Unveiling the Mystery of Deep Learning: Past, Present, and Future

Dr. Elham Barezi,
AI Research scientist

Co-Sponsored by Rosen Center for Advanced Computing (RCAC), and IPAI

Spring and Summer 2025

PURDUE UNIVERSITY

Course Outline

1. History and Basics of DNN
   a. From traditional ML to DNN
2. Fundamental deep learning: from discriminative to generative
   a. CNN, RNN, Autoencoders, attention,
   b. Deep learning for Representation Learning and feature extraction
   c. Discriminative vs generative deep learning: VAE, GAN, Diffusion Models
3. Transformers Era
   a. self-attention, encoders, decoders, masking,
   b. self attention vs old attention
   c. Transformers for other modalities: text, image, video, speech,
4. LLMs in Practice
   a. Prompt Engineering Methods: COT, TOT, Self-Consistency, RAG, Agents,
   b. Fine-tuning Methods: instruct tuning, RLHF, Adapters like LORA,
   c. Deep learning for different domains
5. AI safety and Governance

Course Outline-first session-March 5th 2025

1. History and Basics of DNN
    a. AI hypes and winters
    b. Deep learning from 1950s
    c. From single neurons to deep networks
    d. Deep learning challenges solved from 1950-present
        i. Model overfitting
        ii. Activation function saturation
        iii. Vanishing/exploding gradient
    e. Deep learning weaknesses

Course Outline-Second Session-April 18th 2025

2. Fundamental deep learning models, from discriminative to generative:

a. CNN,

b. RNN,

c. Earlier version of **attention**,

d. Deep learning for **Representation Learning and feature extraction**

e. Earlier **Pre-Training** models

f. Discriminative vs Generative deep learning

PURDUE UNIVERSITY

Course Outline-3rd session-July 14th 2025

1. From RNNs to Transformers

- Brief recap of:
  - Sequence modeling with RNNs
  - Traditional attention mechanisms
- Motivation for Transformers

2. Transformer Architecture Overview and Core components:

- Tokenization
- Self-attention
- Positional encoding: Sinusoidal vs learned positional encodings
- Layer normalization & residual connections

3. LLM Variants and Evolution

- Encoder-Only models,Architecture, Training and Prediction.
- Encoder-Decoder models,Architecture, Training and Prediction.
- Decoder-only models, Architecture, Training and Prediction.

4. LLM Tips and Tricks

- Context Window Size
- Inference and Next Token Prediction
- Selection Criteria for LLMs

Transformers and Language Models

PURDUE UNIVERSITY

Course Outline-3rd session-Transformers and LLMs:

1. From RNNs to Transformers

- Brief recap:
  - Sequence modeling with RNNs
  - Traditional attention mechanisms
- Motivation for Transformers

2. Transformer Architecture Overview and Core components:

- Tokenization
- Positional encoding: Sinusoidal vs learned positional encodings
- Layer normalization & residual connections
- Self-attention

3. LLM Variants and Evolution

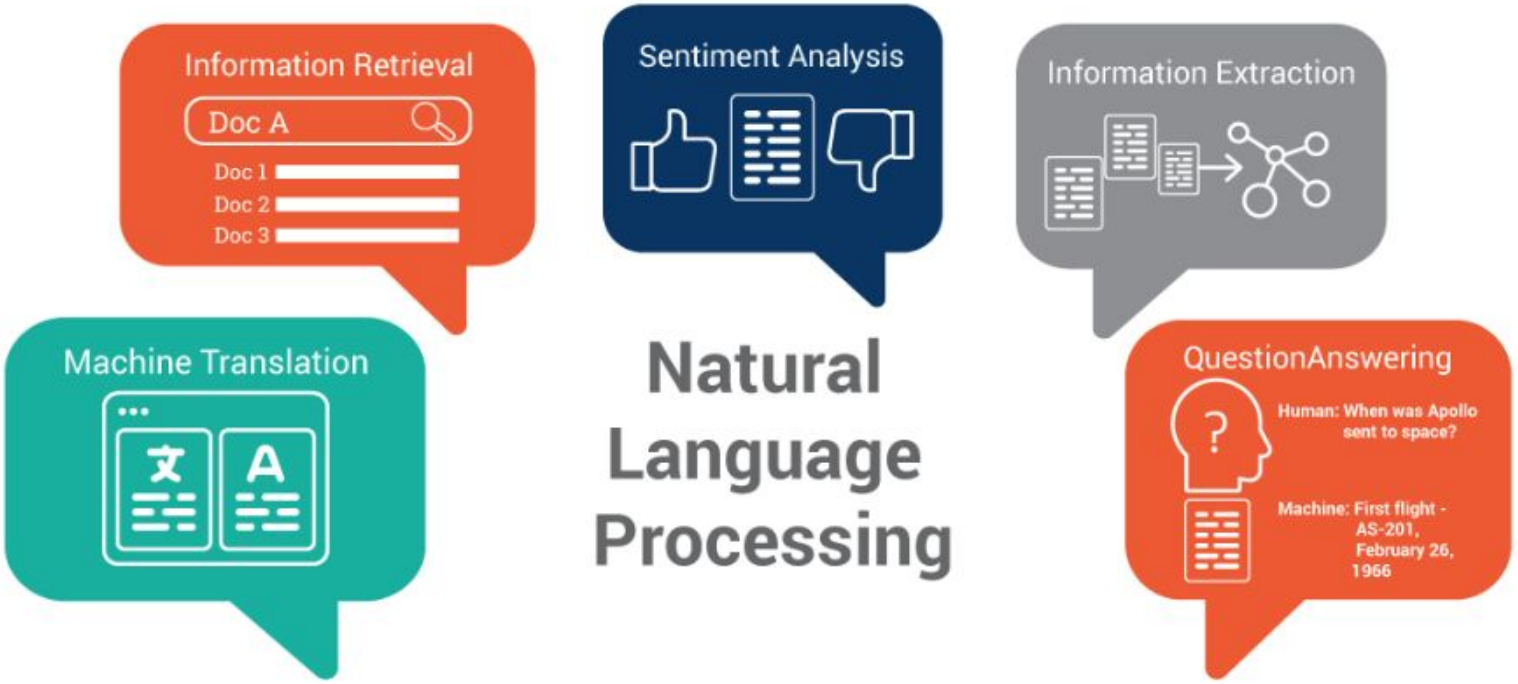
- Encoder-Only models,Architecture, Training and Prediction.
- Encoder-Decoder models,Architecture, Training and Prediction.
- Decoder-only models, Architecture, Training and Prediction.

4. LLM Tips and Tricks

- Context Window Size
- Inference and Next Token Prediction
- Selection Criteria for LLMs

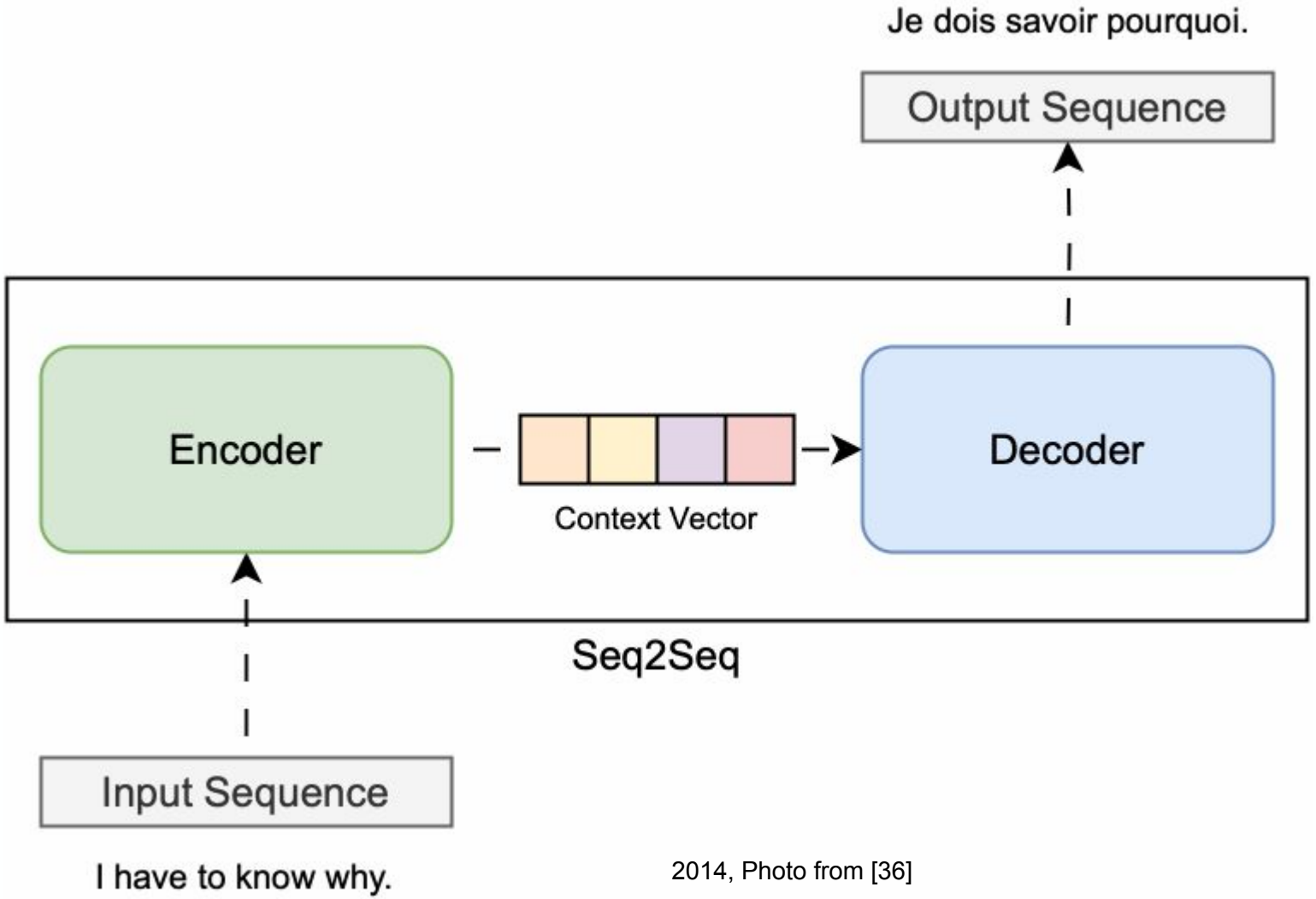
What is Natural Language Processing? (NLP)

- It is a branch of artificial intelligence (AI) that focuses on **creating computational models and algorithms** that allow machines to **comprehend**, **interpret**, and **produce** human language.



https://www.ontotext.com/blog/top-5-semantic-technology-trends-2017/

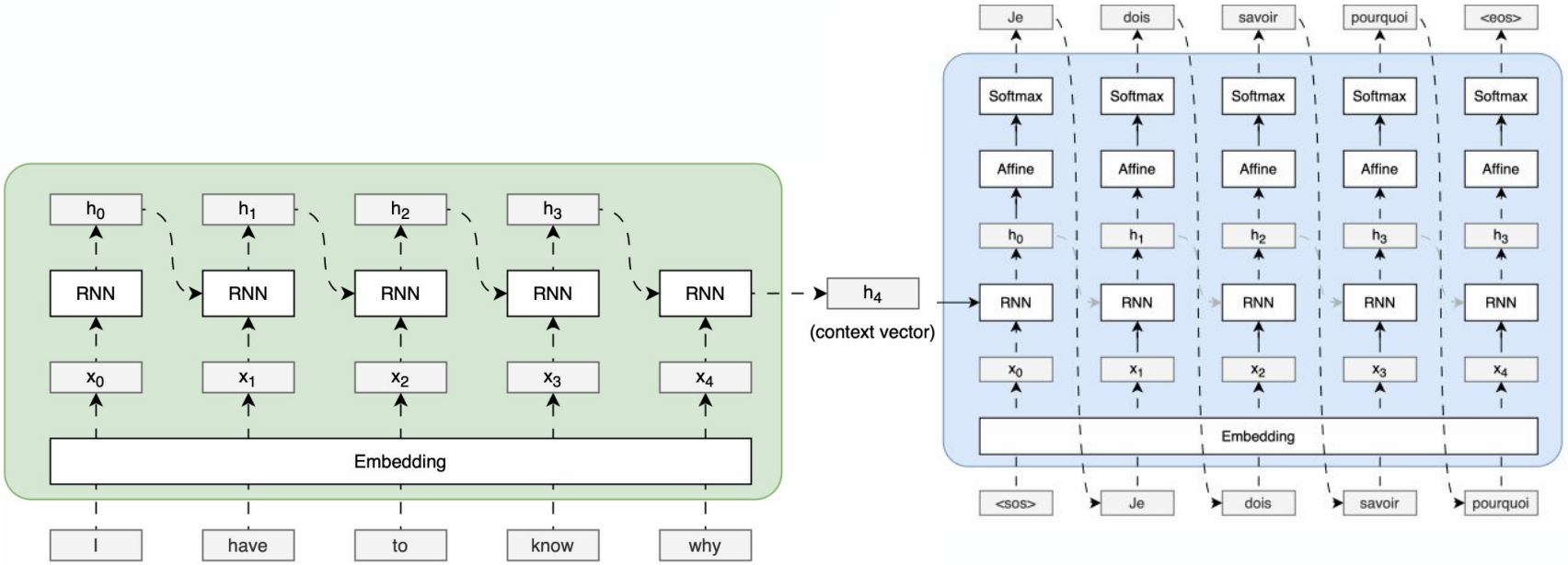
RNN to capture order in sequences



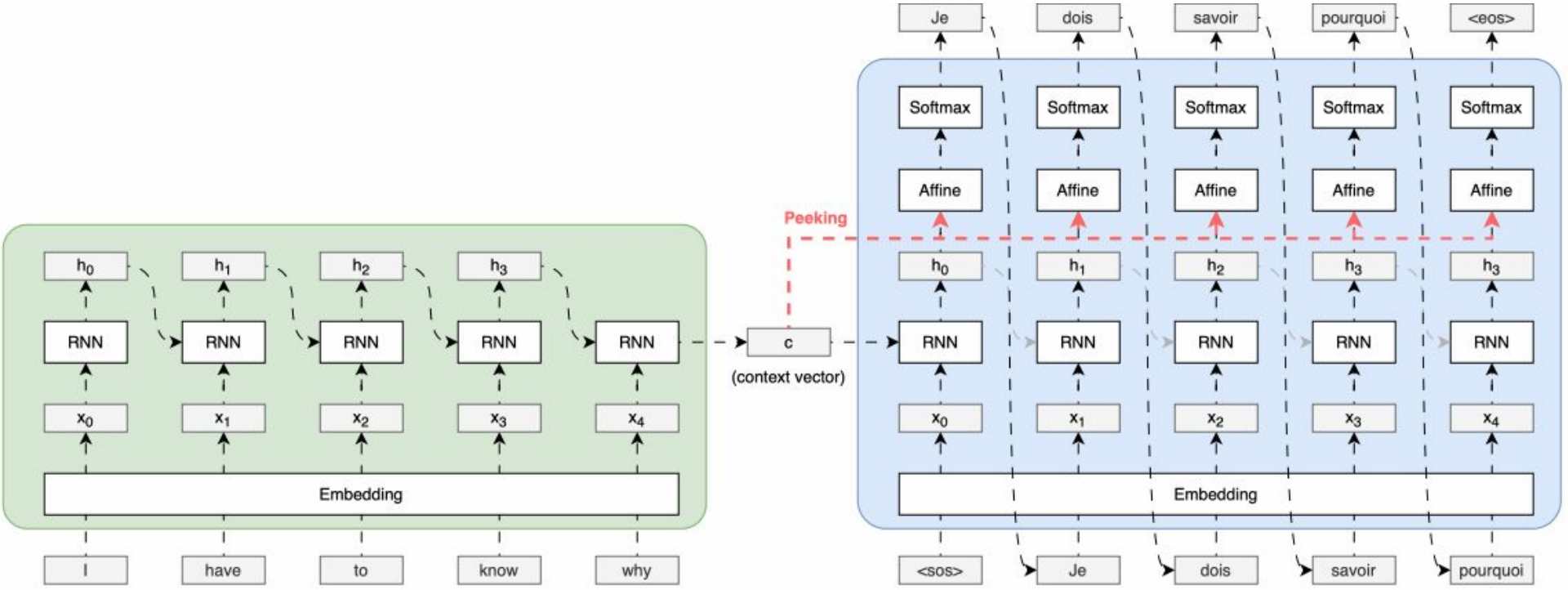
2014, Photo from [36]

RNN Shortcomings

- RNNs are poor at capturing **long-term dependencies**
- RNNs require recursion, and therefore have **poor throughput**.
  - Recursive calls and word-by-word processing of RNNs lead to many **interdependencies** that **can't be parallelized**.

