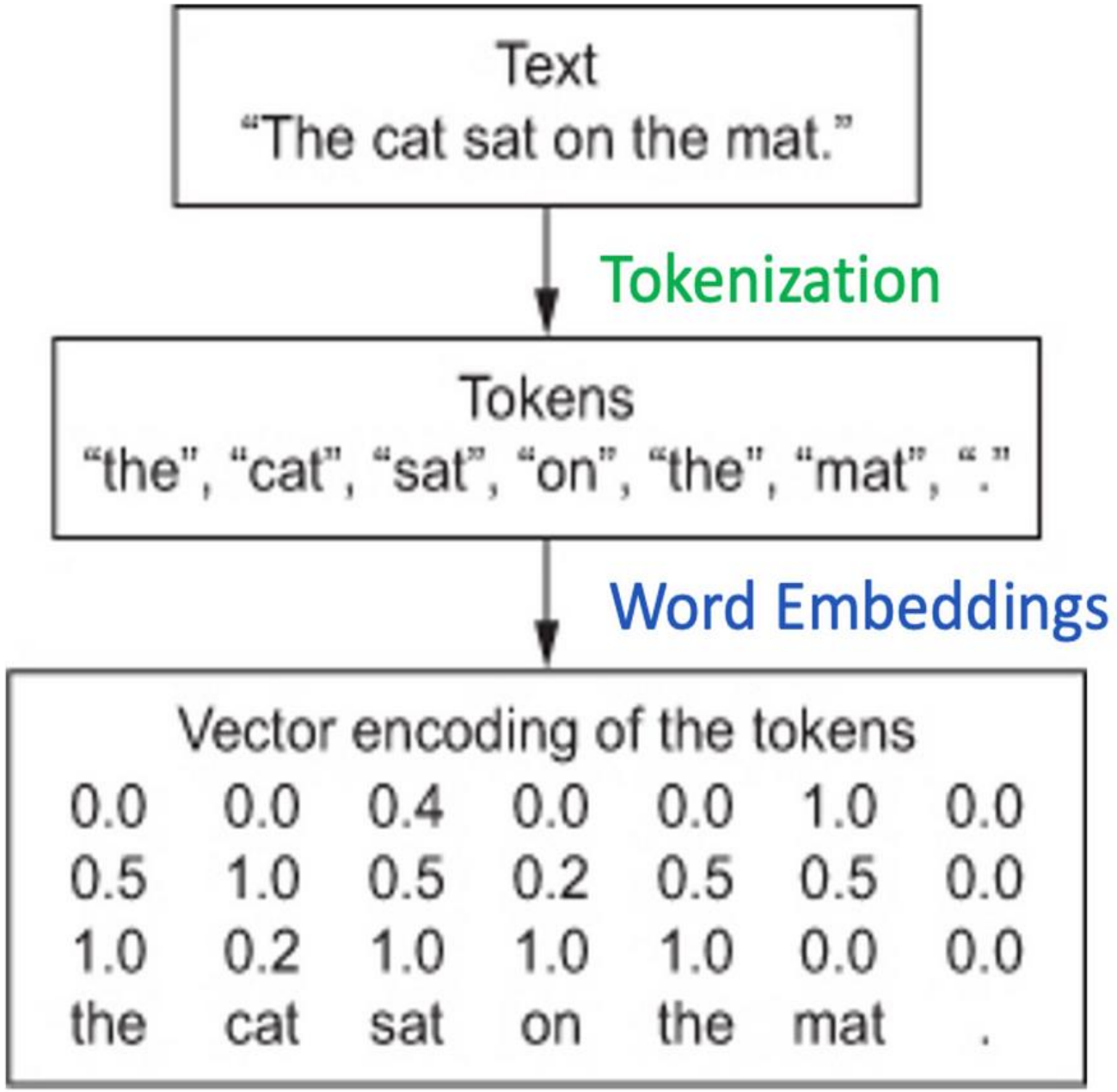

Encoder
Residual Connections
4.37
-1.01
0.82
-1.43
Self-Attention
Q
V
K
Position Encoding
Word Embedding
1 0 0 0
let's to go <EOS>
0 0 1 0
let's to go <EOS>
Encoder-Decoder Attention
Decoder
el vamos y <EOS>
0 1 0 0
SoftMax
-5.1 5.7 -3.6 -4.6
5.9 0.5
el vamos y <EOS>
0 0 0 1
SoftMax
-15 -3.5 -10 9.8
1.4 6.7
0 0 0 1
ir vamos y <EOS>
0 1 0 0
ir vamos y <EOS>

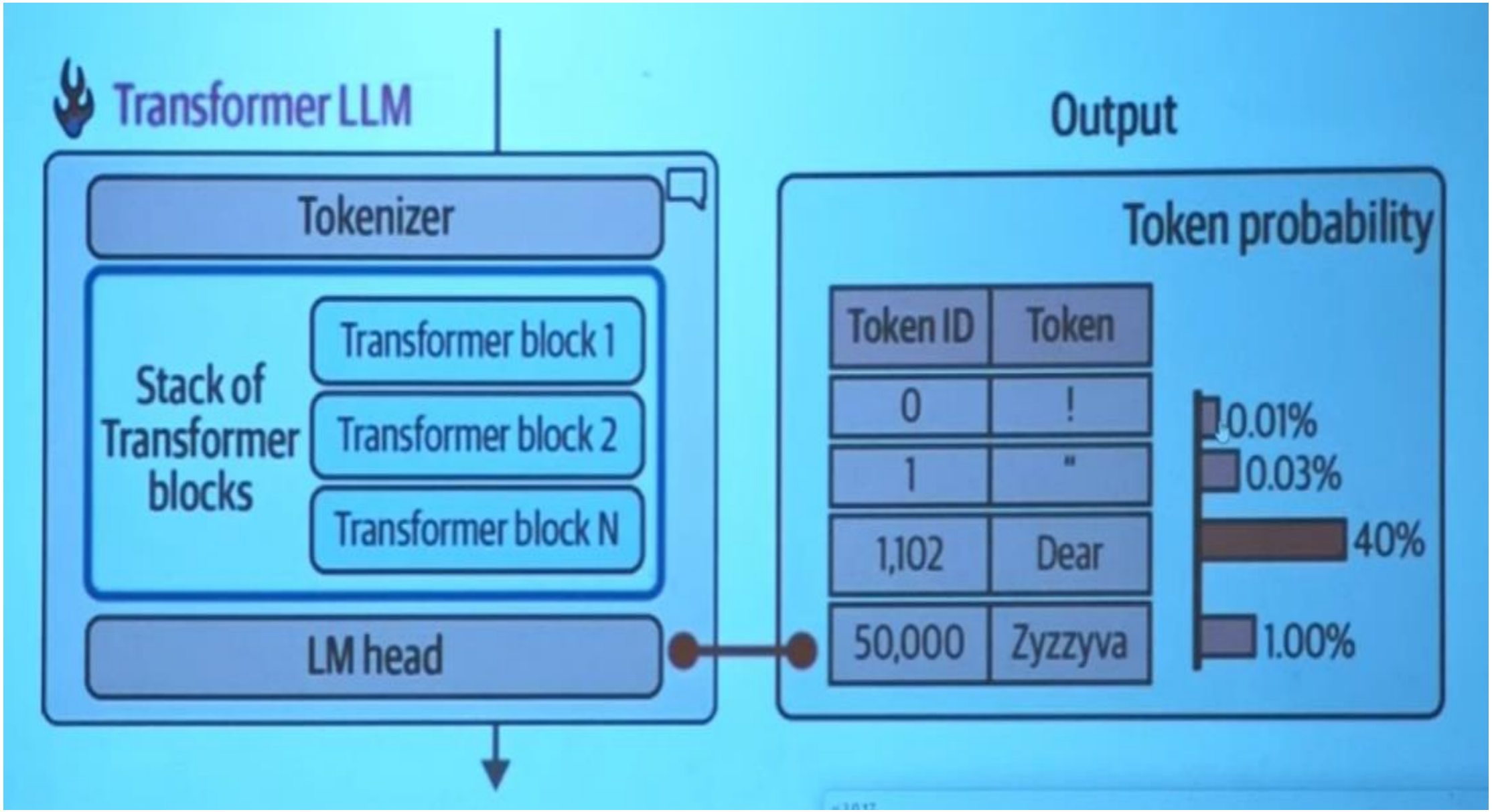

Prompt
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Generation
Dear Sarah,
I'm writing to apologize for the incident last week. [...]