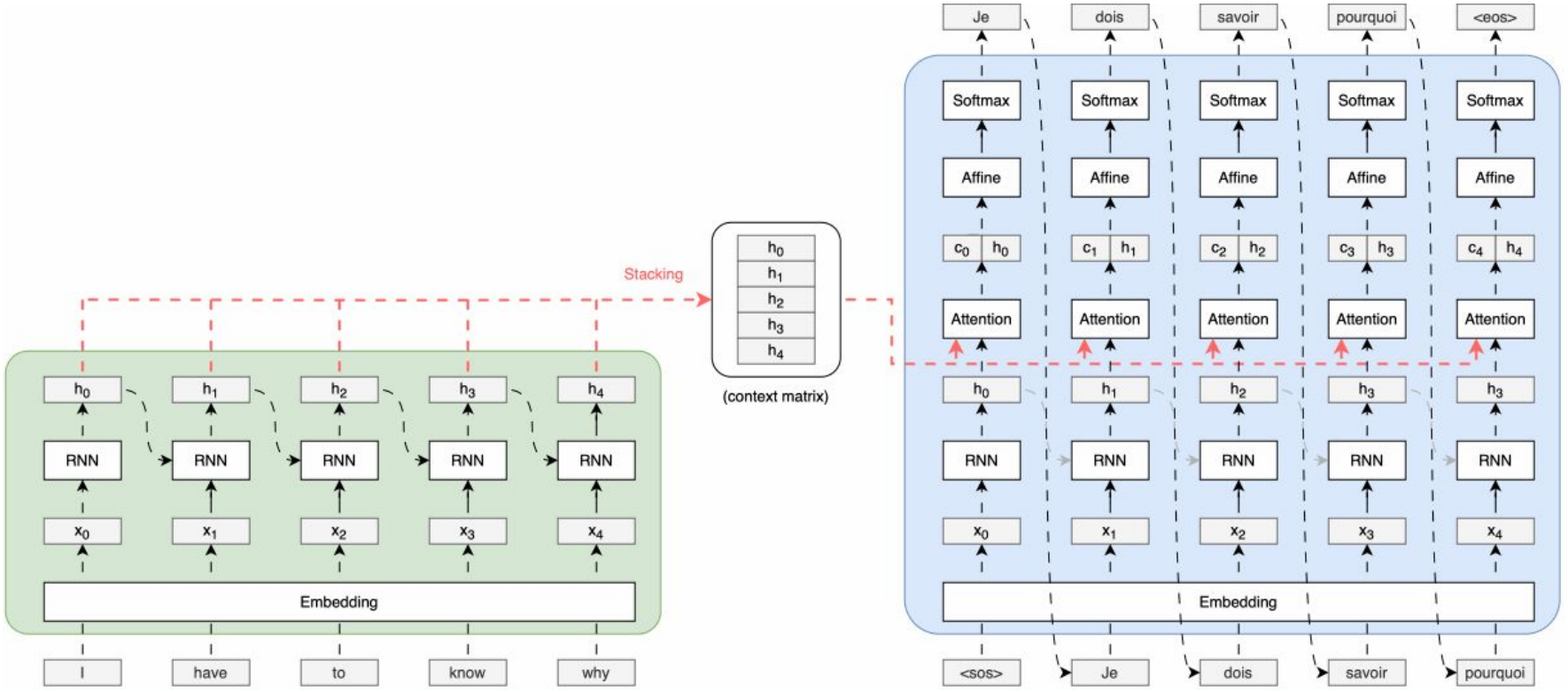


RNN Shortcomings



Attention for Machine Translation

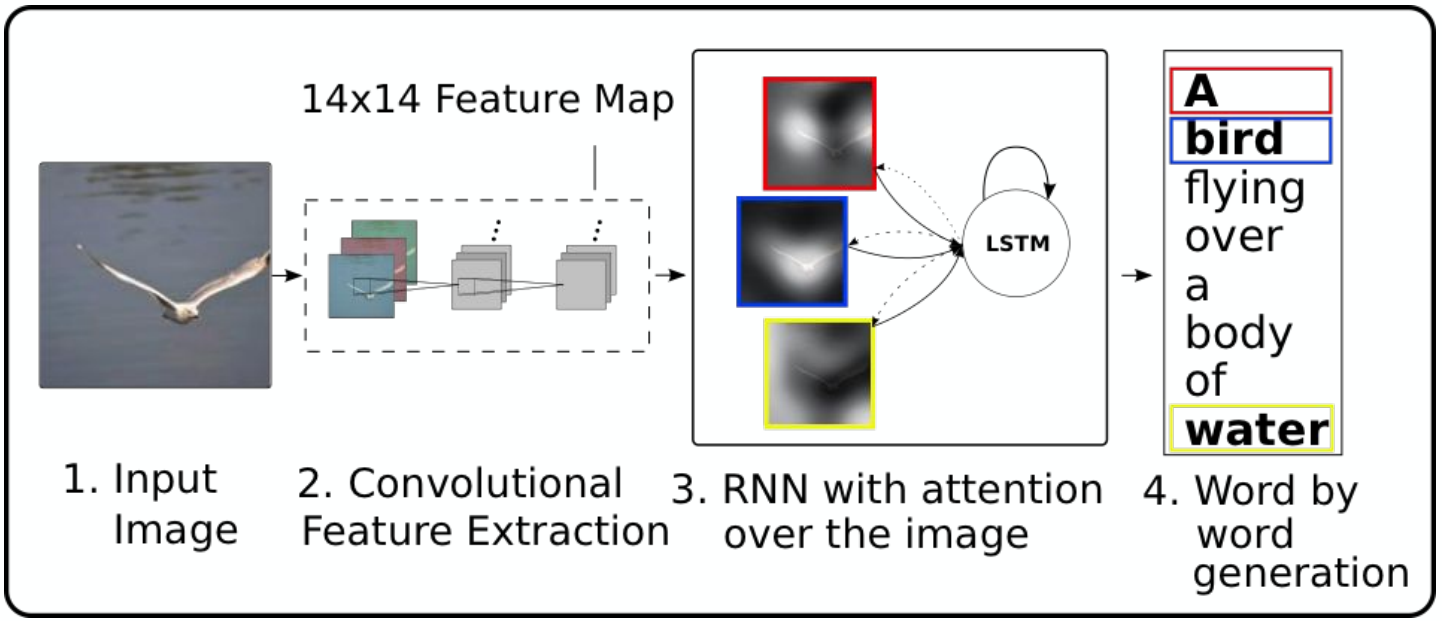
The limitations of RNNs led to attention mechanisms ⇒ **transformers** ⇒ GPT, BERT, and the AI we have now.



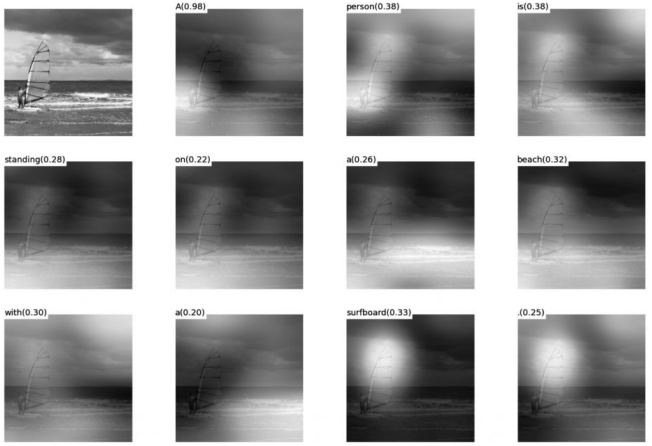
$$\text{score}(h_t, h_s) = \begin{cases} h_t^\top h_s & dot \\ h_t^\top W_a h_s & general \\ v_a^\top \tanh(W_a[h_t; h_s]) & concat \end{cases} \quad \Rightarrow \quad a_{ts} = \frac{\exp(\text{score}(h_t, h_s))}{\sum_{s'} \exp(\text{score}(h_t, h_{s'}))} \quad \Rightarrow \quad c_t = \sum_s \alpha_{ts} h_s$$

NEURAL MACHINE TRANSLATION, Bengio et al 2015

Attention for Caption Generation



Xu, Kelvin, et al. Arxiv'15



(b) A person is standing on a beach with a surfboard.

From RNNs to Transformers

RNNs process inputs **word-by-word**, transformers process the input sequence **as a whole,**

a. Solves **long-term dependency issue**

b. Every word in the sequence is treated equally in a transformer

RNNs use **recurrence** to capture word orders**,** Transformers use **positional embeddings**,

a. Enables **massively-parallel computation** for transformers

b. No longer have to worry about **gradient vanishing and explosions**

c. Also helps alleviate the **long-term dependency issue**

Transformers can scale up to large models using **unlabeled data,** despite RNNs.

Transformers-2017

- Transformer is made of **encoder-decoder** style architectures
- **Input**:
  - The **encoder** receives the input text that is to be processed, and the **decoder** receives the target text.
- **Output**:
  - **Encoders** are designed to **learn embeddings** that can be used for various predictive modeling tasks such as classification.
  - **Decoders** are designed to **generate new texts,**

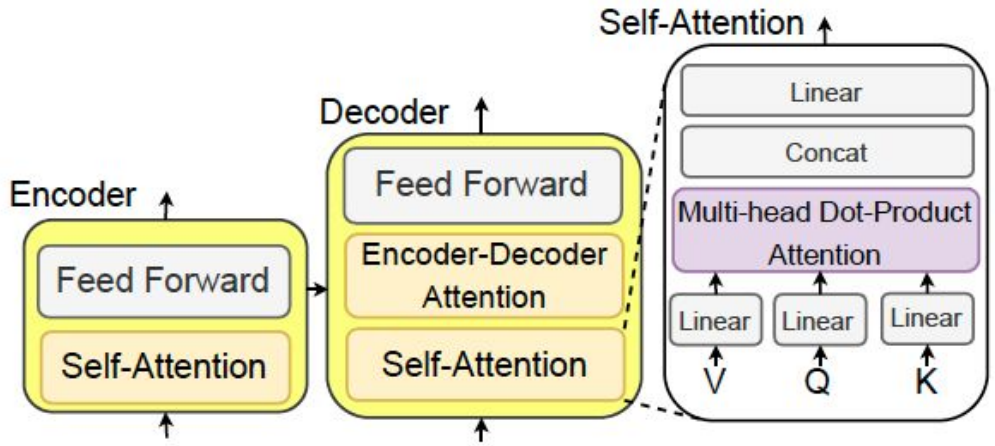


Photo from: "Attention is all you need"

Transformers

Limitations

- They can be **expensive to train** and require **large data sets**⇒ due to **Self-attention calculations.**
- The resulting models are also quite large, which makes it challenging to identify the **source of bias or inaccurate results**.
- Their complexity can also make it **difficult to interpret** their inner workings, hindering their **explainability and transparency**.

Course Outline-3rd session-Transformers and LLMs:

1. From RNNs to Transformers

- Brief recap of:
  Sequence modeling with RNNs (limitations: vanishing gradients, sequential computation bottlenecks).
  - Traditional attention mechanisms (Bahdanau, Luong).
- Motivation for Transformers

2. Transformer Architecture Overview and Core components:

- Tokenization
- Self-attention
- Positional encoding: Sinusoidal vs learned positional encodings
- Layer normalization & residual connections

3. LLM Variants and Evolution

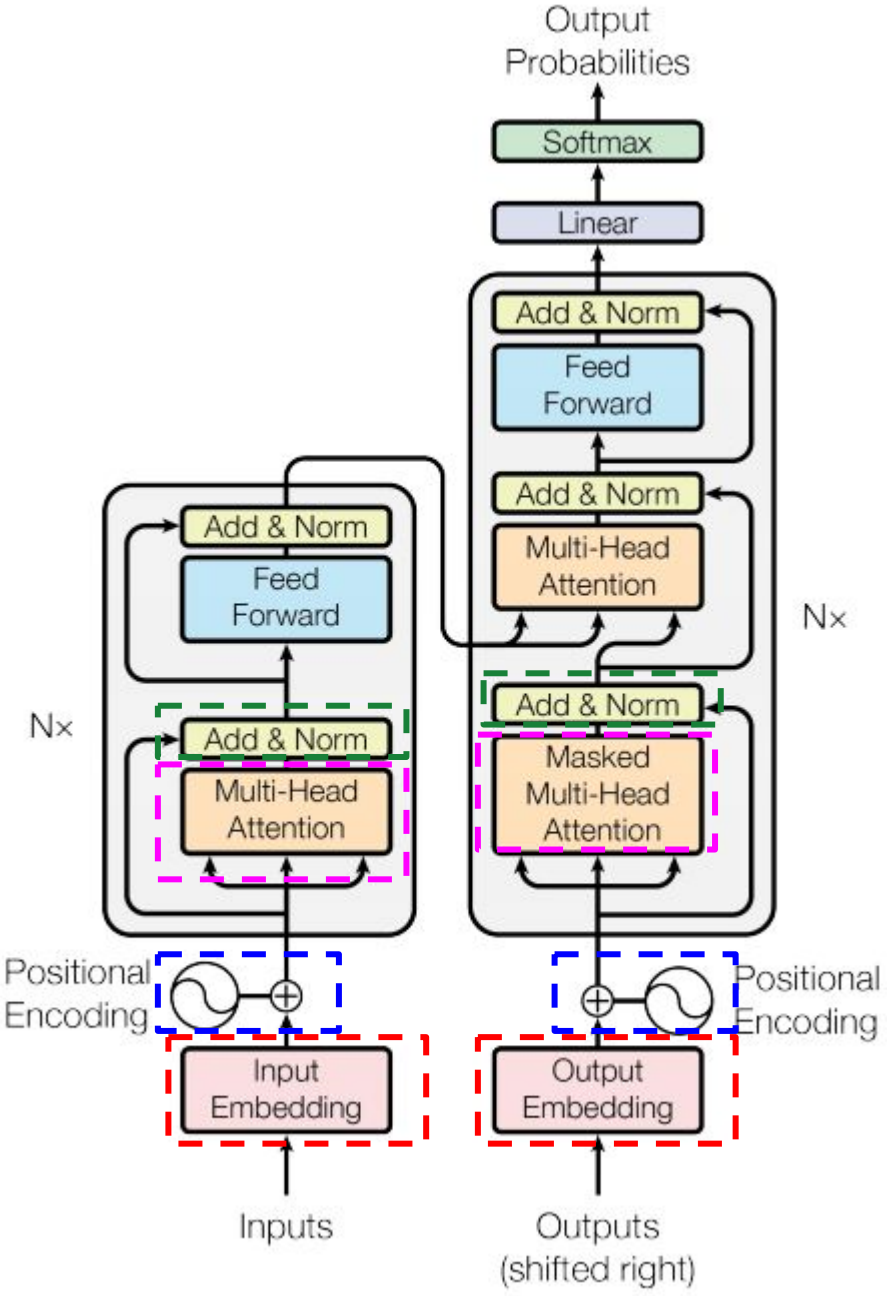
- Encoder-Only models,Architecture, Training and Prediction.
- Encoder-Decoder models,Architecture, Training and Prediction.
- Decoder-only models, Architecture, Training and Prediction.

4. LLM Tips and Tricks

- Context Window Size
- Inference and Next Token Prediction
- Selection Criteria for LLMs

Transformer Details

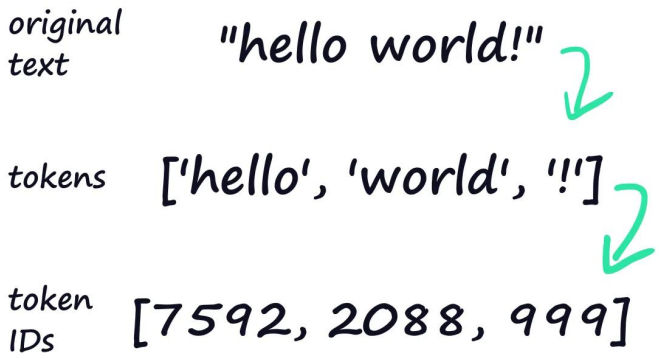
1. Tokenization and Embedding
2. Positional Encoding
3. Residual Addition an Normalization
4. Attention
   a. self vs cross attention,
   b. single vs multi head attention,



PURDUE UNIVERSITY

Tokenizer

We must convert words to machine readable data (numbers)!



We need a tokenizer to find text units(tokens) and convert them to numbers using a dictionary of units and their matching numerical IDS

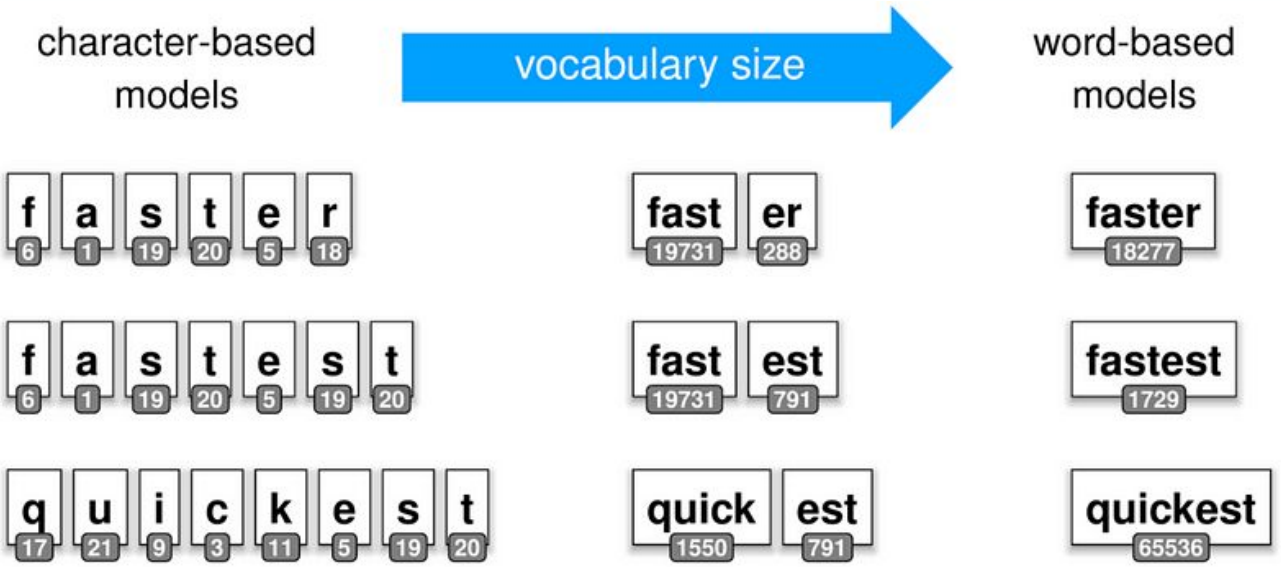
➥ A Tokenizer: each tokenizer comes with its dictionary of tokens and their matching IDs.

Token

Word level: Human readable and fast, but a big dictionary that cannot handle rare/new words.

Subword level: Smaller Vocabulary, while keeping semantic parts intact

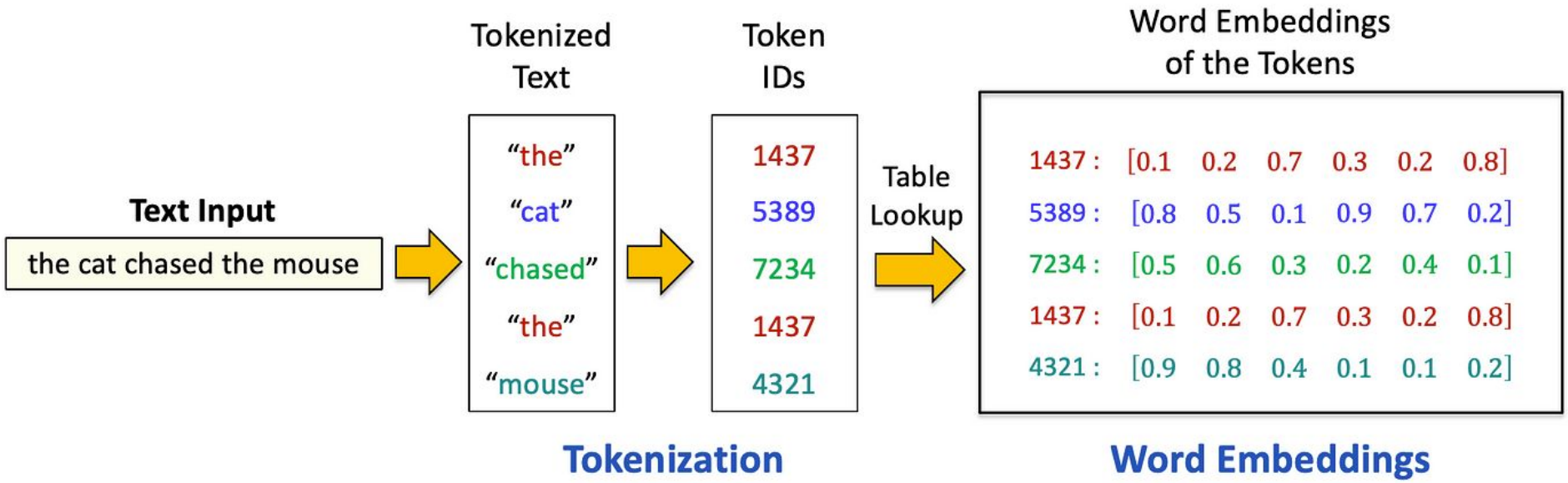
Character level: very small dictionary, but does not capture semantics, and generates long sequences.



PURDUE UNIVERSITY

Embedding

- Embedding is a **vector representation of a token** that **captures its semantic meaning** in a continuous space.
- **Calculation:** The embedding matrix is learned during **pretraining** of the model.
- Embedding matrix is of size **vocabulary*embedding_size**.



https://medium.com/@lmpo/tokenization-and-word-embeddings-the-building-blocks-of-advanced-nlp-c203b78bfd07

PURDUE UNIVERSITY

Positional encoding

- Transformers process all tokens **in parallel** (unlike RNNs),
- Transformers do not have recurrence or convolution information**= no order info**,
- How does the model know that token A came before token B?

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

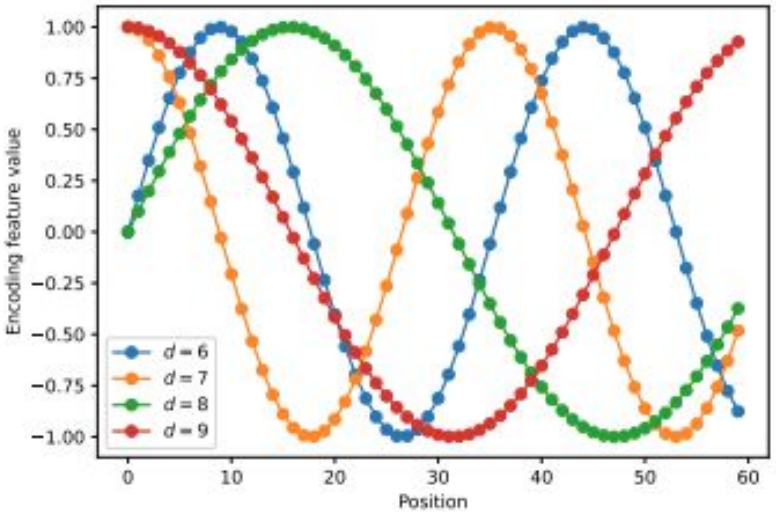
- pos is the position of the token in the sequence (0-indexed).
- i is the dimension index within the vector $0 \le i < d/2$.
- d_{model} is the dimensionality of the model's input embeddings.
- Even indices get *sine*, odd indices get *cosine*.

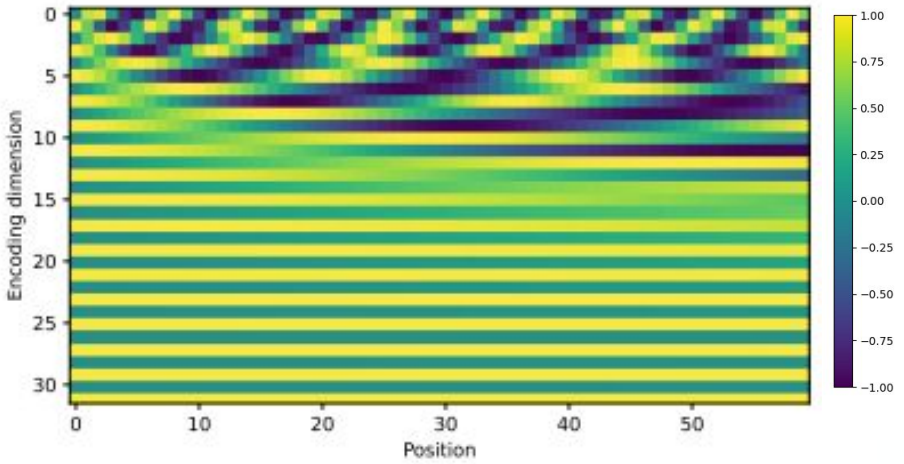
$$\mathbf{p}_t = \begin{bmatrix} \sin\left(\frac{t}{\lambda_1}\right) \\ \cos\left(\frac{t}{\lambda_1}\right) \\ \sin\left(\frac{t}{\lambda_2}\right) \\ \cos\left(\frac{t}{\lambda_2}\right) \\ \vdots \\ \sin\left(\frac{t}{\lambda_{d_{model}/2}}\right) \\ \cos\left(\frac{t}{\lambda_{d_{model}/2}}\right) \end{bmatrix}$$

$$\lambda_i = 10\,000^{2i/d_{model}}$$

PURDUE UNIVERSITY

Positional Encoding Visualization





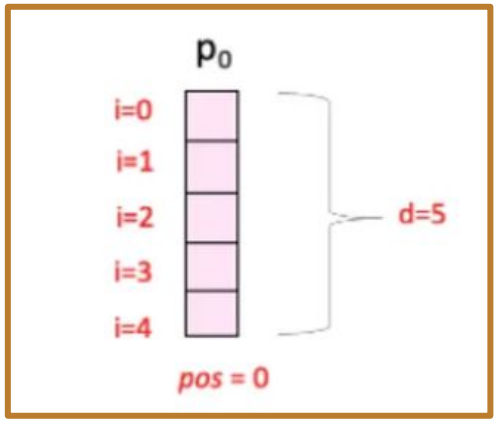
PURDUE UNIVERSITY

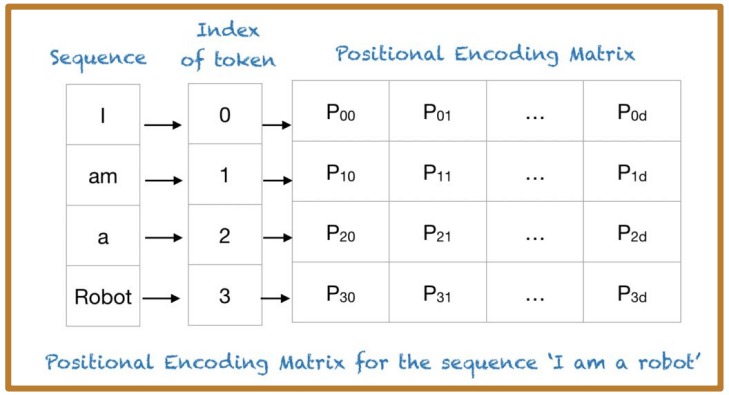
Positional Encoding

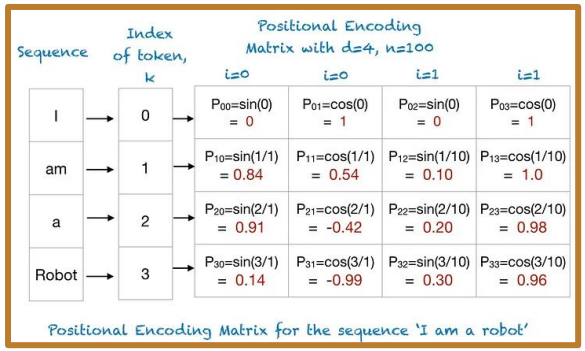
An Example

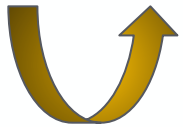
$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$









PURDUE UNIVERSITY

Image partially obtained from:
https://www.cjiawei.com/ox-hugo/20220730-014840_positional_encoding.png

Adding Residual and Normalization

1. Layer Normalization Stabilizes Training

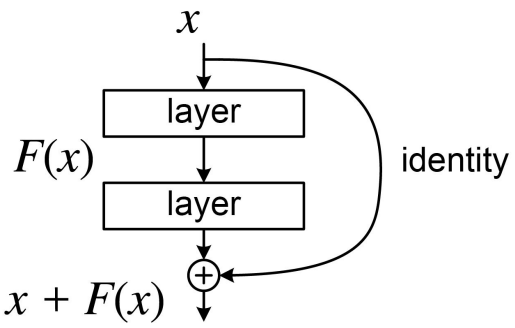
- Layer normalization helps reduce **internal covariate shift**, meaning the distribution of neuron activations stays more stable as training progresses.

2. Adding residual connection helps with Gradient Flow

- The **residual connection (Add)** makes it easier for gradients to flow back during training, especially in **deep networks**.

3. Adding residual connection preserves Original Information

- The residual connection **preserves the input signal**, even after passing through complex transformations.



PURDUE UNIVERSITY

Self-Attention

- Self-attention enables the model to weight the importance of different elements in an input sequence and adjust their influence on the output.
- This is especially important for language processing tasks, **where the meaning of a word can change based on its context within a sentence or document**.

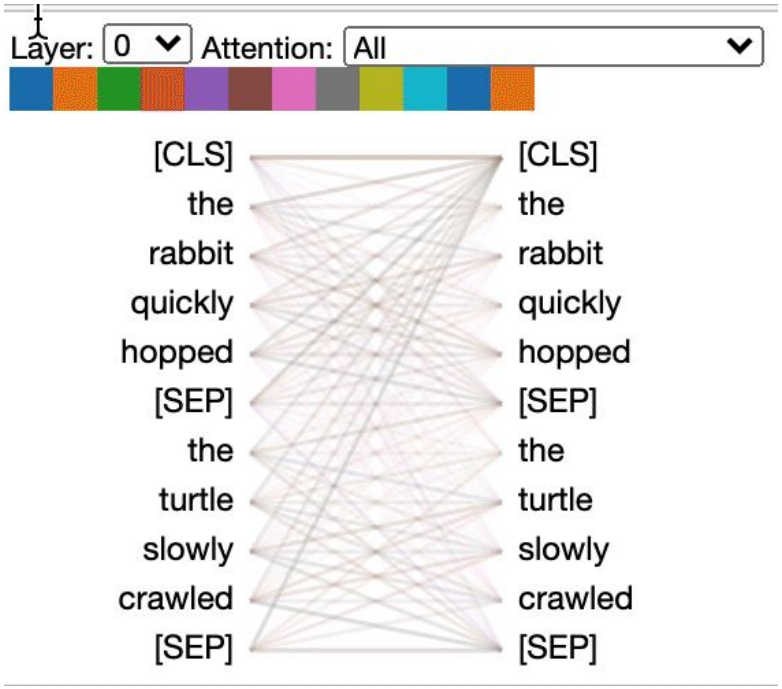
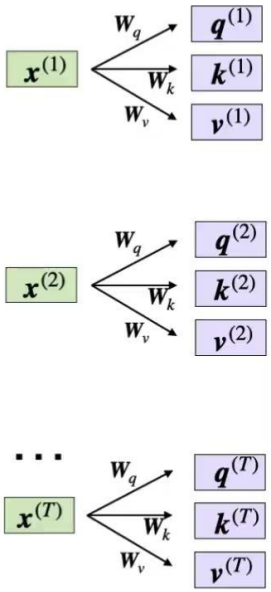


Photo from: https://github.com/jessevig/bertviz

Self-Attention



Video generated by canva.com

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

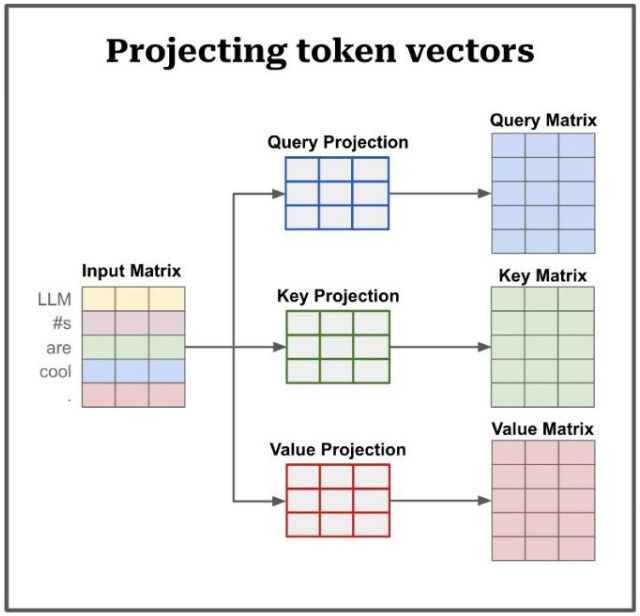
Q: query matrix, K: key matrix, V: value matrix,d_k: queries and keys of dimension

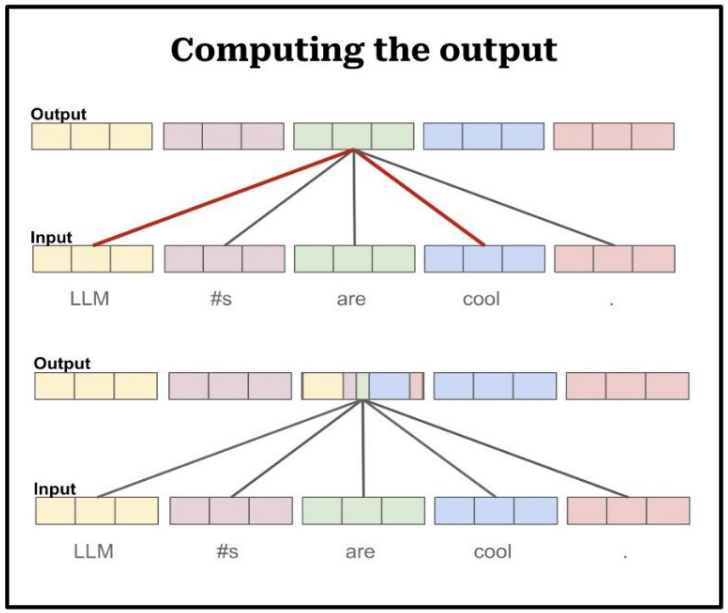
PURDUE UNIVERSITY

Attention Patterns

Masked Self-Attention

```python
class MaskedSelfAttention(nn.Module):

    def __init__(
        self,
        d,
        T,
        bias=False,
        dropout=0.2,
    ):
        super().__init__()
        self.d = d

        # key, query, value projections for all heads, but in a batch
        # output is 3X the dimension because it includes key, query and value
        self.c_attn = nn.Linear(d, 3*d, bias=bias)

        # causal mask to ensure that attention is only applied to
        # the left in the input sequence
        self.register_buffer("mask", torch.tril(torch.ones(T, T))
                                     .view(1, 1, T, T))

    def forward(self, x):
        B, T, _ = x.size() # batch size, sequence length, embedding dimensionality

        # compute query, key, and value vectors for all heads in batch
        # split the output into separate query, key, and value tensors
        q, k, v  = self.c_attn(x).split(self.d, dim=2) # [B, T, d]

        # compute the attention matrix, perform masking, and apply dropout
        att = (q @ k.transpose(-2, -1)) * (1.0 / math.sqrt(k.size(-1))) # [B, T, T]
        att = att.masked_fill(self.bias[:,:T,:T] == 0, float('-inf'))
        att = F.softmax(att, dim=-1)

        # compute output vectors for each token
        y = att @ v # [B, T, d]
        return y
```