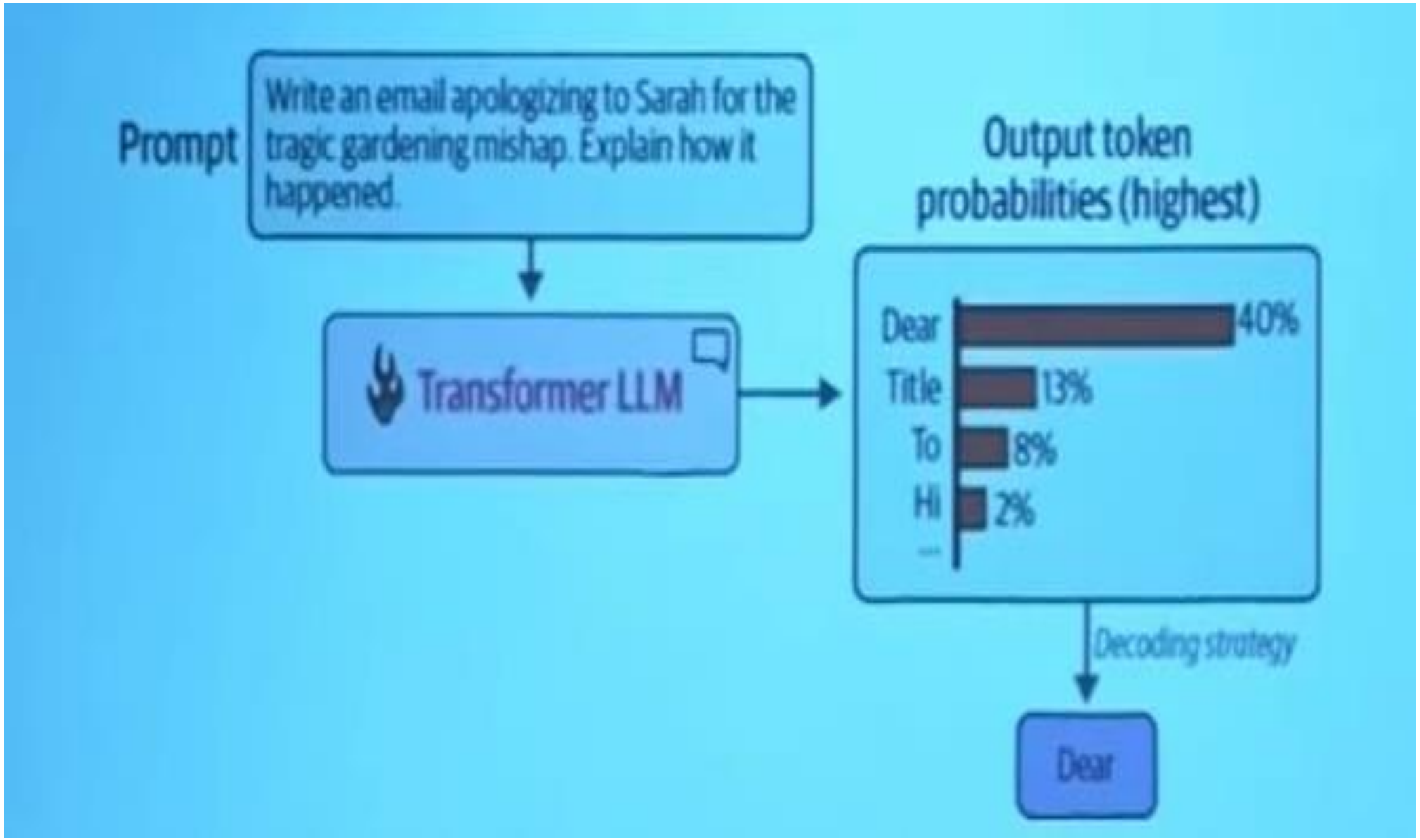

Prompt
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Generation
#1
#2
#3
#4
Dear
Sarah
.
\n

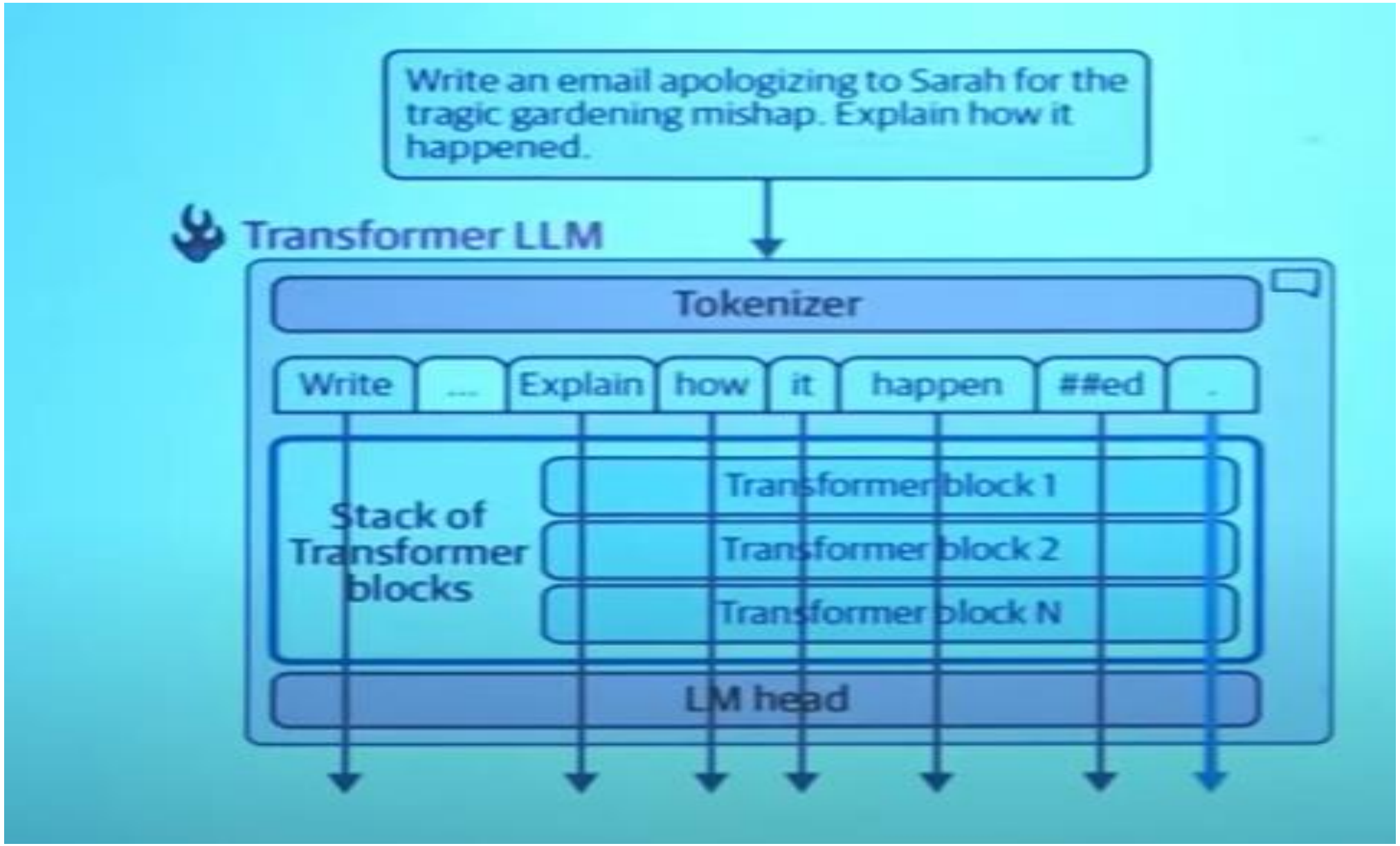

Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Dear
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened. Dear
Transformer LLM
Sarah

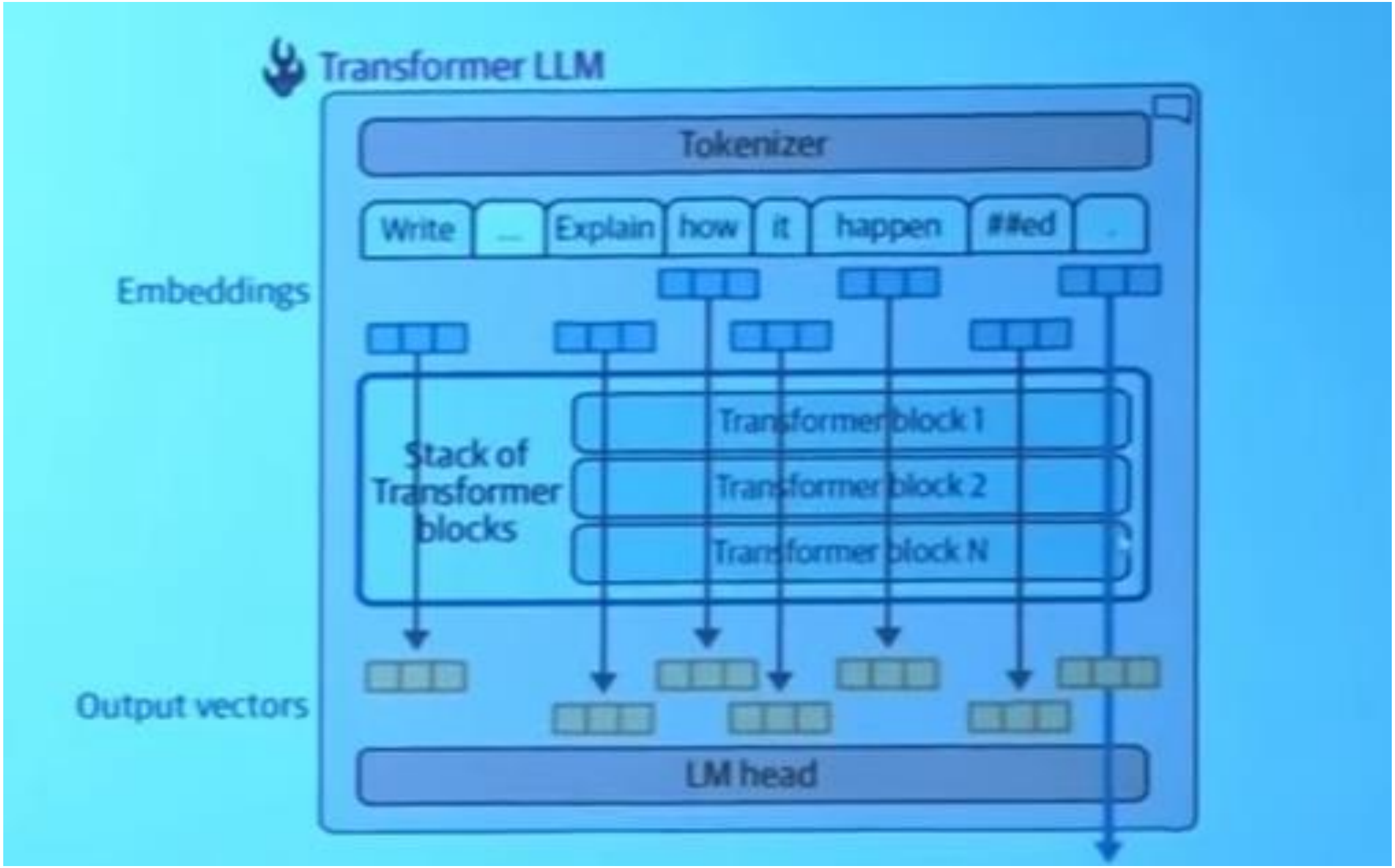

Write an email apologizing to Sarah for the
tragic gardening mishap. Explain how it
happened.
Transformer LLM
Dear
Tokenizer
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
LM head

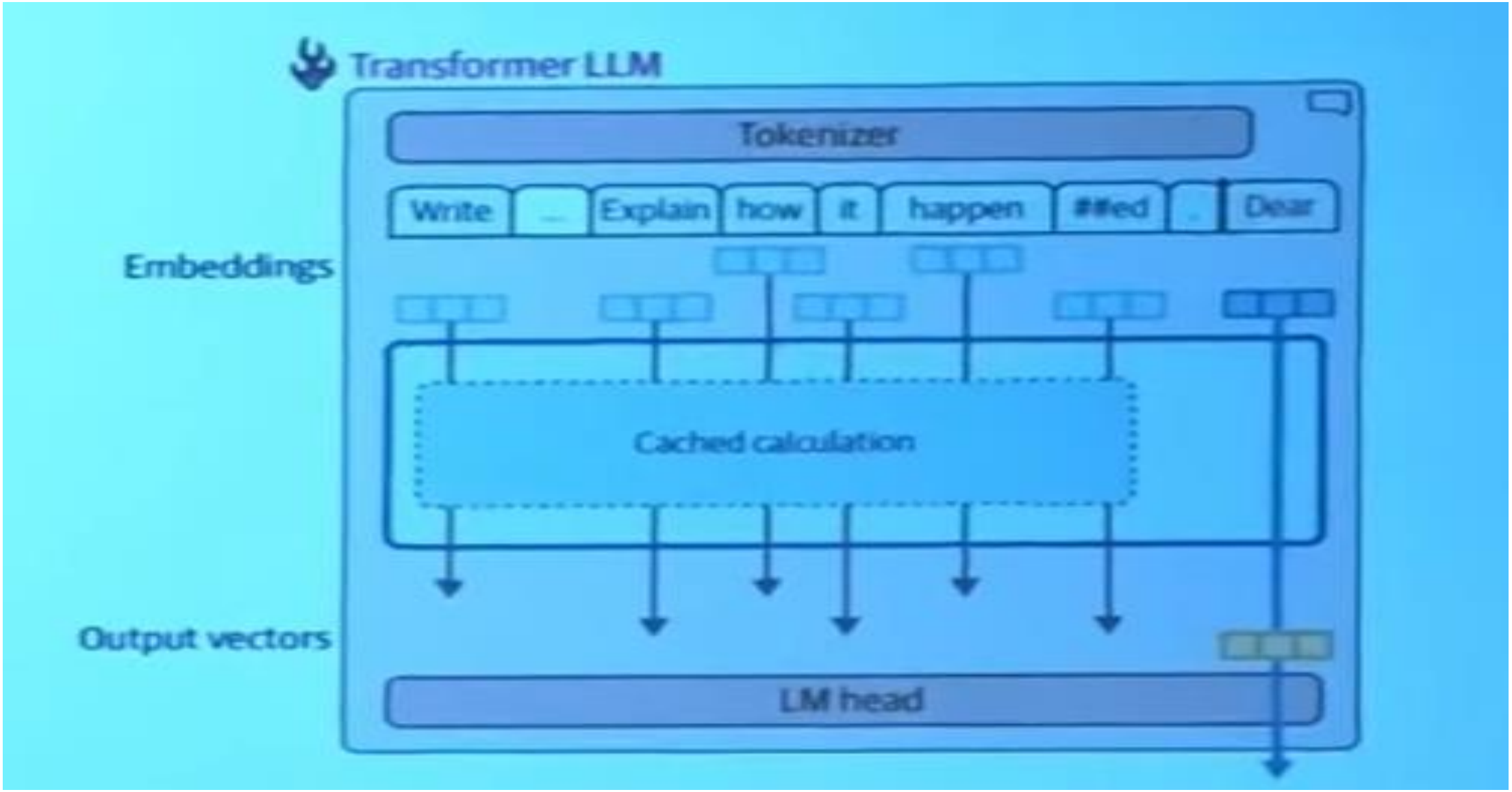

Text
"The cat sat on the mat."
Tokenization
Tokens
"the", "cat", "sat", "on", "the", "mat", "."
Word Embeddings
Vector encoding of the tokens
0.0 0.0 0.4 0.0 0.0 1.0 0.0
0.5 1.0 0.5 0.2 0.5 0.5 0.0
1.0 0.2 1.0 1.0 1.0 0.0 0.0
the cat sat on the mat .