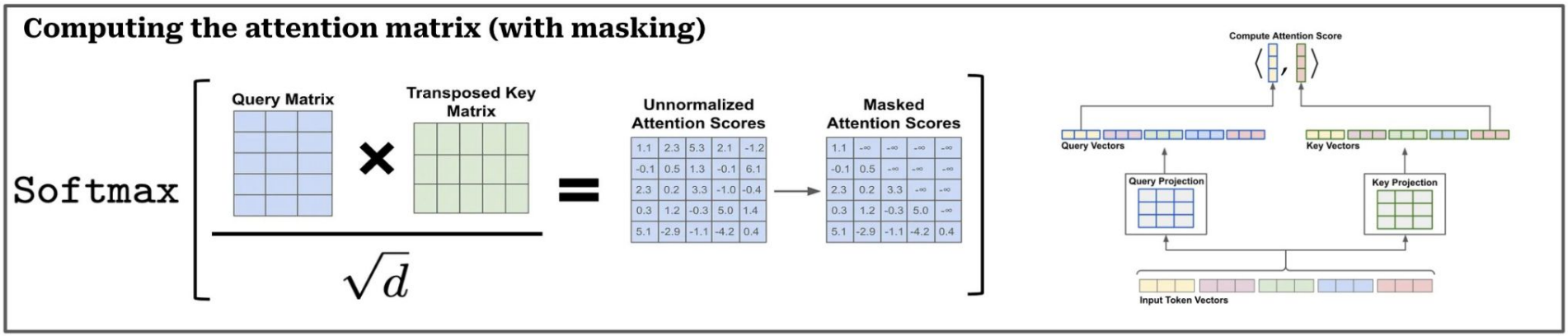
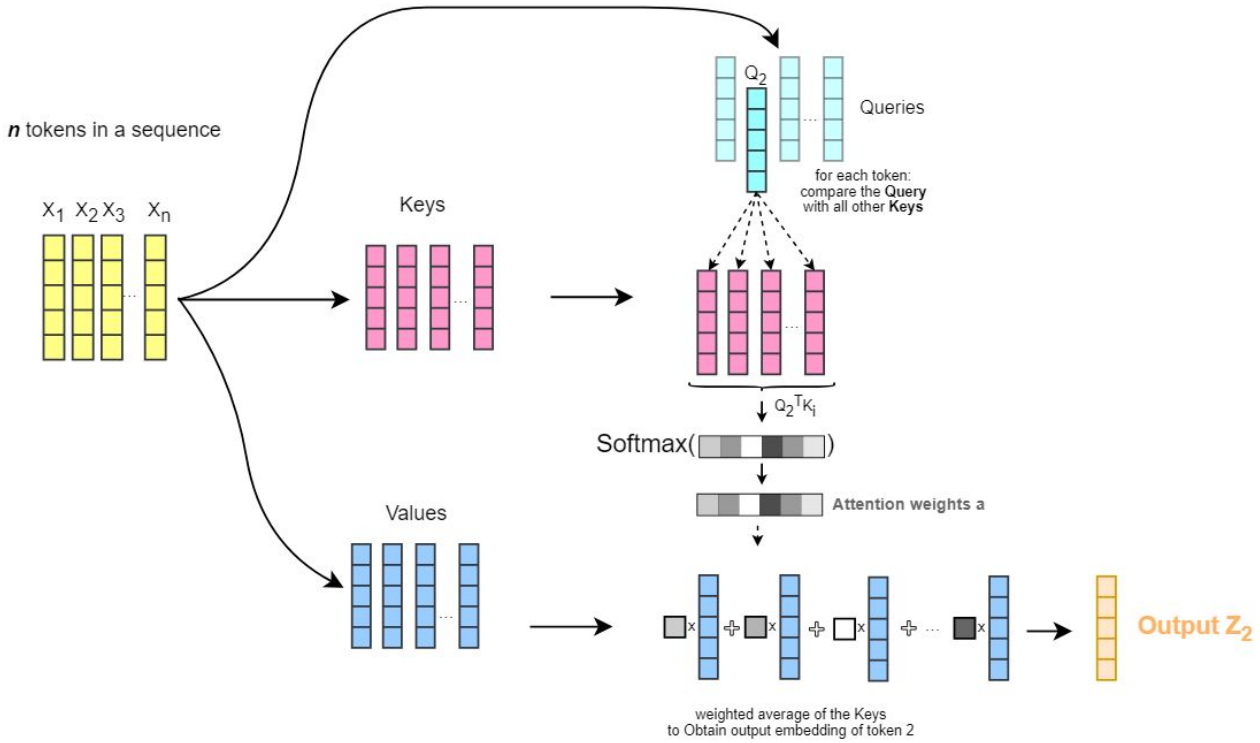


Photo taken from [17]

Self-Attention



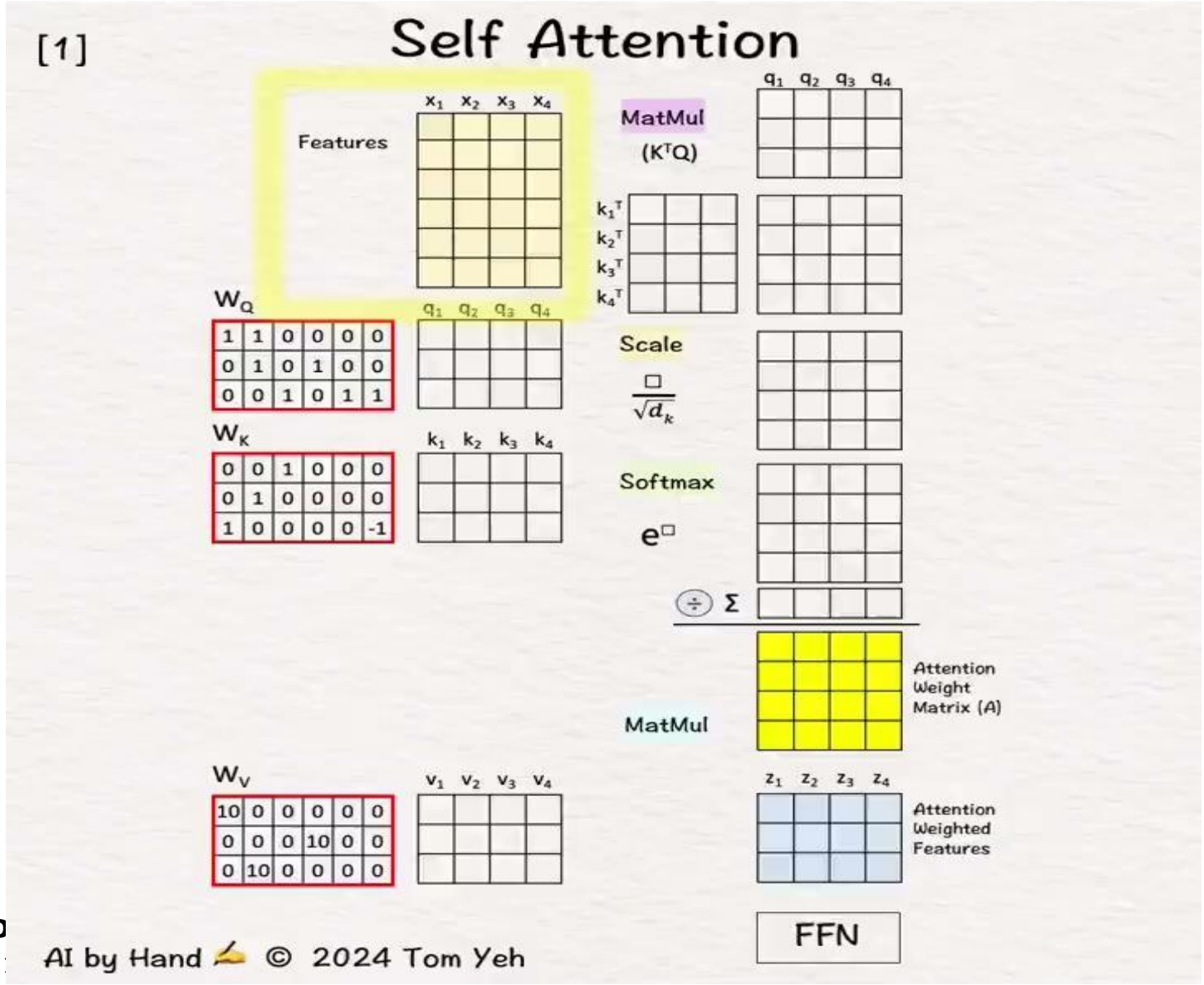
$$Attention(Q, K, V) = softmax(\frac{QK^T}{d_k})V$$

Q: query matrix, K: key matrix, V: value matrix,d_k: queries and keys of dimension

PURDUE UNIVERSITY

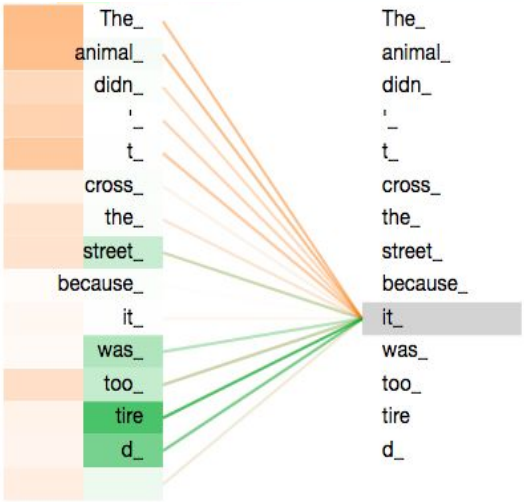
Photo from https://epichka.com/blog/2023/qkv-transformer/

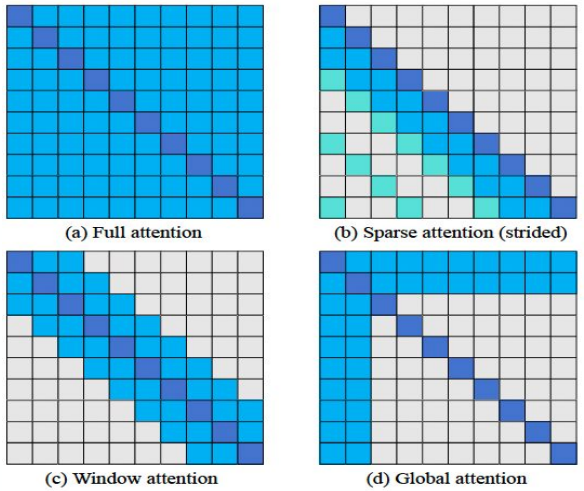
Attention Calculation Costs $O(N^2)$!



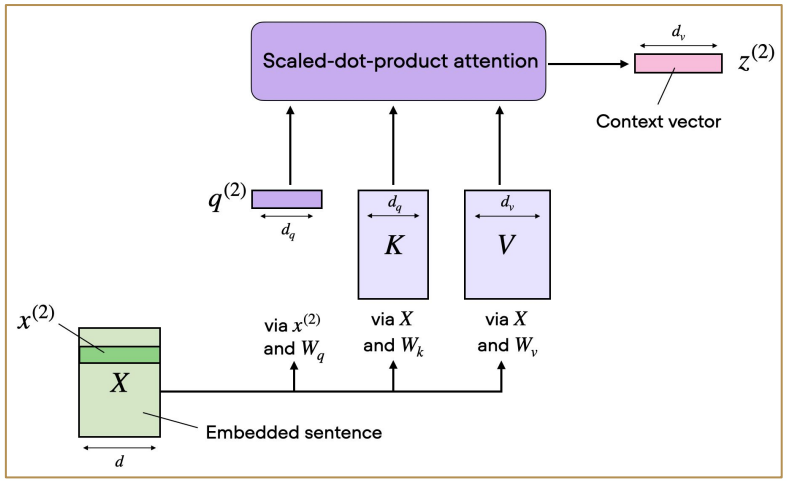
Self-Attention

- Standard self-attention computes interactions between **all token pairs**, which has:
  - **Time complexity**: O(N^2)
  - **Memory usage**: O(N^2)
- For long sequences (like in documents, videos, or genomics), full attention is too slow and memory-intensive.
- Parallel processing is needed for transformers.





Single vs Multi Head Attention



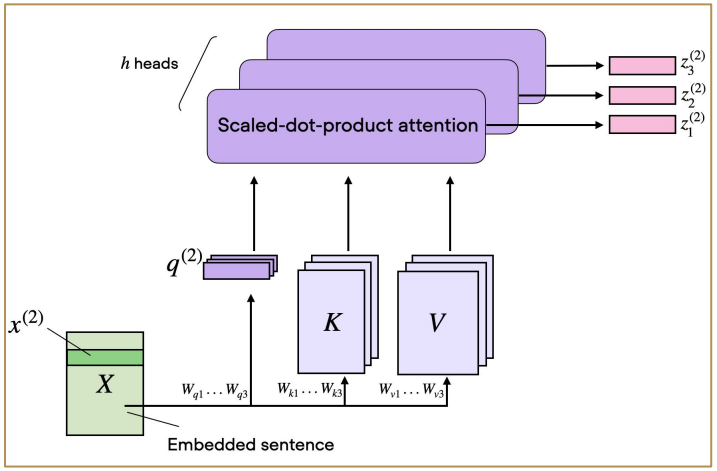


Photo taken from [44]

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \dots, \text{head}_h]\mathbf{W}_0$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$

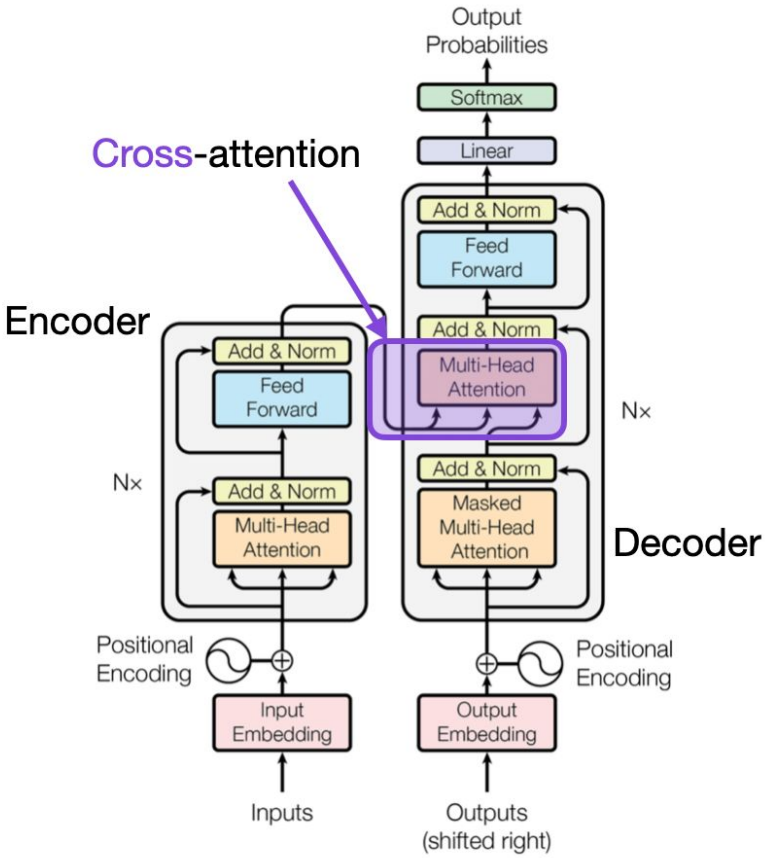
- **Captures multiple dependencies:** Detects **short-term and long-term relationships** in the data.
- **Learns diverse representations:** Different heads focus on **syntax, semantics, and meaning**.

PURDUE UNIVERSITY

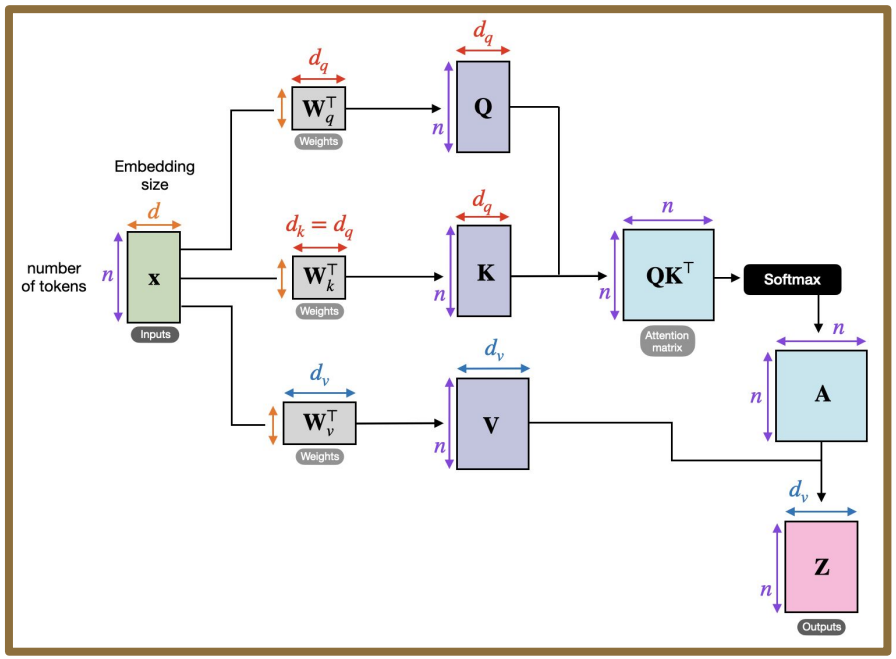
Cross-Attention

Key and Value: The sequence returned by the encoder module (the left 2 arrows)

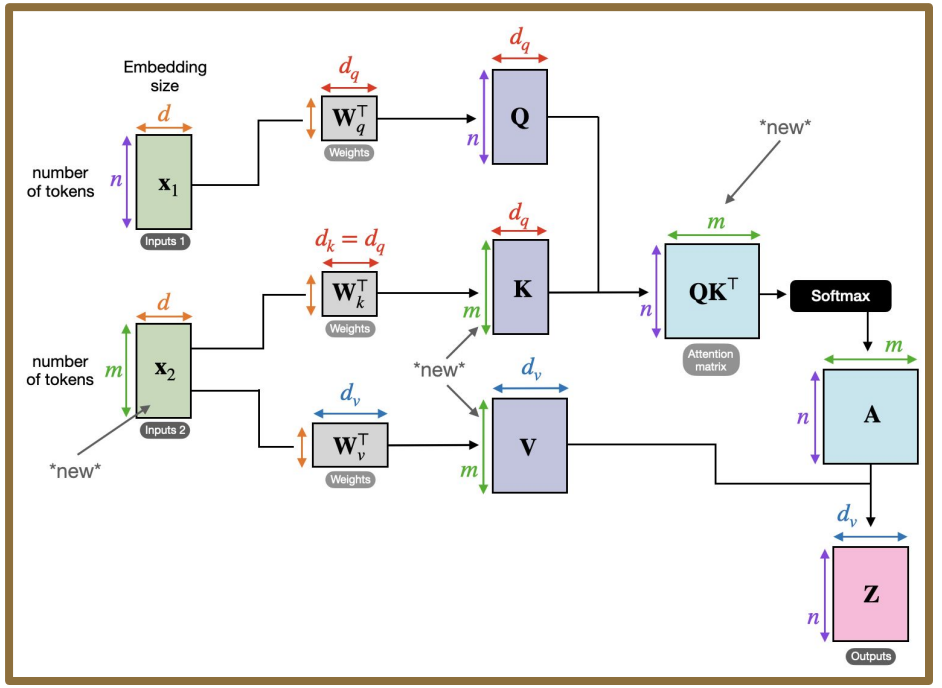
Query: The sequence generated by the decoder part (the right arrow)



Self-attention vs Cross-attention



- In self-attention, we work with the same input sequence.
- In cross-attention, we mix or combine two *different* input sequences.