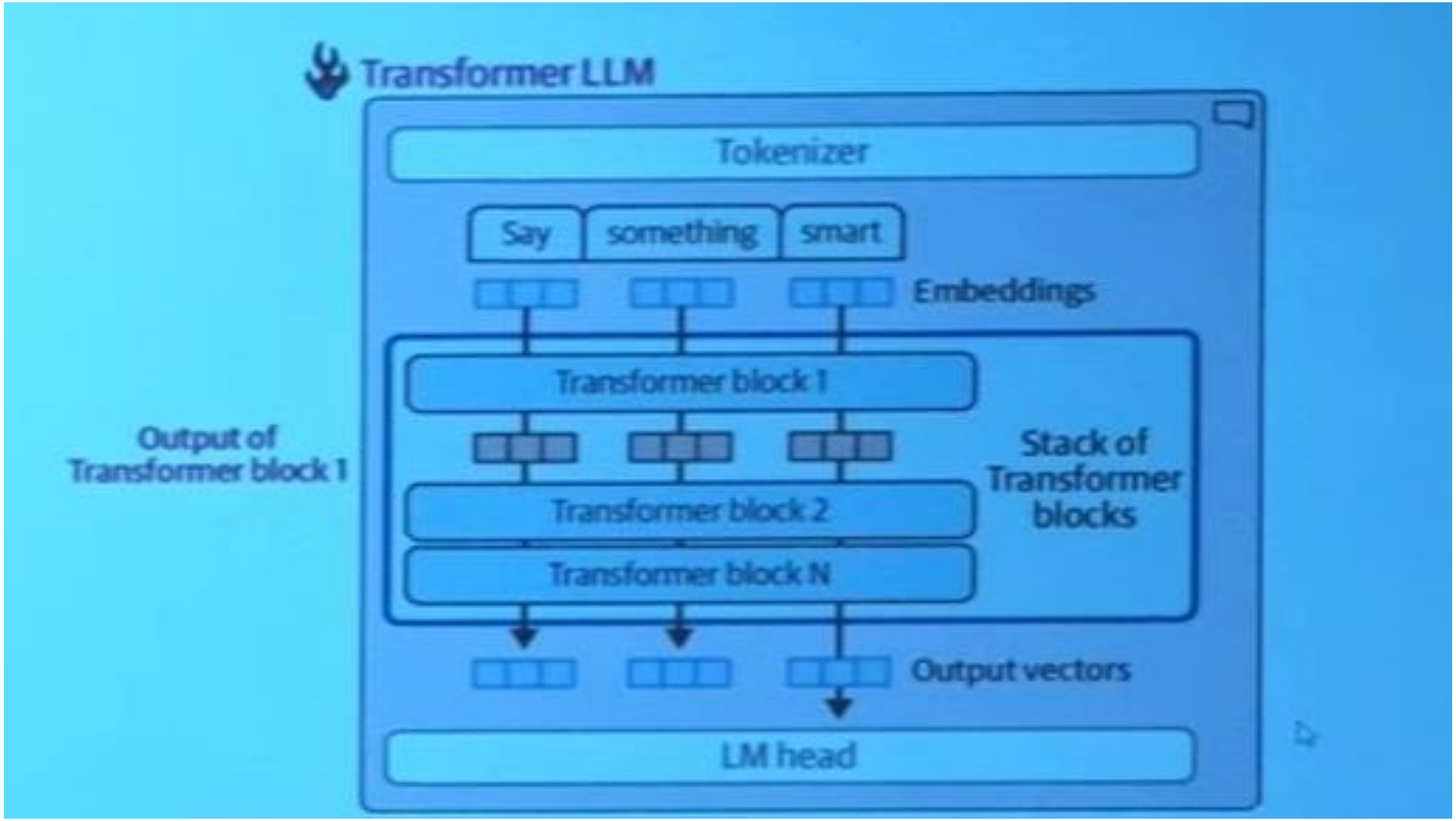

Transformer LLM
Tokenizer
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
LM head
Output
Token probability
Token ID
Token
0
!
1
"
1,102
Dear
50,000
Zyzzyva
0.01%
0.03%
40%
1.00%

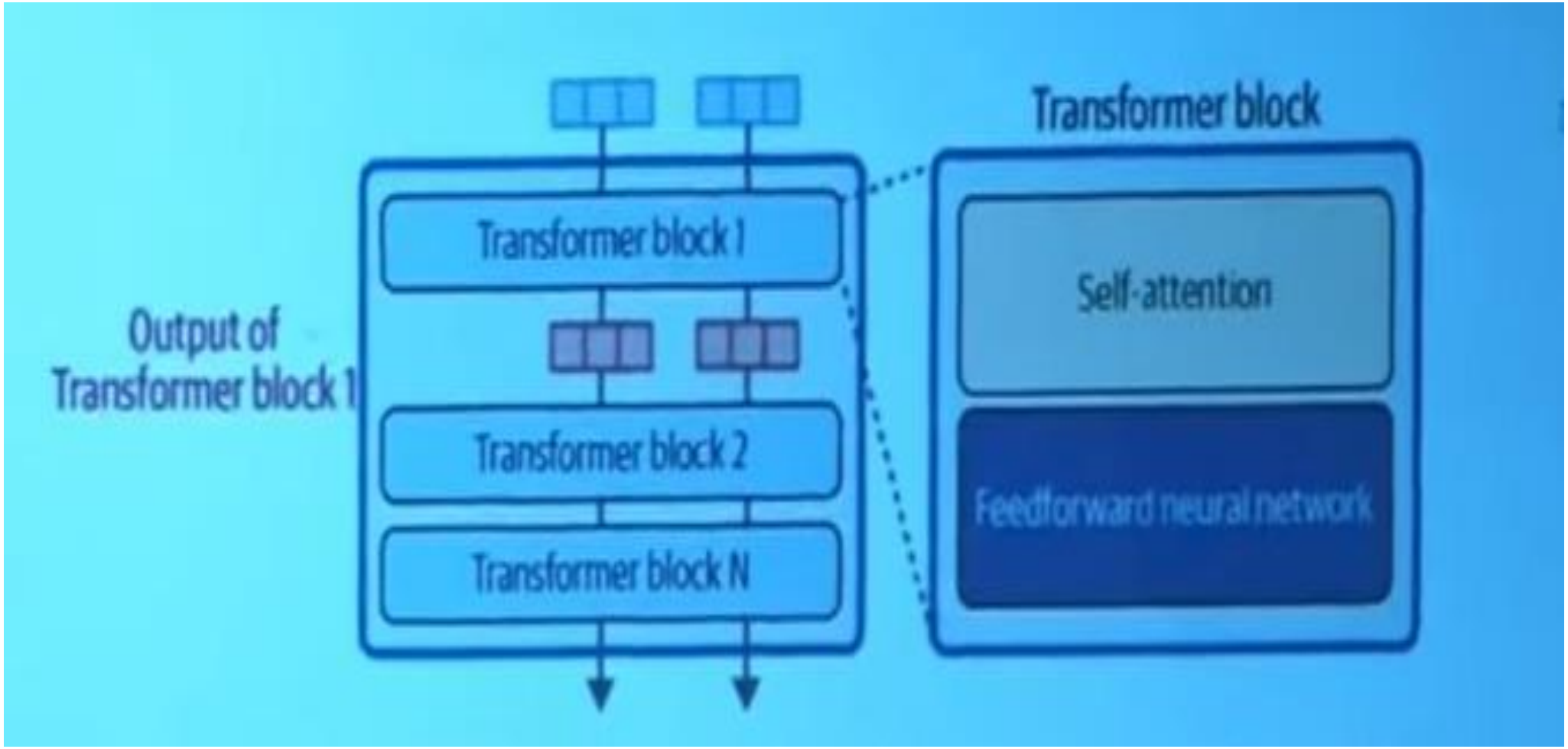

Prompt
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Output token probabilities (highest)
Dear 40%
Title 13%
To 8%
Hi 2%
...
Decoding strategy
Dear