PURDUE UNIVERSITY

Course Outline-3rd session-Transformers and LLMs:

1. From RNNs to Transformers

- Brief recap of:
  Sequence modeling with RNNs (limitations: vanishing gradients, sequential computation bottlenecks).
  - Traditional attention mechanisms (Bahdanau, Luong).
- Motivation for Transformers

2. Transformer Architecture Overview and Core components:

- Tokenization
- Self-attention
- Positional encoding: Sinusoidal vs learned positional encodings
- Layer normalization & residual connections

3. LLM Variants and Evolution

- Encoder-Only models,
- Encoder-Decoder models,
- Decoder-only models,

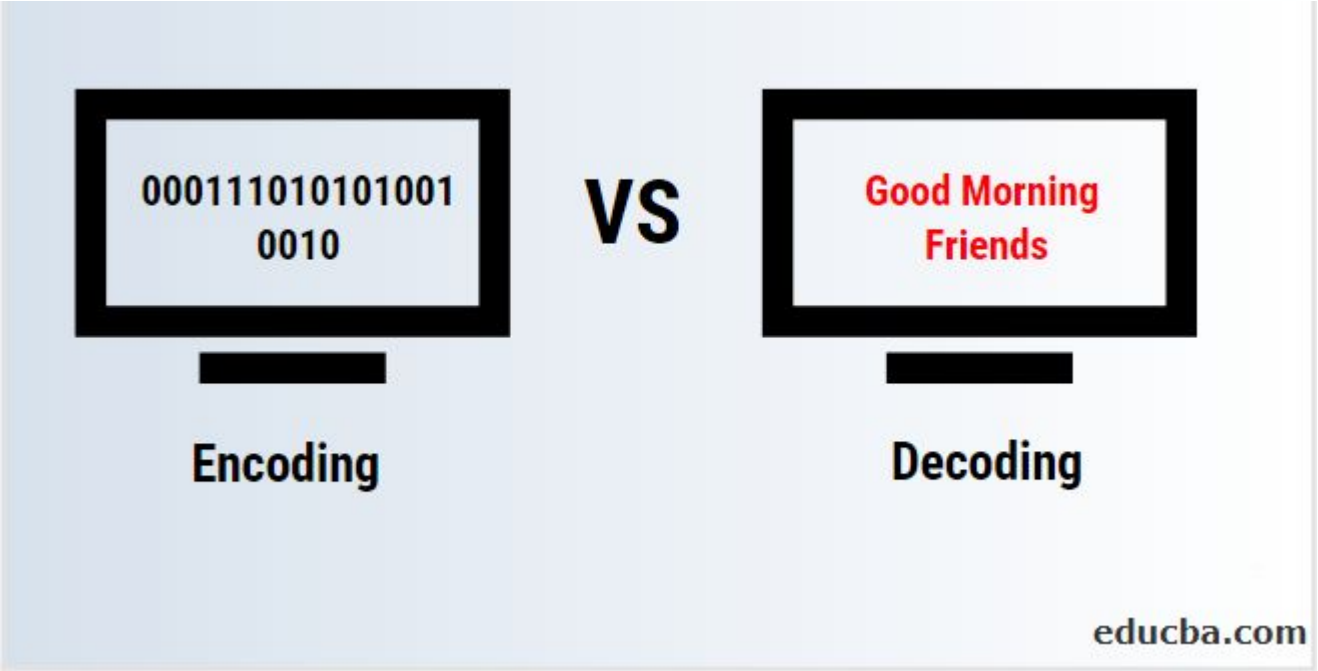
4. LLM Tips and Tricks

- Context Window Size
- Inference and Next Token Prediction
- Selection Criteria for LLMs

From Encoding to Decoding

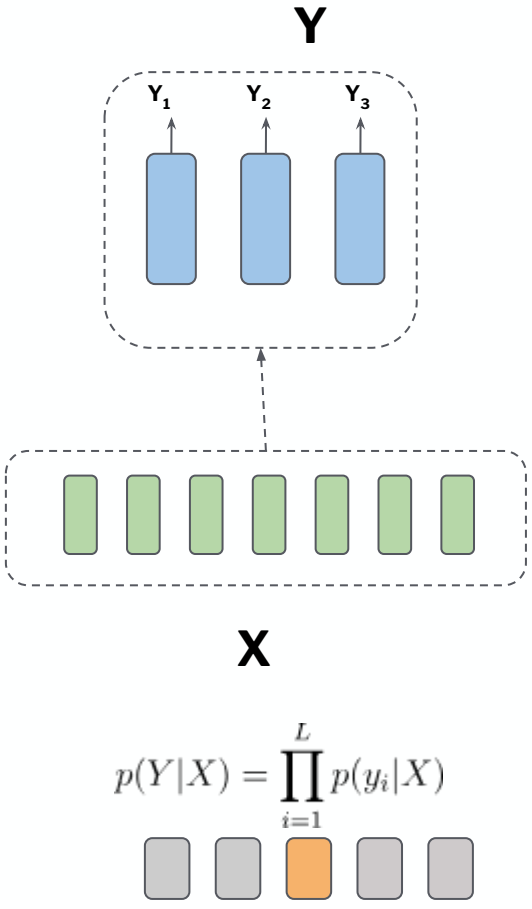
Encoding and Self Attention help to **gather context**.

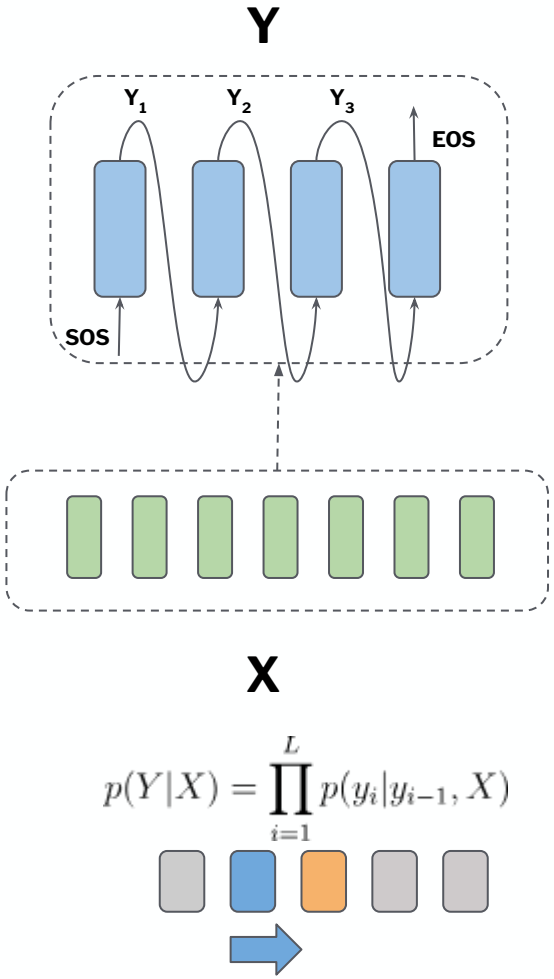
But for generating text, we must decide ***how*** **to generate tokens over time**.



Decoding Methods

Non-autoregressive and Recurrent

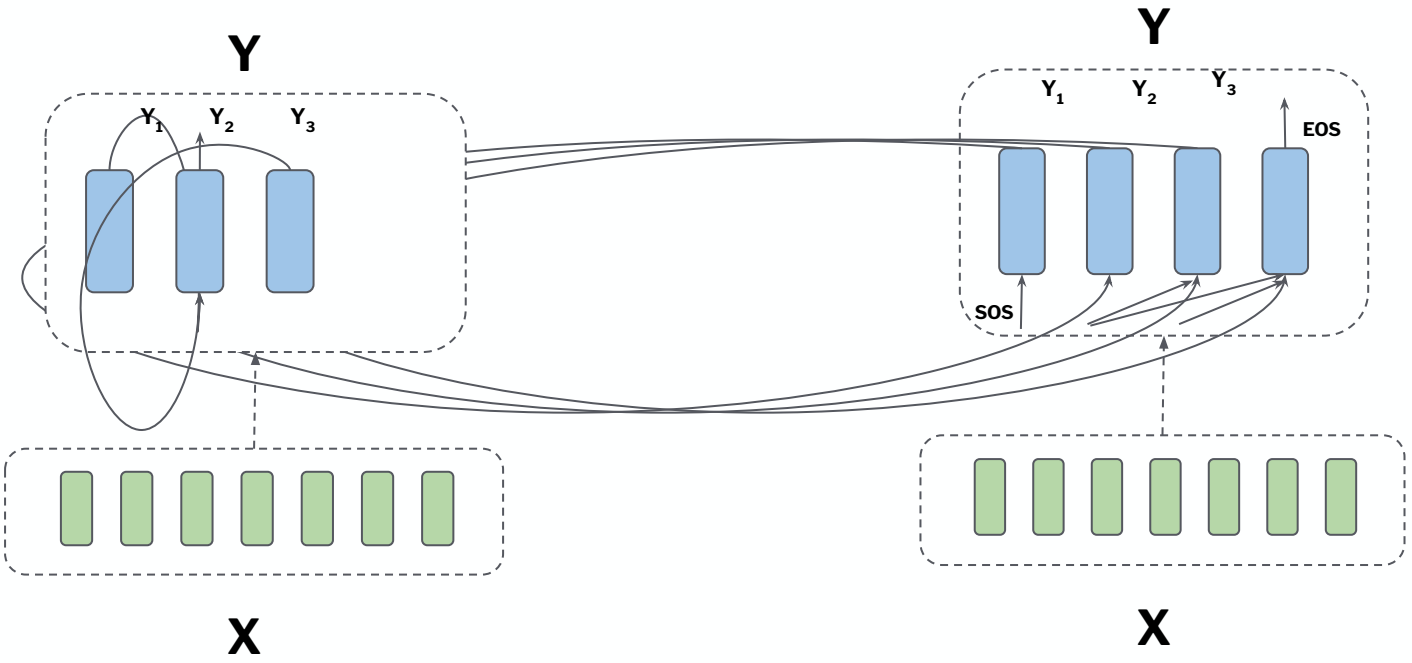




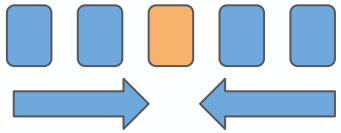
PURDUE UNIVERSITY

Decoding Methods

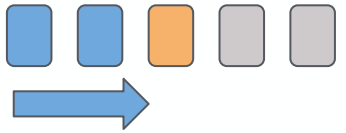
Masked and Autoregressive



$$p(Y|X) = \prod_{i=1}^{L} p(y_i|y_1, \ldots, y_{i-1}, y_{i+1}, y_L, X) = \prod_{i=1}^{L} p(y_i|y_{<i}, y_{>i}, X) = \prod_{i=1}^{L} p(y_i|y_{\bar{i}}, X)$$

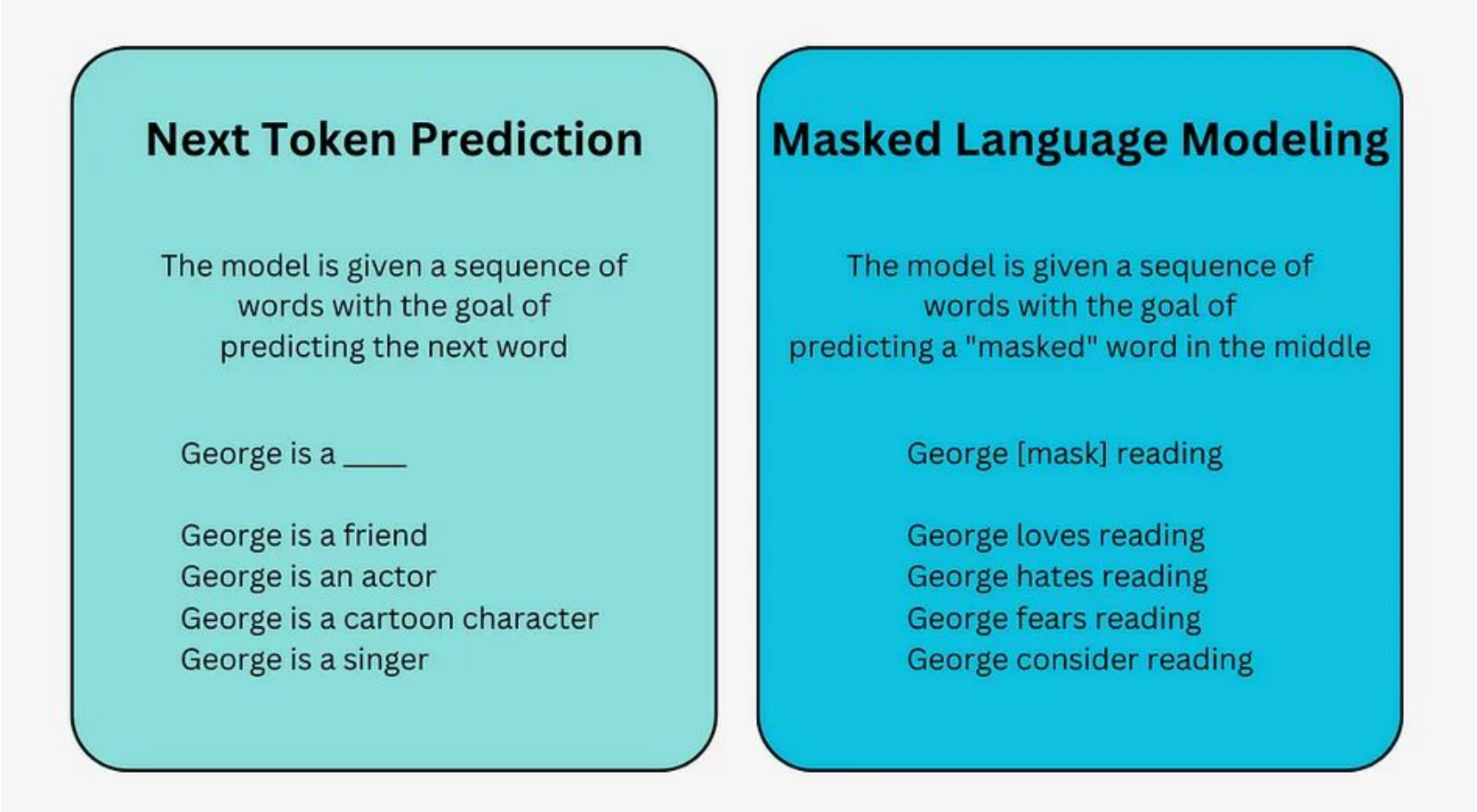


$$p(Y|X) = \prod_{i=1}^{L} p(y_i|y_1, \ldots, y_{i-1}, X) = \prod_{i=1}^{L} p(y_i|y_{<i}, X)$$



PURDUE UNIVERSITY

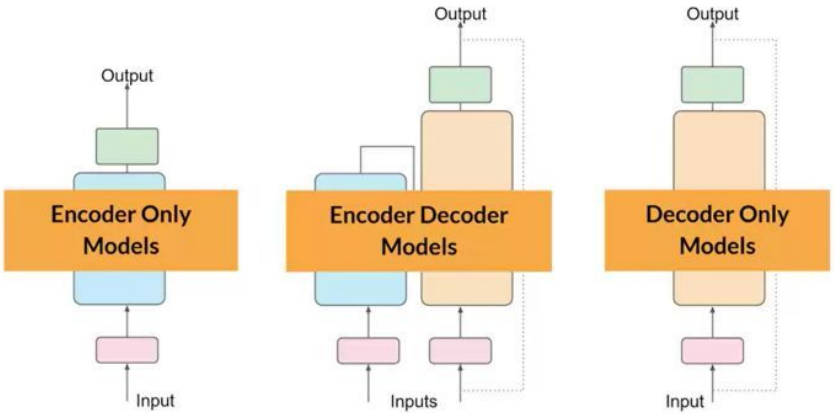
Decoding Methods

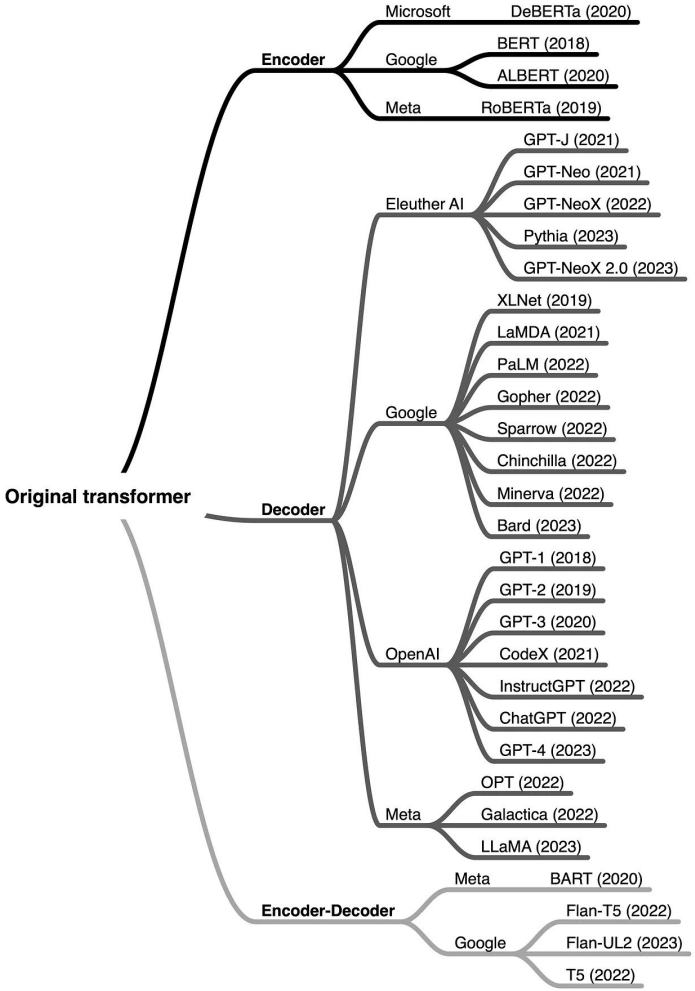


https://medium.com/@eugene-s/unleashing-the-potential-of-large-language-models-llms-with-chatgpt-8210f0cb063d

Transformer-based Language Models

Language models by company and year





PURDUE UNIVERSITY

Photo from:
https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder

Transformer-based Language Models

Language models by size and availability

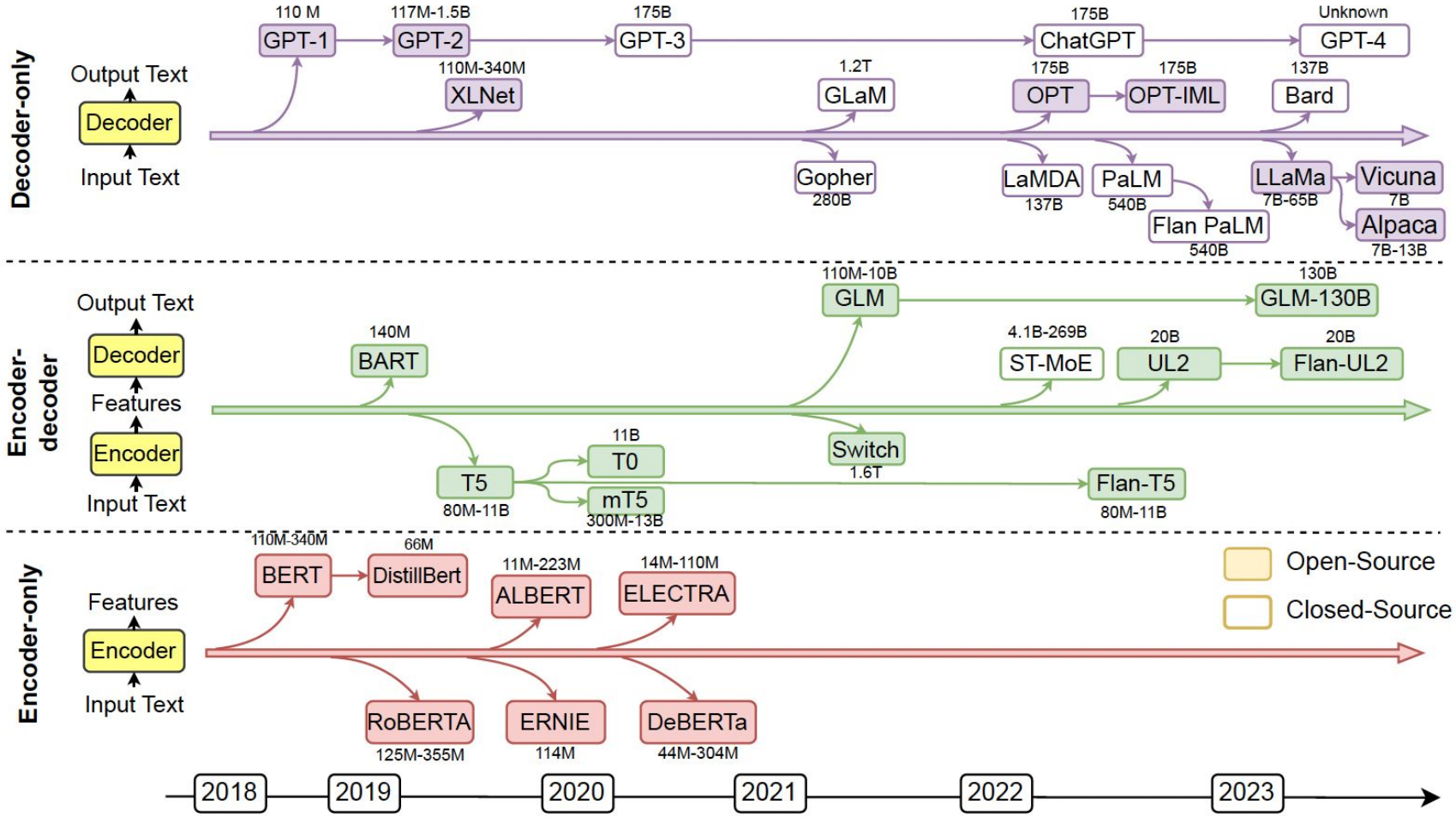


Photo from Unifying Large Language Models and Knowledge Graphs: A Roadmap,

Encode-only Models

Bert Breakthrough

Architecture:

- Encoder-only large language models only use the encoder to encode the sentence

Training:

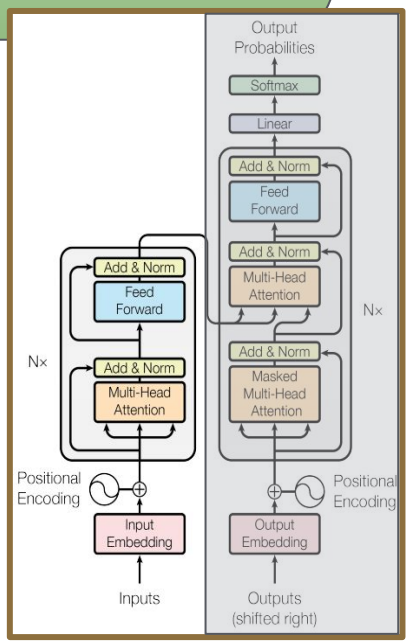
- The common **training paradigm** for these model is to **predict the mask words** in an input sentence.

$$\mathcal{L} = -\sum_{t \in M} \log P(y_t | x_{\text{masked}})$$

- This method is **unsupervised** (same sequence in input and output) and can be trained on the large-scale corpus.

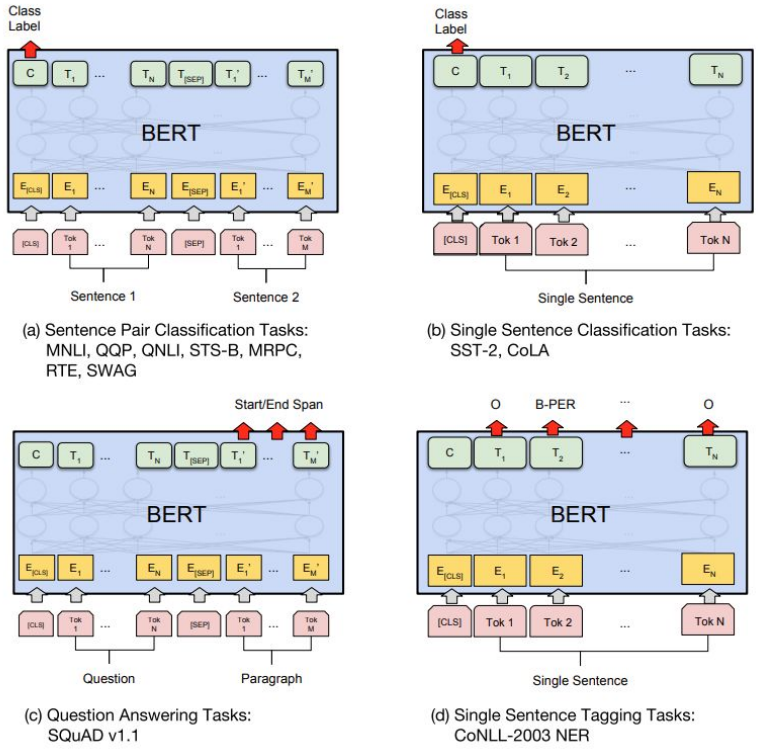
Prediction:

- These models are most effective for tasks that require **understanding the entire sentence**, such as text classification.
- Encoder-only models are still very useful for training **predictive models versus generating texts**.



Encoder-only Models at Prediction

- Classify token ([CLS]) represents **the entire input** sequence or sentence.
- The model uses this representation to **make predictions or classify the input into predefined categories.**
- **Fine-Tune BERT if you want to use CLS token**: Freeze part of model if your data/hardware is small.



Fine-tuning BERT on different tasks (Bert, 2019)

Encoder-Decoder Models

T5 breakthrough

Architecture:

- Encoder-decoder large language models adopt both the encoder and decoder module.

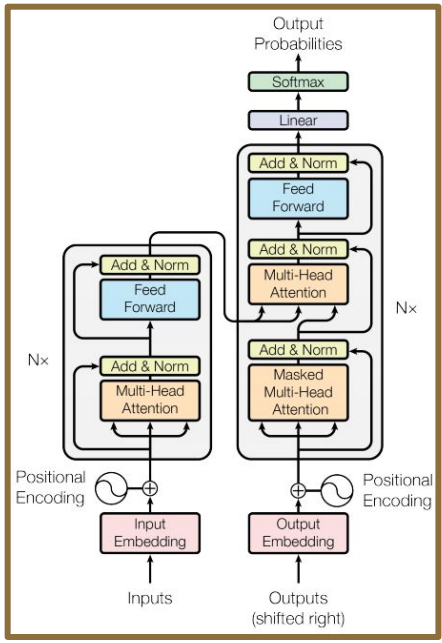
Training:

- The training strategies in encoder-decoder LLMs can be more flexible. For example, Masked Span Prediction, and autoregressive prediction.

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t | y_{<t}, x)$$

Prediction:

- They are typically used for tasks that involve **understanding input sequences** and **generating output sequences**, often with different lengths and structures.
- They are particularly good at tasks where where it is crucial to capture the relationships between the elements in both sequences.
- Some common use cases include text translation and summarization.



PURDUE UNIVERSITY

T5, Breakthrough of transfer Learning

- **Transfer Learning**: It enables a model to transfer knowledge across a wide range of tasks with minimal task-specific tuning.
- **Unified Approach**: T5 frames all NLP tasks (e.g., translation, summarization, classification) as a text-to-text problem, simplifying model training and application.

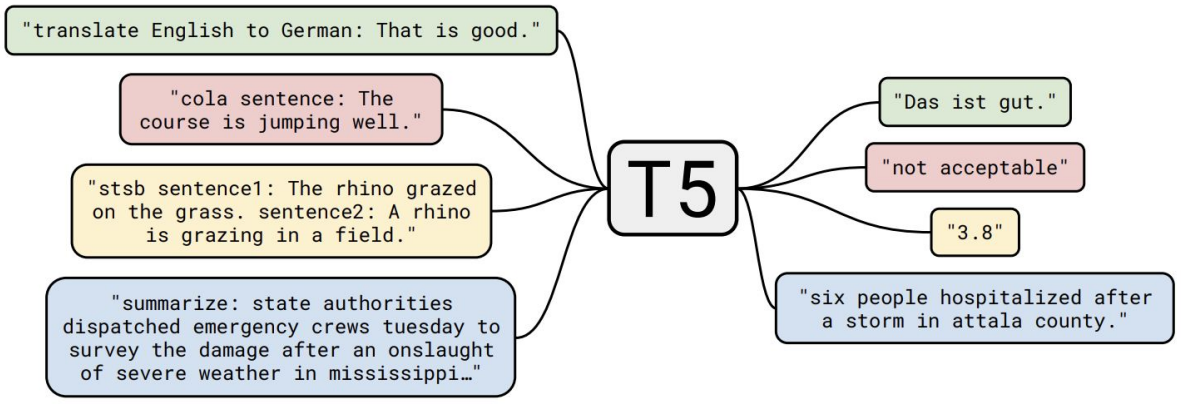


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".

Decoder-only Models

Zero and few-shot breakthrough

Architecture:

- Decoder-only large language models only adopt the decoder module to generate target output text.

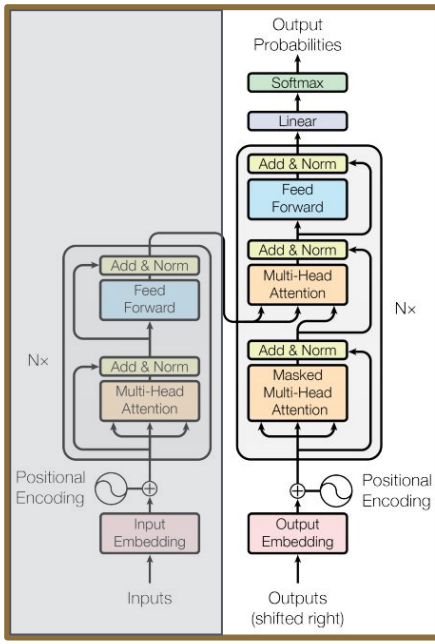
Training:

- The common **training paradigm** for these models is to **predict the next token (Autoregressive)**.

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t | y_{<t})$$

Prediction:

- Large-scale decoder-only LLMs can generally perform downstream tasks **from a few examples or simple instructions, without adding prediction heads or fine-tuning.**
- Many state-of-the-art LLMs (e.g., Chat-GPT and Llama) follow the decoder-only architecture.

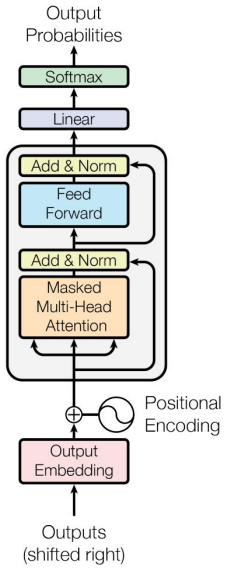


PURDUE UNIVERSITY

Training Objective

Decoder-only GPT

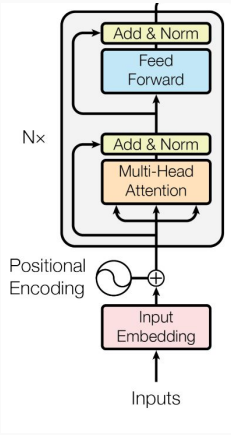
[sat_]



[START] [The_] [cat_]

Encoder-only BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



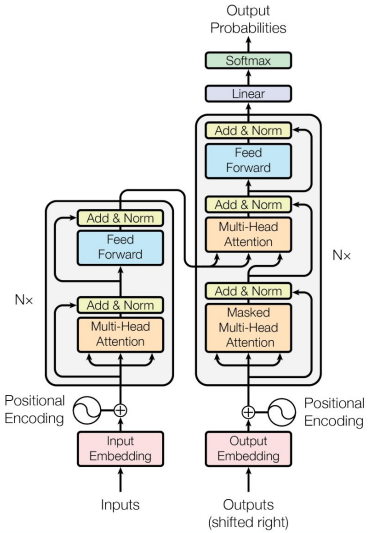
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Enc-Dec T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

PURDUE UNIVERSITY

Photo from Luca Bayer lecture on transformers

Course Outline-3rd session-Transformers and LLMs:

1. From RNNs to Transformers

- Brief recap of:
  Sequence modeling with RNNs (limitations: vanishing gradients, sequential computation bottlenecks).
  - Traditional attention mechanisms (Bahdanau, Luong).
- Motivation for Transformers

2. Transformer Architecture Overview and Core components:

- Tokenization
- Self-attention
- Positional encoding: Sinusoidal vs learned positional encodings
- Layer normalization & residual connections

3. LLM Variants and Evolution

- Encoder-Only models,Architecture, Training and Prediction.
- Encoder-Decoder models,Architecture, Training and Prediction.
- Decoder-only models, Architecture, Training and Prediction.

4. LLM Tips and Tricks

- Context Window Size
- Inference and Next Token Prediction
- Selection Criteria for LLMs

Context Window Size

- **Context window** : **maximum number of tokens** the model can consider at one pass.
- A larger context window allows you to feed a wide variety of extensive materials to the model.

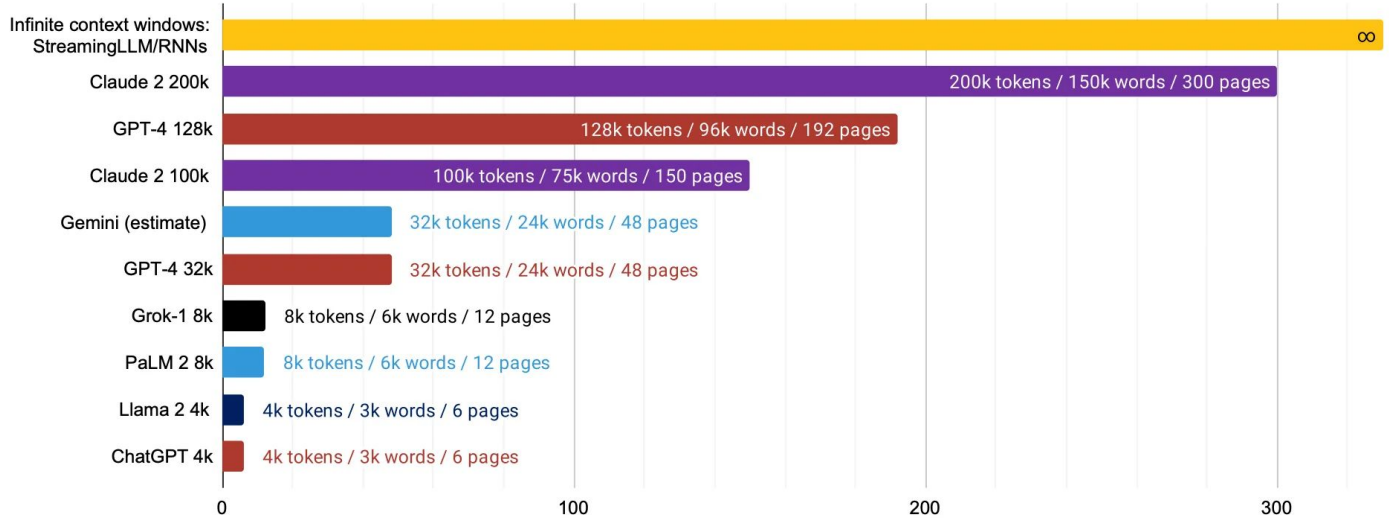


Photo from:https://s10251.pcdn.co/pdf/2023-Alan-D-Thompson-2023-Context-Windows-Rev-0.pdf

PURDUE UNIVERSITY

Context window size

Pros of large context window	Cons of large context window
Improved **Long-Term Dependencies**	**Computation and cost** exhaustive, both training and inference time.
Good for **conversational AI**: ability to 'remember' more and keeping up with extended conversations	The model's ability deteriorates as more documents are introduced. (LLMs perform well when presented with a **more focused** set of documents directly relevant to the context).