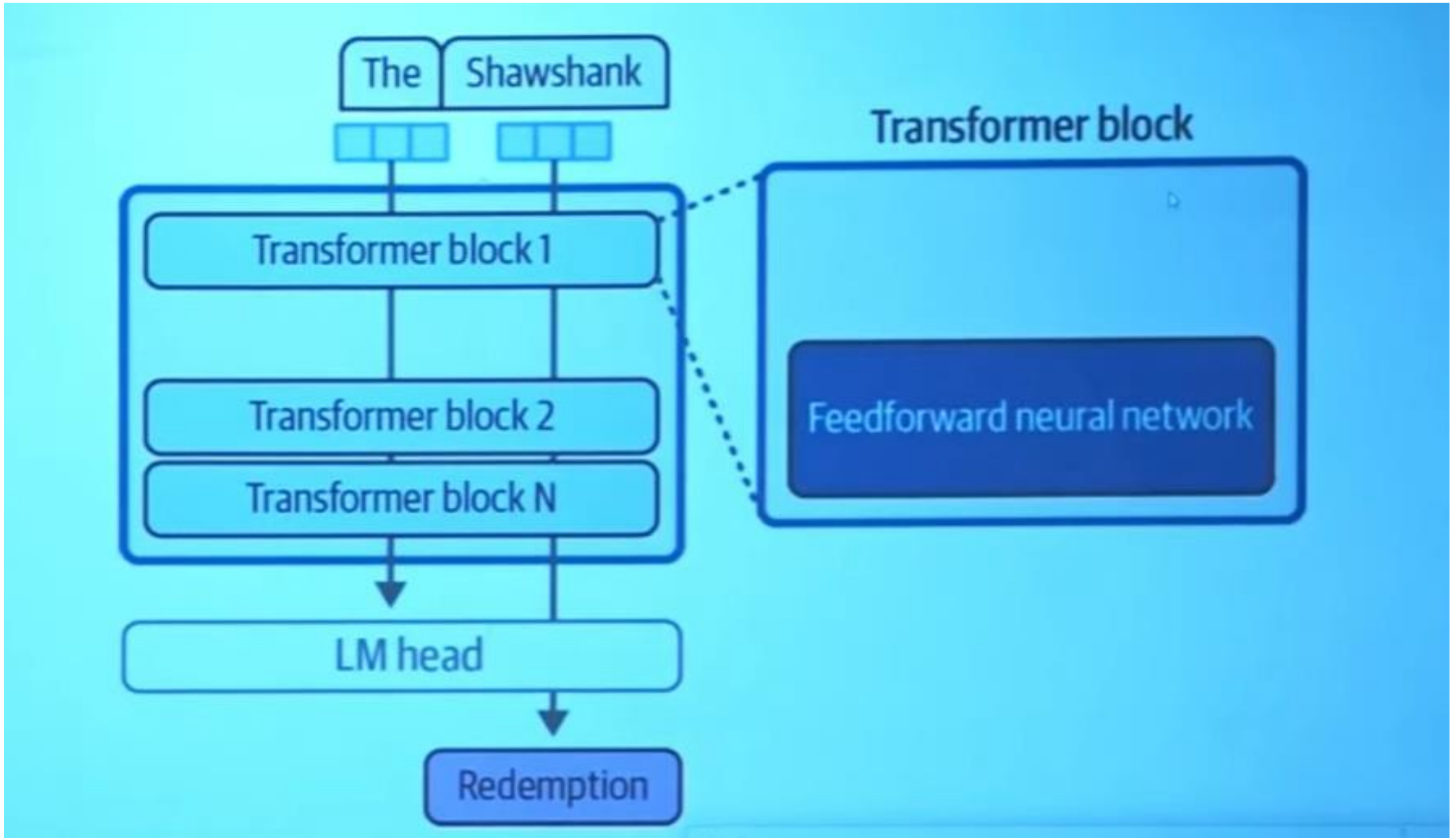

Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Tokenizer
Write
...
Explain
how
it
happen
##ed
.
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
LM head

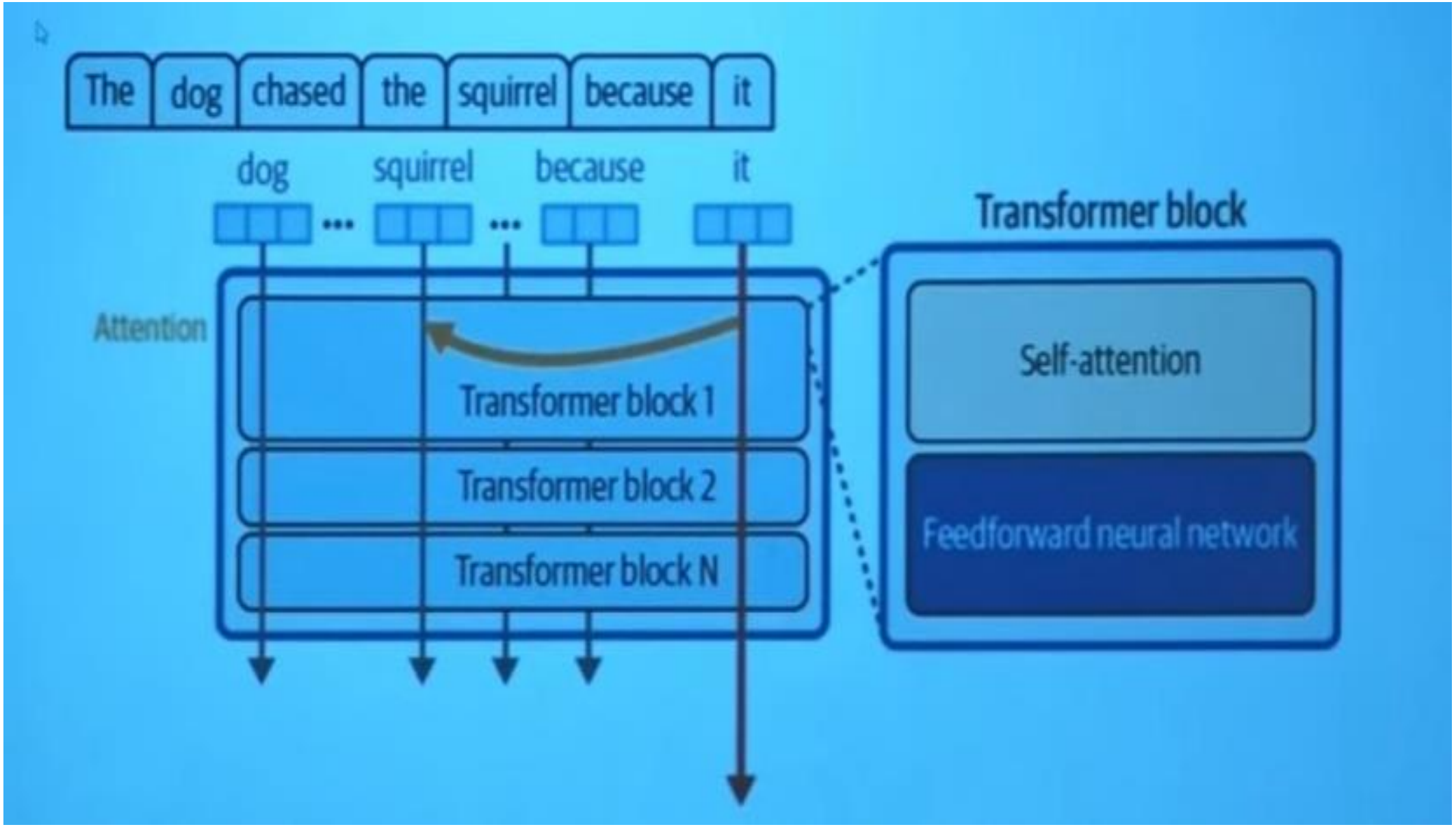

Transformer LLM
Tokenizer
Write
_
Explain
how
it
happen
##ed
.
Embeddings
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
Output vectors
LM head


Transformer LLM
Tokenizer
Write
_
Explain
how
it
happen
##ed
.
Dear
Embeddings
Cached calculation
Output vectors
LM head


Transformer LLM
Tokenizer
Say
something
smart
Embeddings
Transformer block 1
Output of
Transformer block 1
Stack of
Transformer
blocks
Transformer block 2
Transformer block N
Output vectors
LM head


Transformer block
Transformer block 1
Self-attention
Output of
Transformer block 1
Transformer block 2
Feedforward neural network
Transformer block N


The
Shawshank
Transformer block
Transformer block 1
Transformer block 2
Transformer block N
Feedforward neural network
LM head
Redemption


The dog chased the squirrel because it
dog
squirrel
because
it
Attention
Transformer block 1
Transformer block 2
Transformer block N
Transformer block
Self-attention
Feedforward neural network

Step 1 — Defining our Dataset

The dataset used for creating ChatGPT is **570 GB.** On the other hand, for our purposes, we will be using a very small dataset to perform numerical calculations visually.

Dataset (corpus)

I drink and I know things.
When you play the game of thrones, you win or you die.
The true enemy won't wait out the storm, He brings the storm.

Our entire dataset containing only three sentences

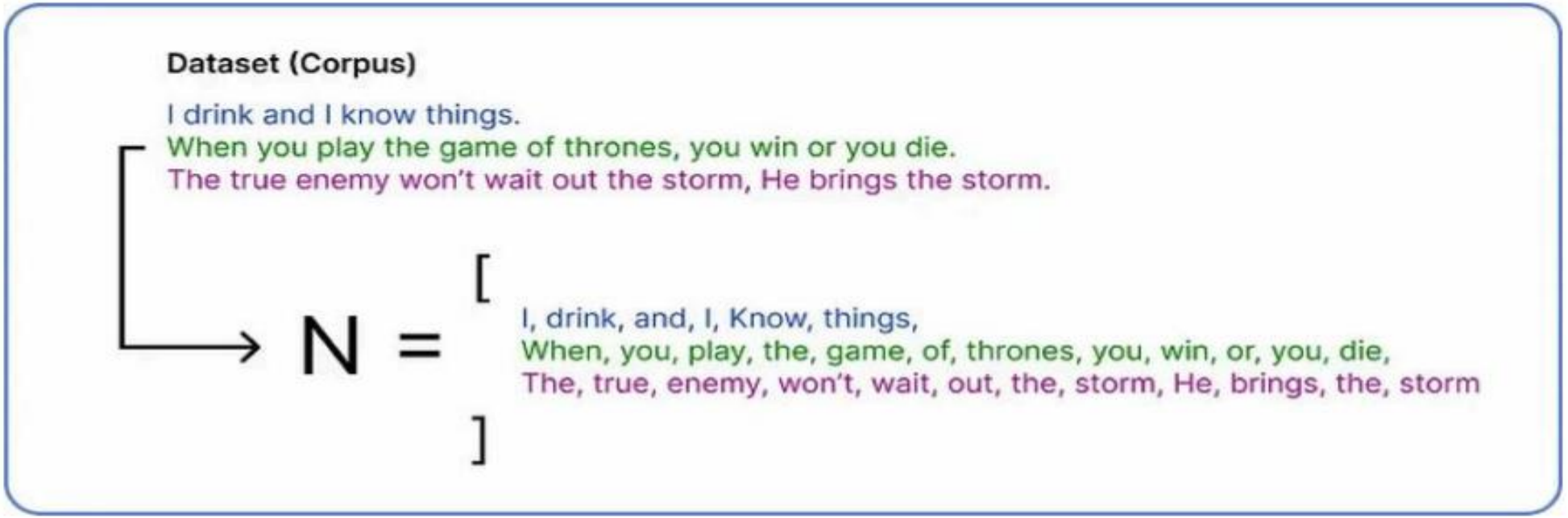
Step 2— Finding Vocab Size

The vocabulary size determines the total number of **unique words** in our dataset. It can be calculated using the below formula, where **N** is the total number of words in our dataset.

$$vocab\ size = count(set(N))$$

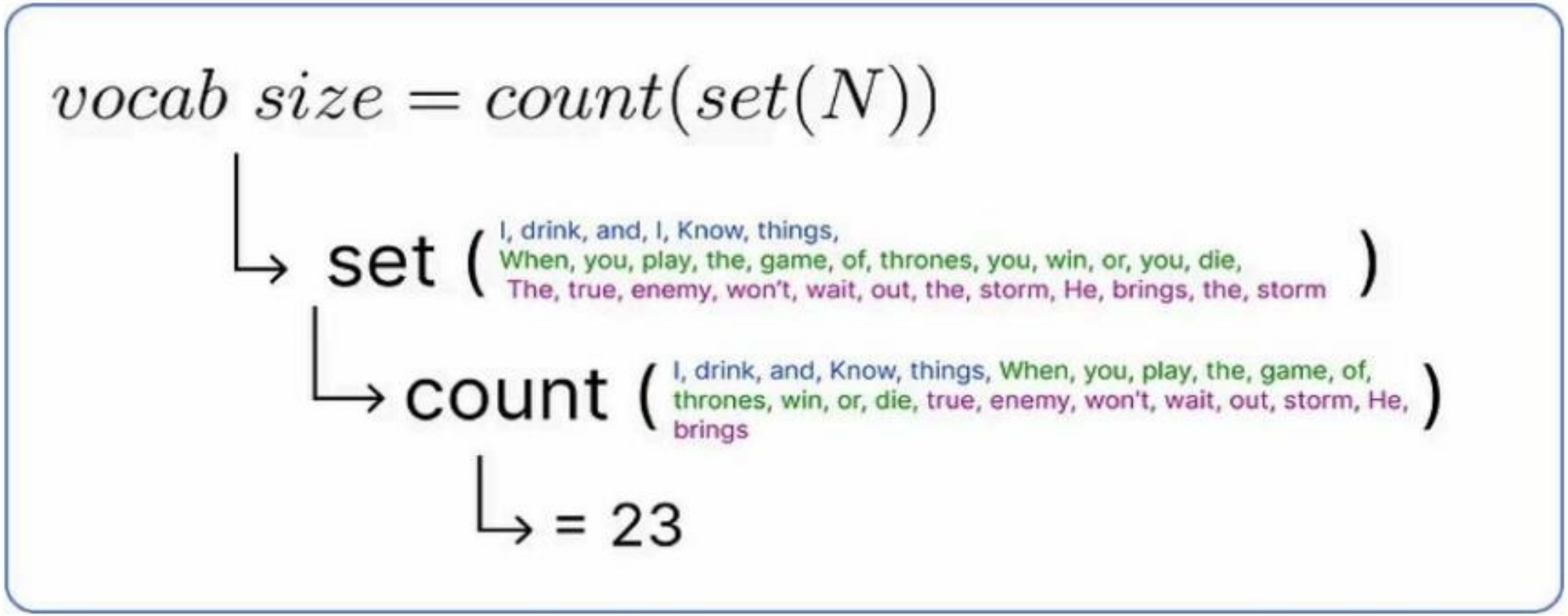
vocab_size formula where N is total number of words

In order to find N, we need to break our dataset into individual words.



calculating variable **N**

After obtaining N, we perform a set operation to remove duplicates, and then we can count the unique words to determine the vocabulary size.



finding vocab size

Therefore, the vocabulary size is **23**, as there are **23** unique words in our dataset.