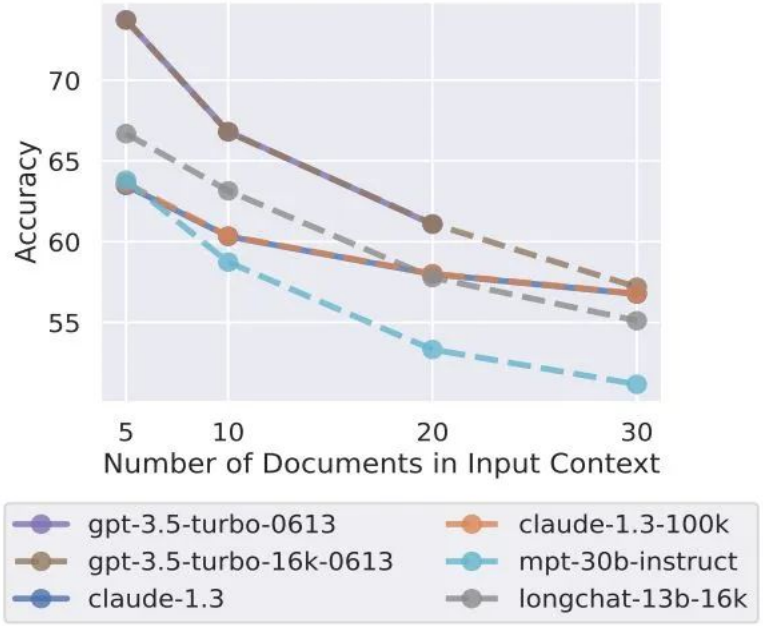


Photo from "lost in the middle" paper.

Decoding or Next Token Prediction

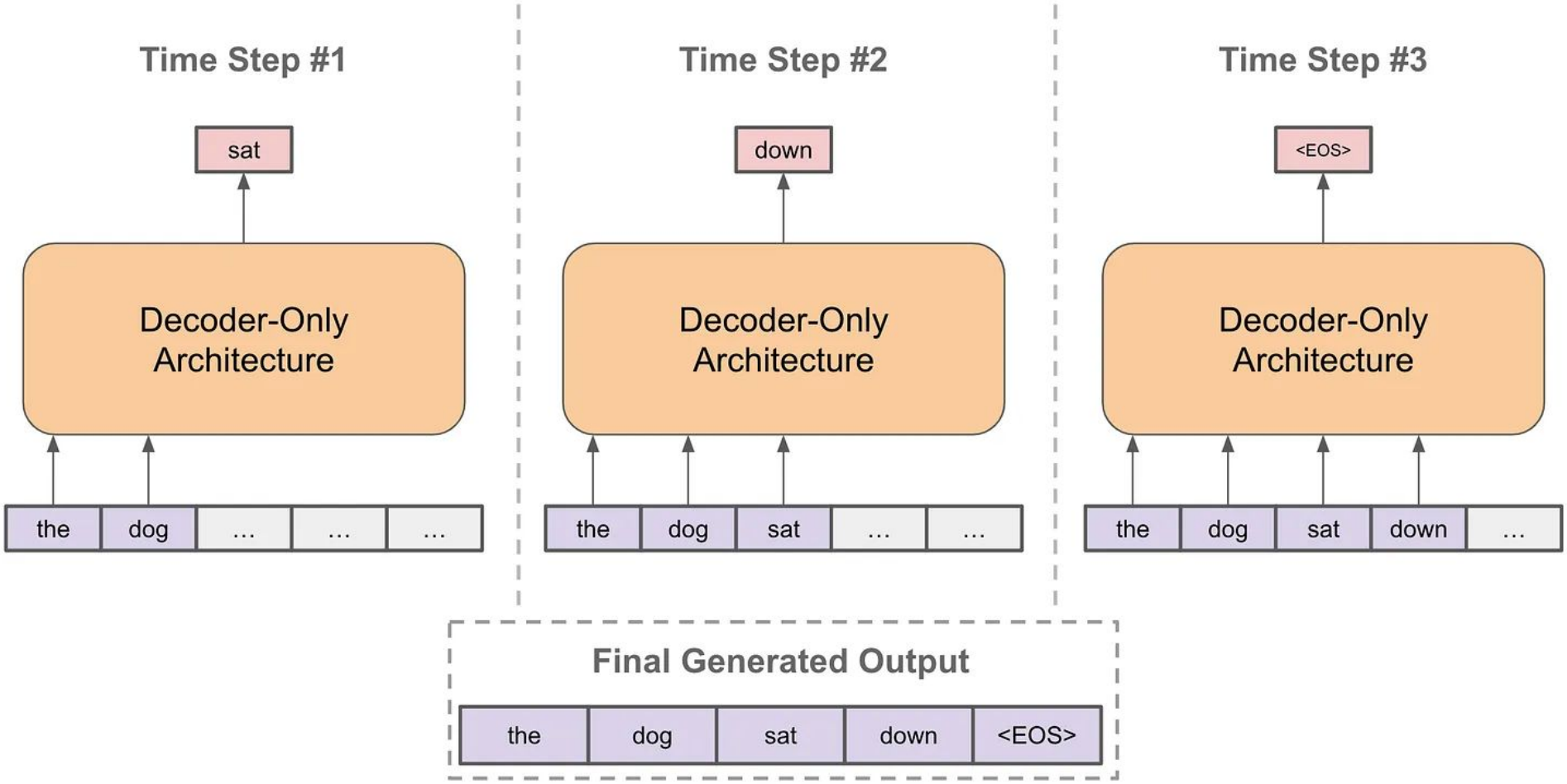


Photo from https://x.com/cwolferesearch/status/1689388468911132672

Decoding or Next Token Prediction

- When the model receives textual input, the text is “tokenized”, or separated into individual words/sub-words.
- Then, we retrieve an embedding (i.e., a vector assigned to each unique token) for every token, forming a sequence of token vectors.

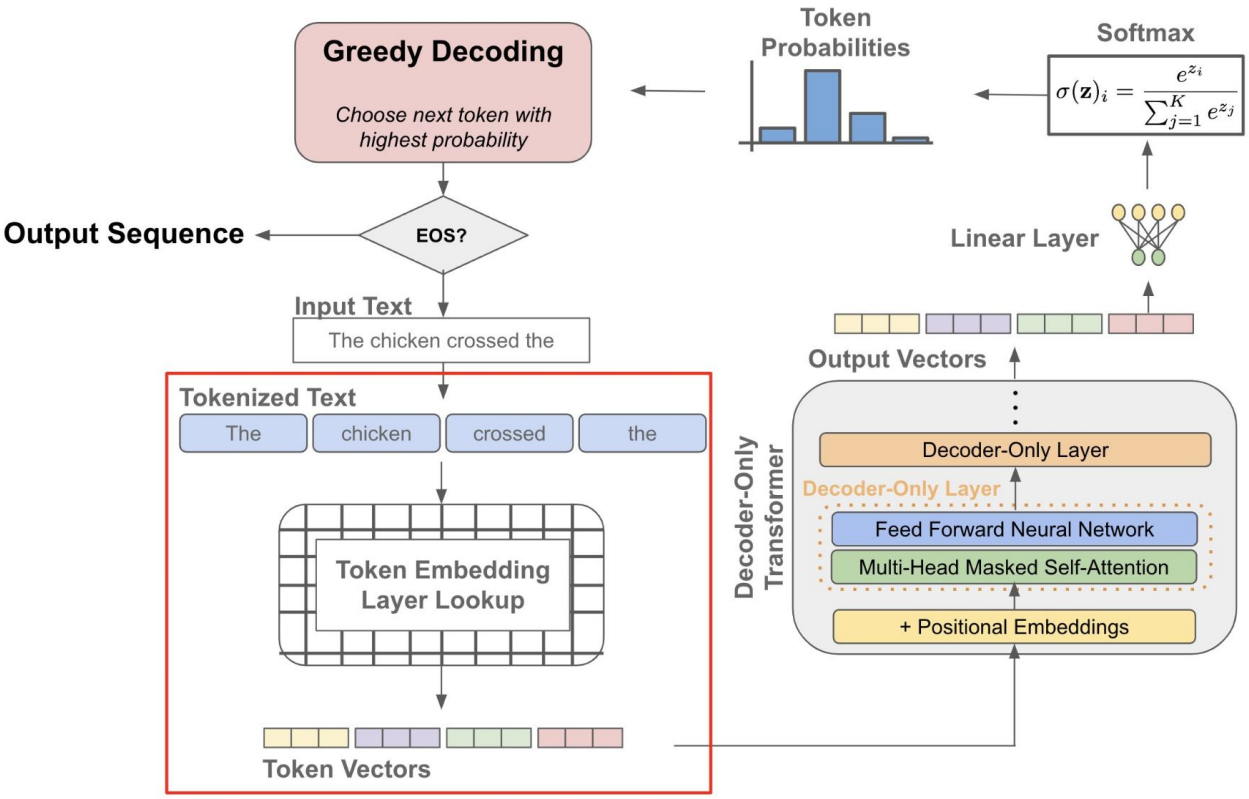


Photo from https://twitter.com/cwolferesearch/status/1659608476455256078?s=20

Decoding or Next Token Prediction

This sequence of token vectors is passed through a decoder-only transformer which outputs a new list of equally-sized token vectors that have been transformed via many masked self-attention and feed-forward layers in sequence.

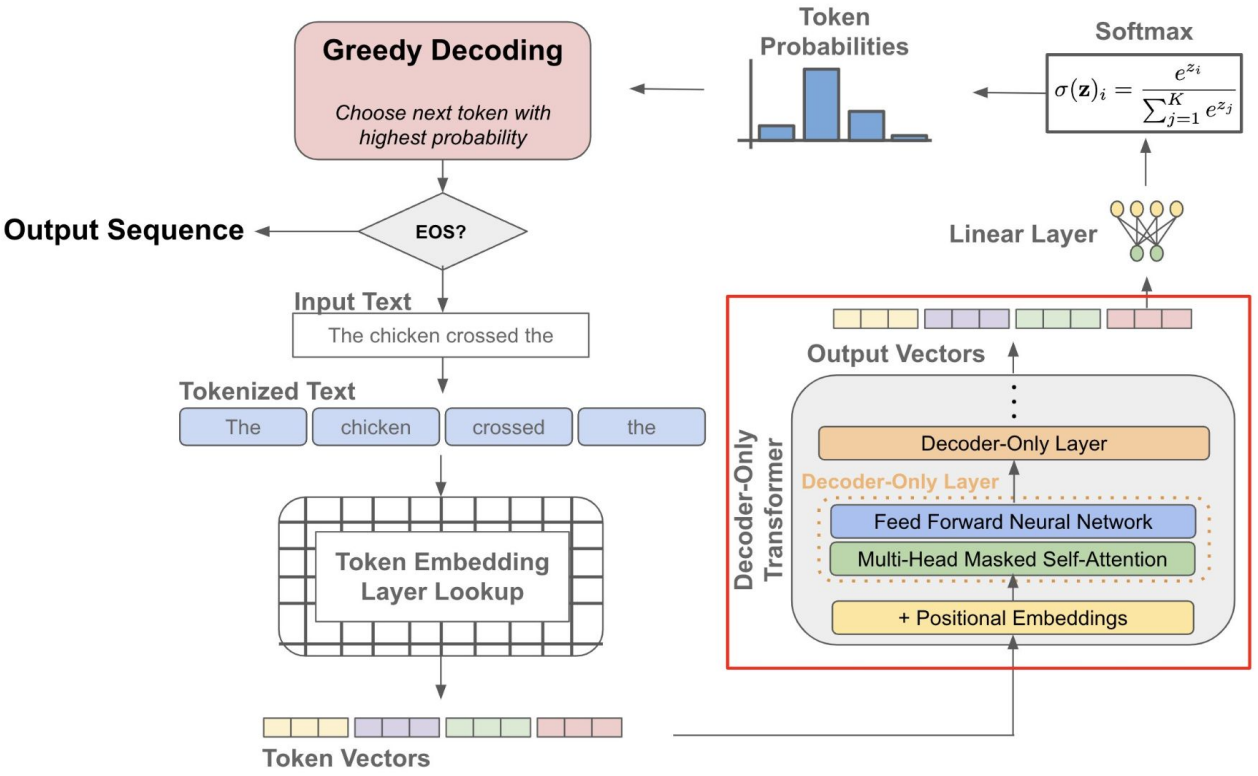


Photo from https://twitter.com/cwolferesearch/status/1659608476455256078?s=20

Decoding or Next Token Prediction

The LLM takes the last vector in this output and passes it through a feed-forward layer, producing **a new vector with the same size as our token vocabulary**.

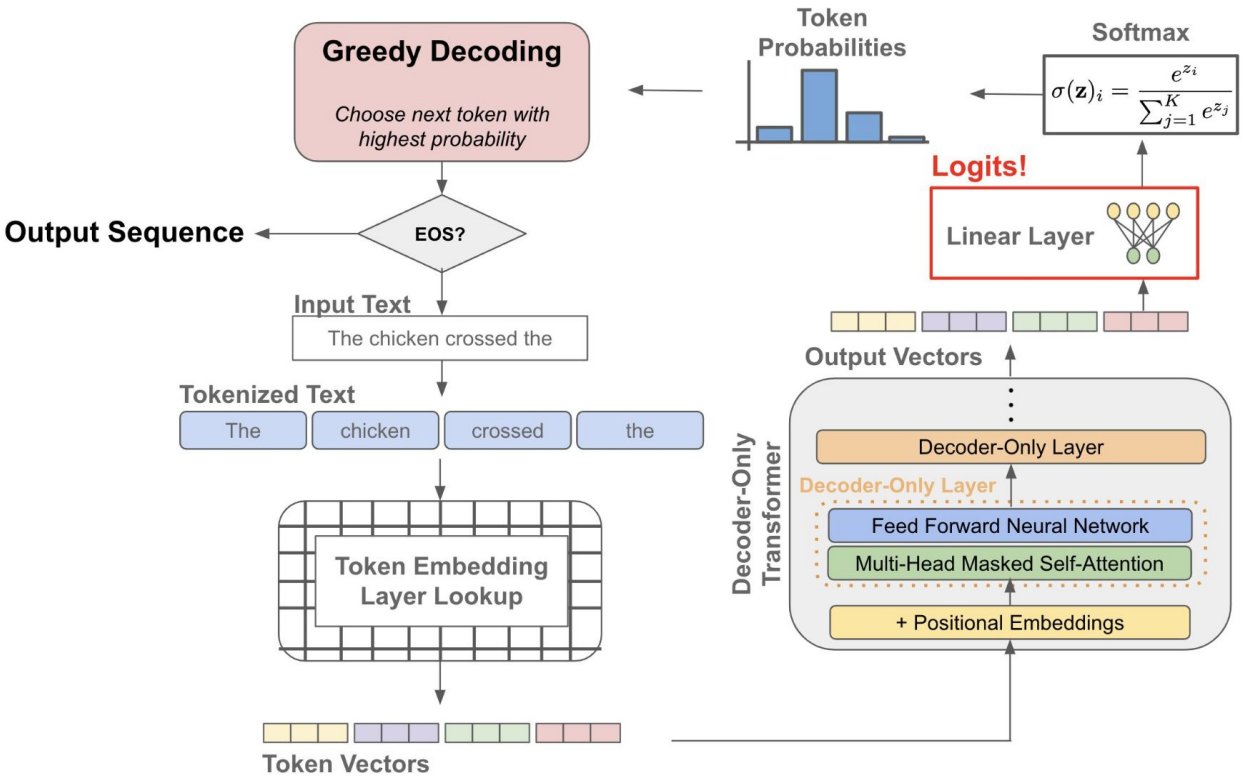


Photo from https://twitter.com/cwolferesearch/status/1659608476455256078?s=20

Decoding or Next Token Prediction

Next, we apply a softmax function to our logits, to form a probability distribution over the space of possible tokens.

In other words, this tells us the probability that each token is the correct “next” token!

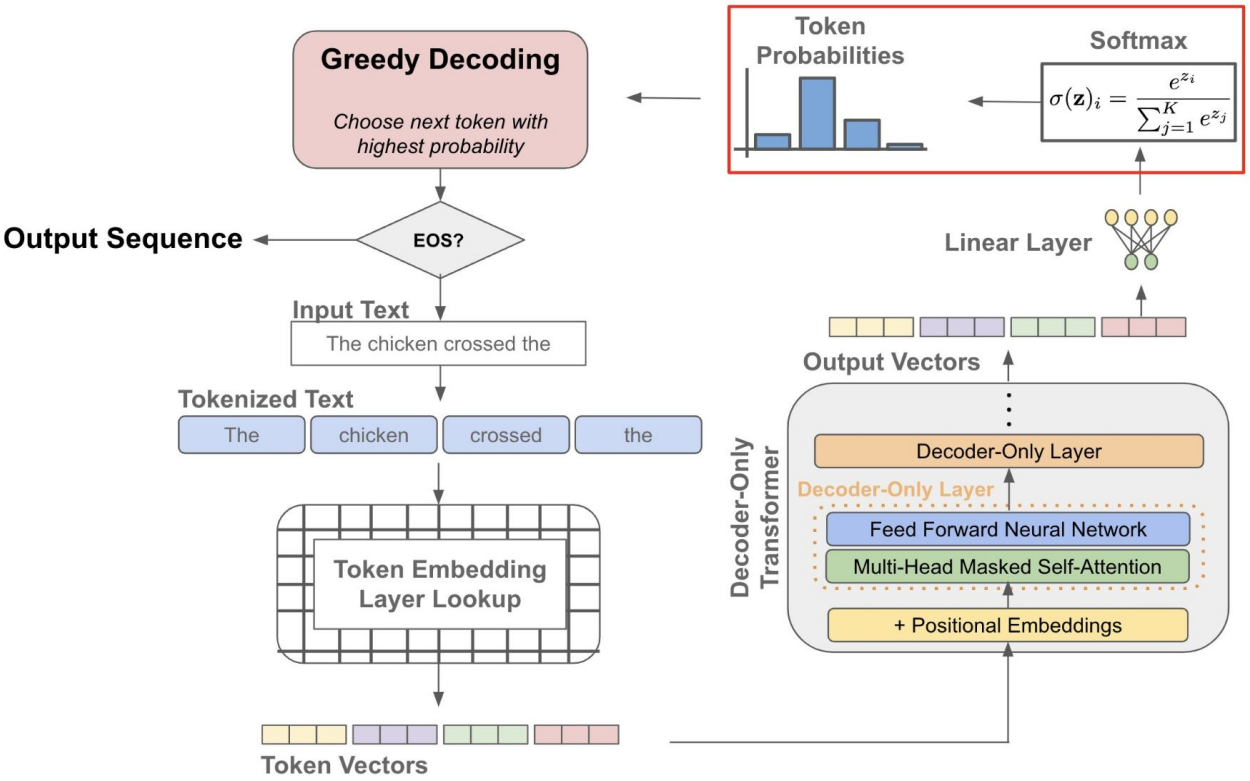


Photo from https://twitter.com/cwolferesearch/status/1659608476455256078?s=20

Greedy Decoding

Greedy decoding just chooses the next token as the **highest-probability token** in this distribution.