Step 3 — Encoding

Now, we need to assign a unique number to each unique word.

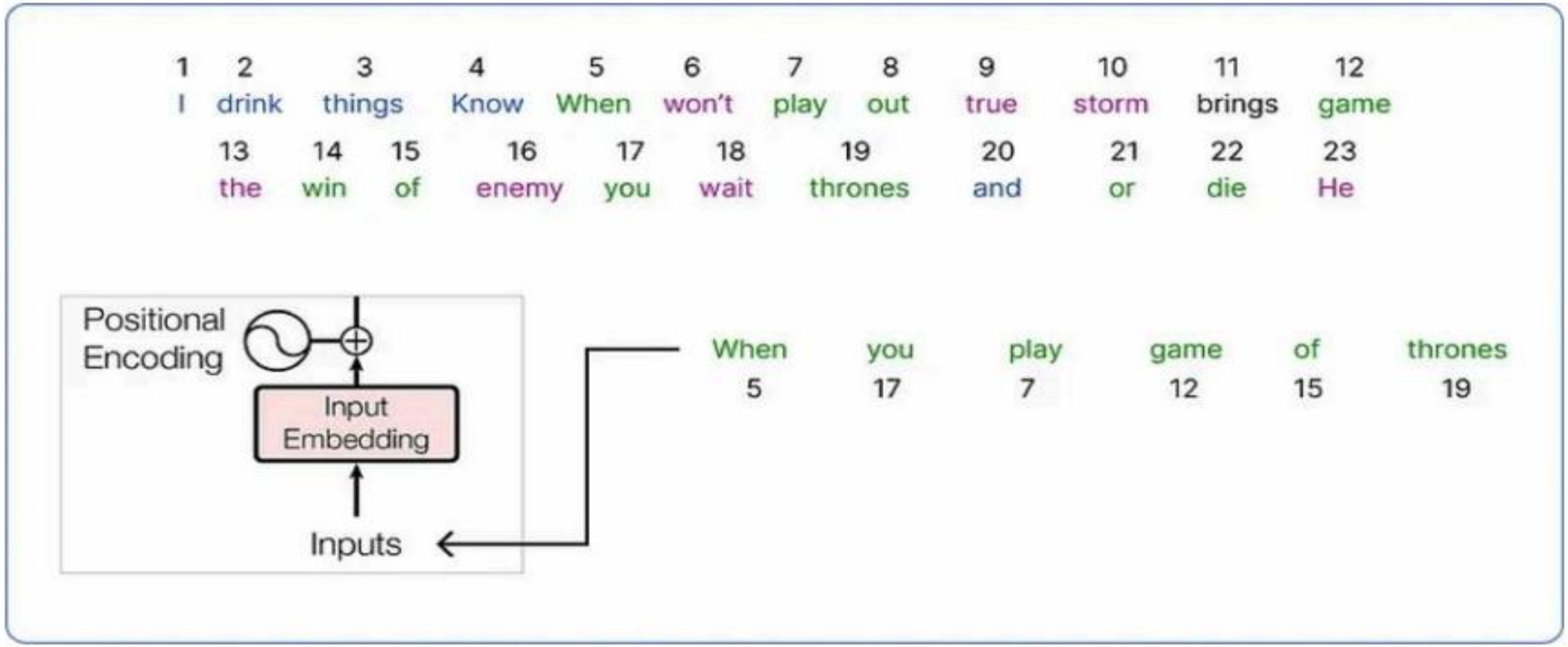
1	2	3	4	5	6	7	8	9	10	11	12
I	drink	things	Know	When	won't	play	out	true	storm	brings	game

13	14	15	16	17	18	19	20	21	22	23
the	win	of	enemy	you	wait	thrones	and	or	die	He

encoding our unique words

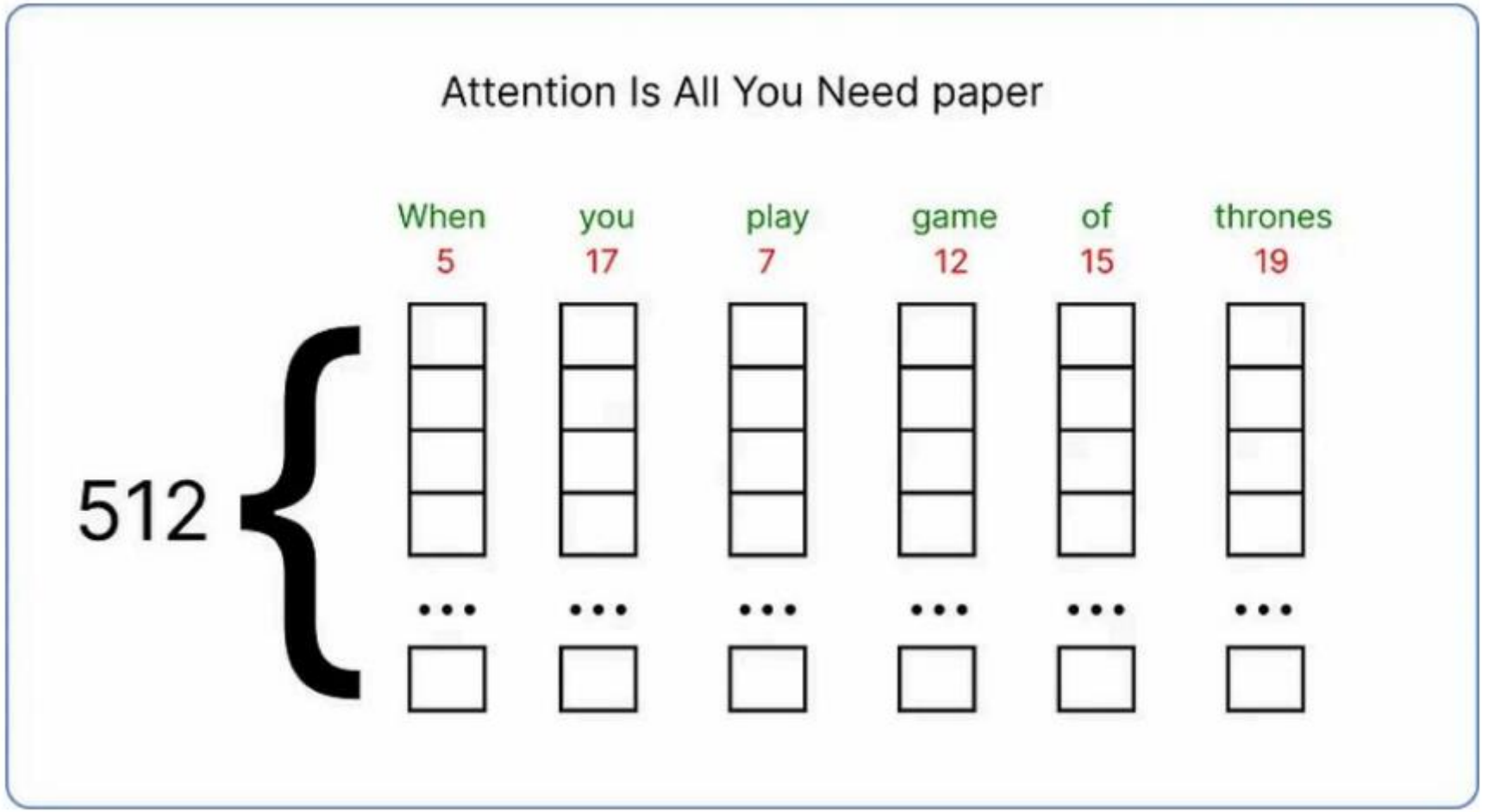
Step 4 — Calculating Embedding

Let's select a sentence from our corpus that will be processed in our transformer architecture.



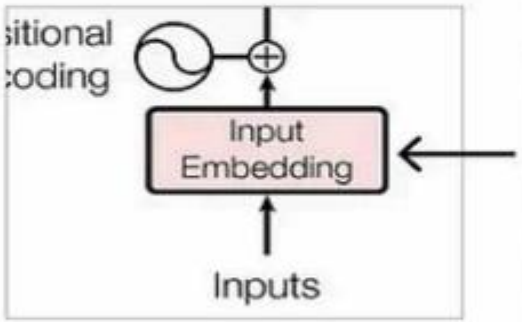
Input sentence for transformer

We have selected our input, and we need to find an embedding vector for it. The original paper uses a **512-dimensional embedding vector** for each input word.



Original Paper uses 512 dimension vector

Since, for our case, we need to work with a smaller dimension of embedding vector to visualize how the calculation is taking place. So, we will be using a dimension of 6 for the embedding vector.



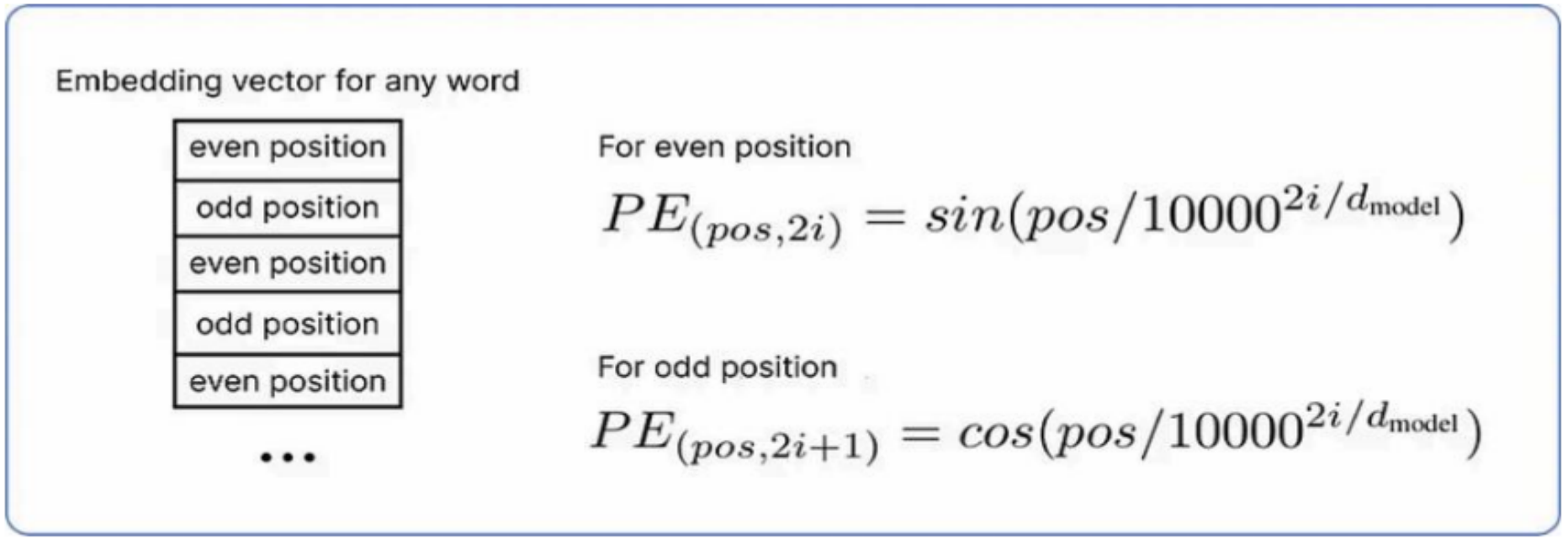
When	you	play	game	of	thrones
5	17	7	12	15	19

e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75
0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.21

Embedding vectors of our input

Step 5 — Calculating Positional Embedding

Now we need to find positional embeddings for our input. There are two formulas for positional embedding depending on the position of the ith value of that embedding vector for each word.



Positional Embedding formula

When
5

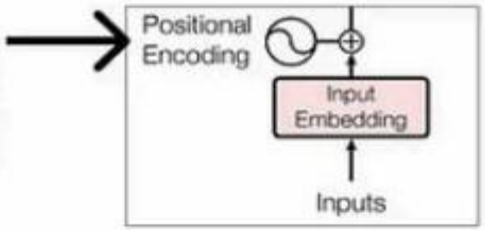
i	e1	Position	Formula	p1
0	0.79	Even	$sin(0/10000^{(2*0/6)})$	0
1	0.6	Odd	$cos(0/10000^{(2*1/6)})$	1
2	0.96	Even	$sin(0/10000^{(2*2/6)})$	0
3	0.64	Odd	$cos(0/10000^{(2*3/6)})$	1
4	0.97	Even	$sin(0/10000^{(2*4/6)})$	0
5	0.2	Odd	$cos(0/10000^{(2*5/6)})$	1
d (dim)	6			
POS	0			

Positional Embedding for word: **When**

Similarly, we can calculate positional embedding for all the words in our input sentence.

When	you	play	game	of	thrones
5	17	7	12	15	19

i	p1	p2	p3	p4	p5	p6
0	0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	1	0.0464	0.9957	0.1388	0.1846	0.9732
2	0	0.0022	0.0043	0.0065	0.0086	0.0108
3	1	0.0001	1	0.0003	0.0004	1
4	0	0	0	0	0	0
5	1	0	1	0	0	1
d (dim)	6	6	6	6	6	6
POS	0	1	2	3	4	5



Calculating Positional Embeddings of our input **(The calculated values are rounded)**

Step 6 — Concatenating Positional and Word Embeddings

After calculating positional embedding, we need to add word embeddings and positional embeddings.

When	you	play	game	of	thrones
5	17	7	12	15	19

Position Embedding Matrix

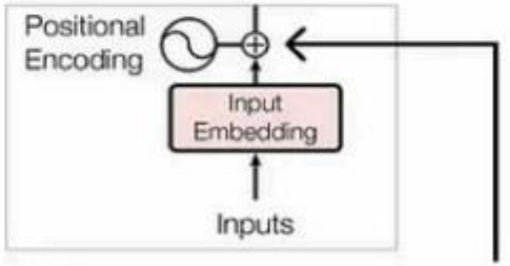
p1	p2	p3	p4	p5	p6
0	0.8415	0.9093	0.1411	-0.7568	-0.9589
1	0.0464	0.9957	0.1388	0.1846	0.9732
0	0.0022	0.0043	0.0065	0.0086	0.0108
1	0.0001	1	0.0003	0.0004	1
0	0	0	0	0	0
1	0	1	0	0	1

+

Word Embedding Matrix

e1	e2	e3	e4	e5	e6
0.79	0.38	0.01	0.12	0.88	0.6
0.6	0.12	0.51	0.6	0.41	0.33
0.96	0.06	0.27	0.65	0.79	0.75
0.64	0.79	0.31	0.22	0.62	0.48
0.97	0.9	0.56	0.07	0.5	0.94
0.2	0.74	0.59	0.37	0.7	0.21

=



ep1	ep2	ep3	ep4	ep5	ep6
0.79	1.22	0.92	0.26	0.12	-0.36
1.6	0.17	1.51	0.74	0.59	1.3
0.96	0.06	0.27	0.66	0.8	0.76
1.64	0.79	1.31	0.22	0.62	1.48
0.97	0.9	0.56	0.07	0.5	0.94
1.2	0.74	1.59	0.37	0.7	1.21

concatenation step

This resultant matrix from combining both matrices (**Word embedding matrix** and **positional embedding matrix**) will be considered as an input to the encoder part.