- We can add this token to our input sequence and use it to generate another token auto-regressively.
- To generate a full sequence, we just keep doing this!

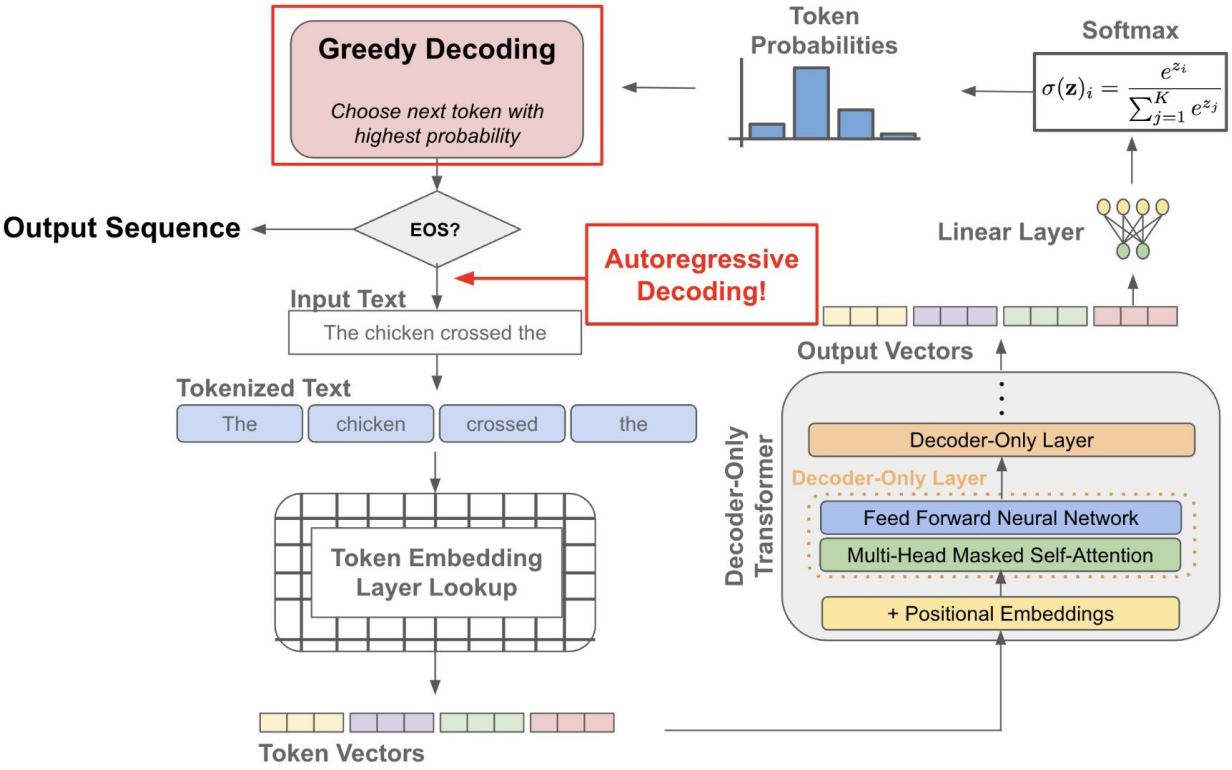


Photo from https://twitter.com/cwolferesearch/status/1659608476455256078?s=20

Strategies for Next Token Prediction

Given this probability distribution, there are several decoding strategies that we can follow, which are just different ways of selecting the next token from this distribution.

- Greedy Decoding
- Temperature
- Top-K Sampling
- Top-P (Nucleus) Sampling

Greedy Decoding

Greedy decoding has some drawbacks; outputs can get stuck in repetitive loops, for example.

- Think of the suggestions in your smartphone's auto-suggest. When you continually pick the highest suggested word, it may devolve into repeated sentences.

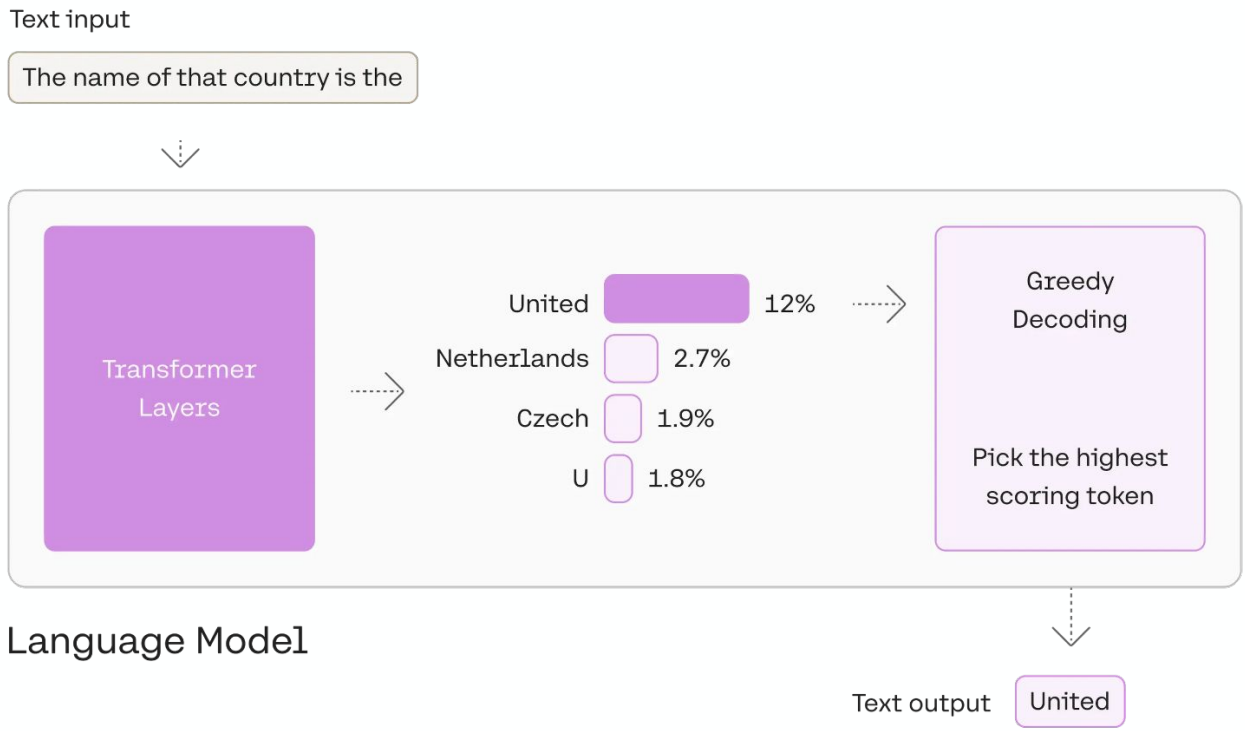


Photo taken from
https://docs.cohere.com/docs/controlling-generation-with-top-k-top-p

Temperature

- Temperature scales the logits before sampling tokens to help with generating more **diverse outputs** with a language model.
- It increases the chance of the model generating something random or irrelevant.
- Higher values of temperature make output **more random**, while lower values of temperature make output more deterministic.
- Use higher values of temperature with caution!

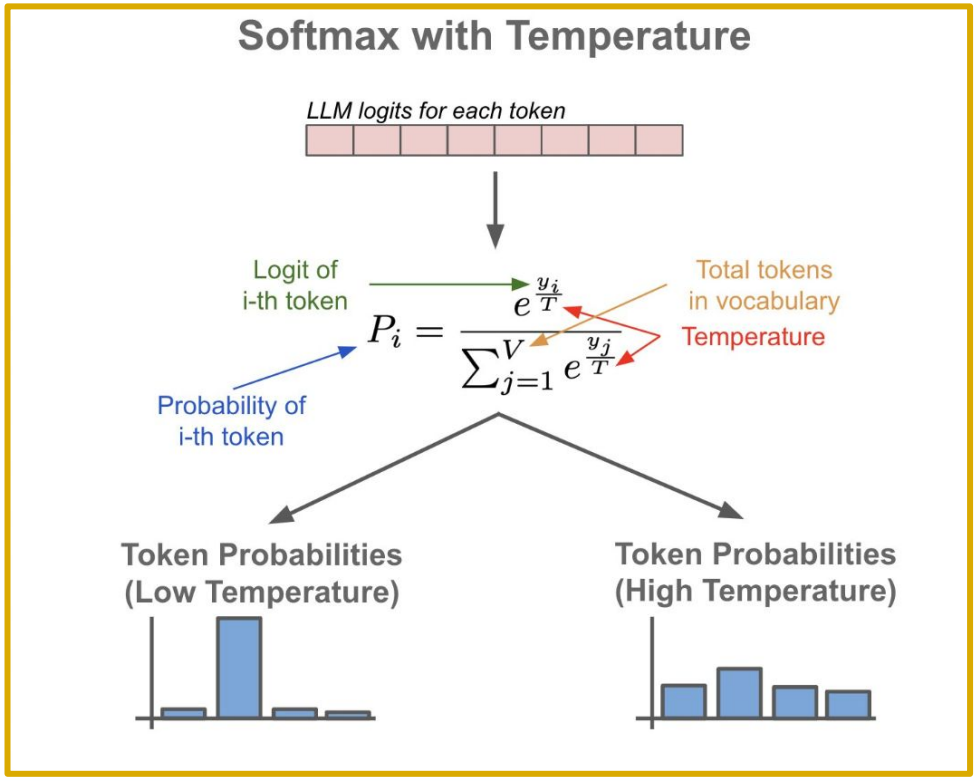


Photo from https://twitter.com/cwolferesearch/status/1671628210180698112?s=20

Top-K sampling

- Another commonly-used strategy is to sample from a **shortlist of the top k tokens**.
- This approach allows the other high-scoring tokens a chance of being picked.
- The randomness introduced by this sampling helps the quality of generation in a lot of scenarios.

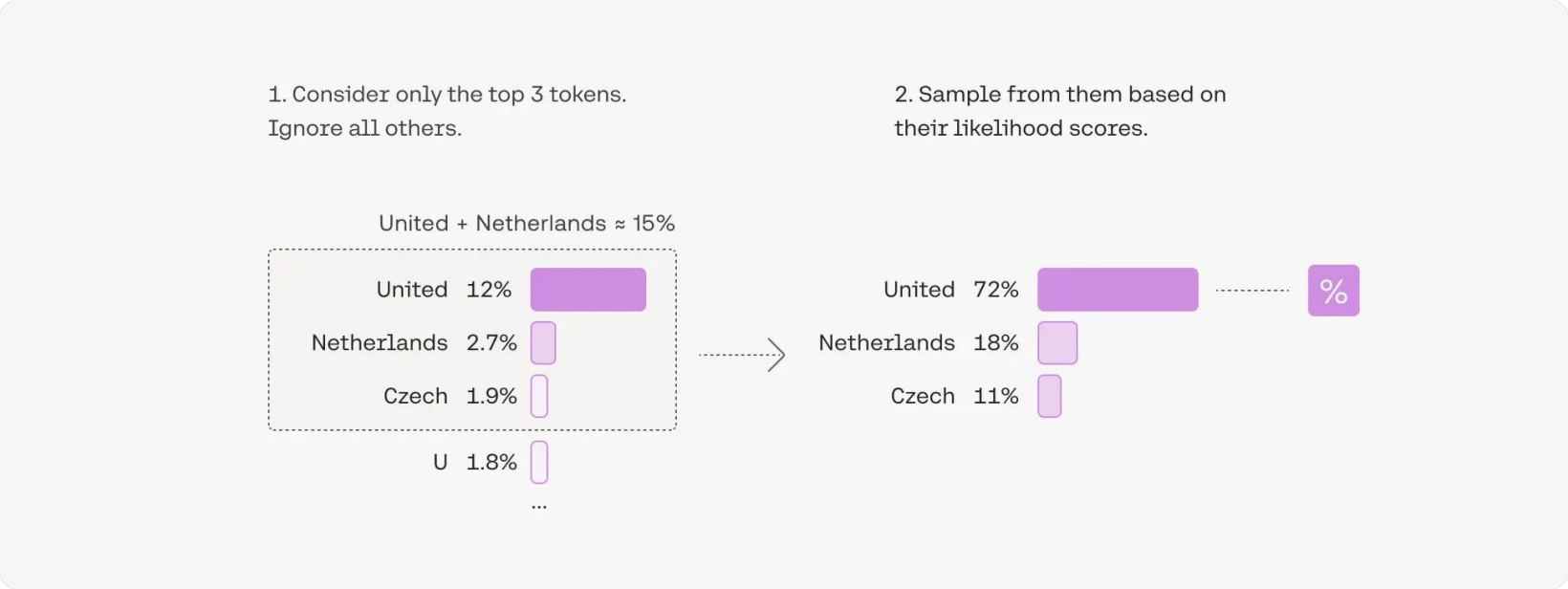


Photo taken from
https://docs.cohere.com/docs/controlling-generation-with-top-k-top-p

Top-P or Nucleus sampling

Nucleus Sampling creates the shortlist by selecting the top tokens whose **sum of likelihoods does not exceed a certain value**.

Top-p is usually set to a high value (like 0.75) with the purpose of **limiting the long tail of low-probability tokens** that may be sampled.

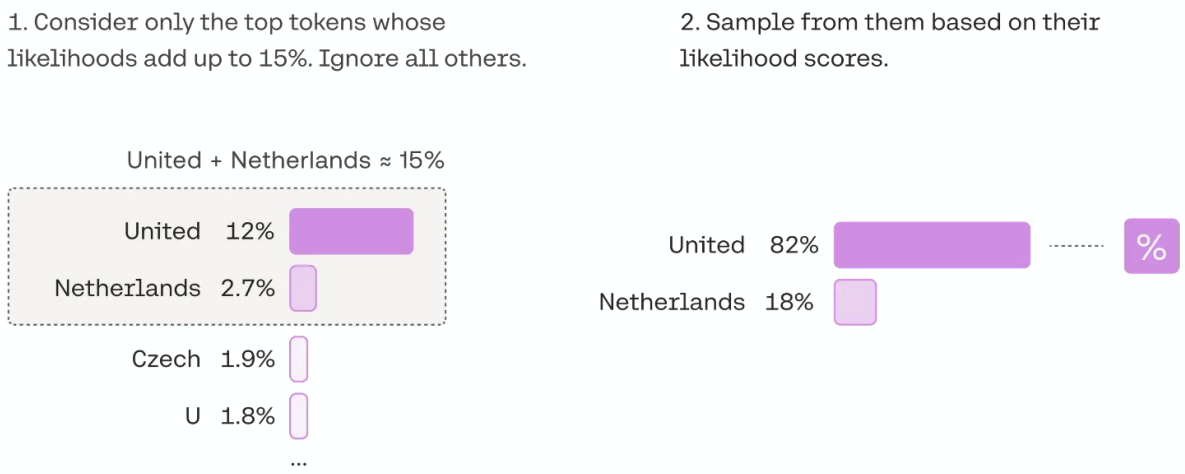


Photo taken from
https://docs.cohere.com/docs/controlling-generation-with-top-k-top-p

Selection Criteria For LLMs

Task alignment: Choose an LLM that aligns to the task:

- **Encoder-only (BERT)**: Best for **classification**, **semantic similarity**, **retrieval**.
- **Decoder-only (GPT, LLaMA)**: Best for **generative tasks**, like conversational AI, writing, answering.
- **Encoder-Decoder (T5, FLAN-T5, BART)**: Good for **translation**, **summarization**, **seq2seq tasks**.

Data Alignment: Choose an LLM that has been **pre-trained or fine-tuned** on data that matches the domain or context of your project.

- **T**o build a chatbot that **answers patient queries using medical literature:**
  - **Use PubMedBERT** or **BioGPT**, trained on biomedical literature and clinical notes, Instead of using GPT-2, trained mostly on generic web data like Reddit or Wikipedia,

Adapting and Tuning: Determine if the chosen LLM can be effectively contextualized with prompts or fine-tuning.

Selection Criteria For LLMs

Explainability

- For high-stakes domains (e.g., health, engineering), use models that allow:
  - **Attribution** (e.g., RAG with source highlighting)
  - **Token attention visualization** (e.g., using attention heatmaps or explainers)

Model size and complexity

- Models with tens of billions of parameters provide higher-quality outputs but require more **computational resources**.

Dataset Size and Fine-Tuning

- If you have a **small domain-specific dataset**:
  - **Use smaller models** which require fewer resources and fine-tune faster.
- Avoid massive models unless you're using **zero-shot** or **few-shot prompting**, not fine-tuning.

Next:

1. Multi-Modal Transformers
    a. (Speech, Image, Video, 3D Vision)
2. RAG, Fine-Tuning, Prompt Engineering, etc.

PURDUE UNIVERSITY

Thank You

PURDUE UNIVERSITY