Transformers & Foundation Models

Instructor: Lara J. Martin (she/they)

TA: Omkar Kulkarni (he)

https://laramartin.net/NLP-class/

Slides modified from Dr. Daphne Ippolito

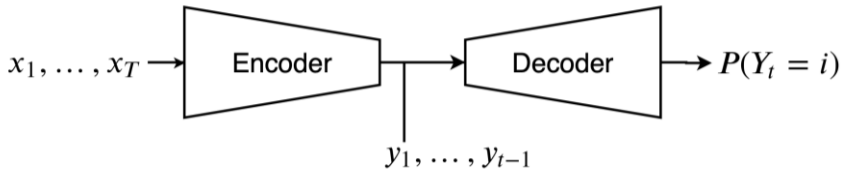
Learning Objectives

Compare sequence-to-sequence RNNs to transformers

Compare and contrast all LM types so far

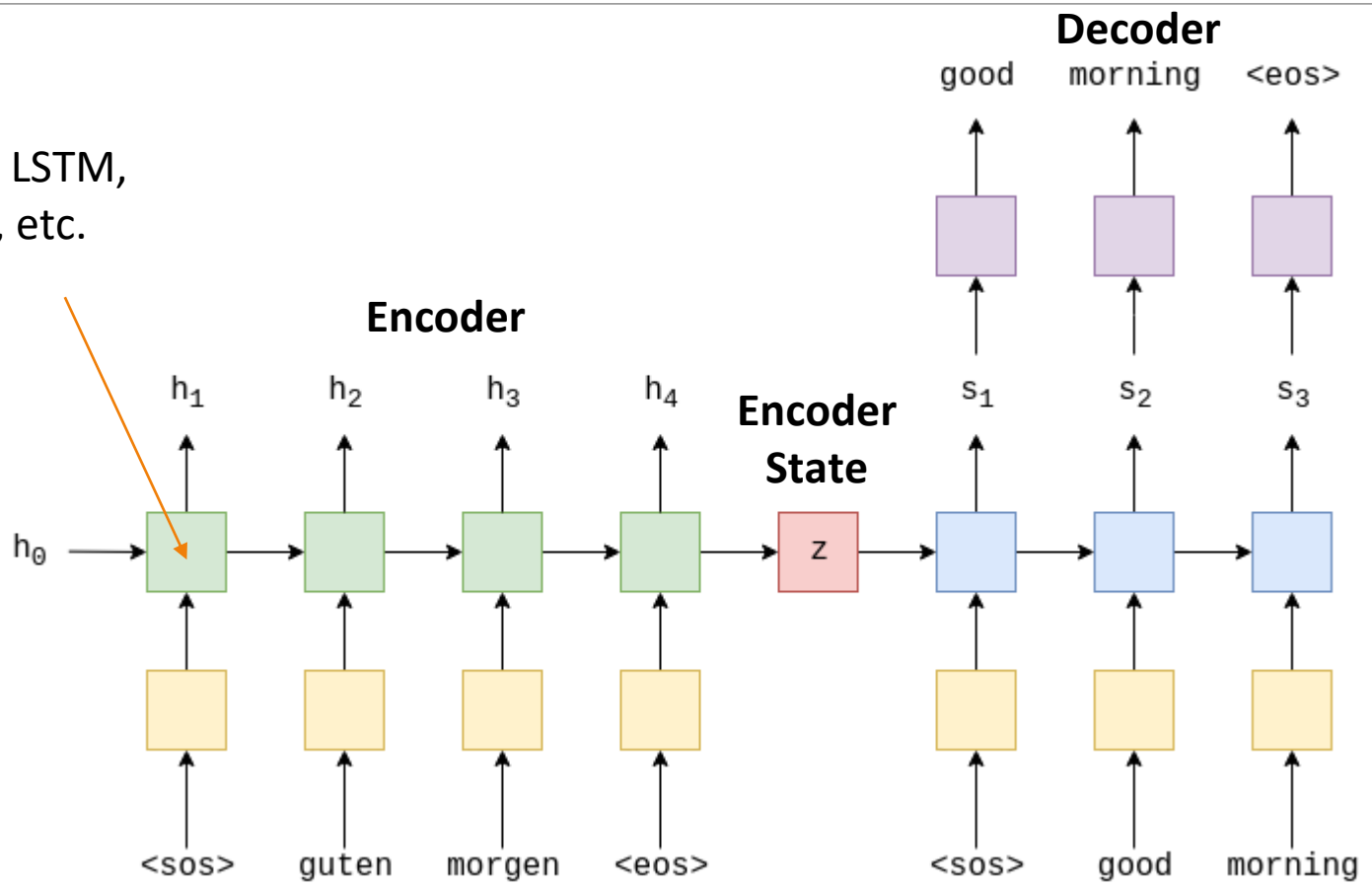
Differentiate between encoder model embeddings and older dense embeddings

Recognize useful encoder-only, encoder-decoder, and decoder-only models



Review: Sequence-to-Sequence

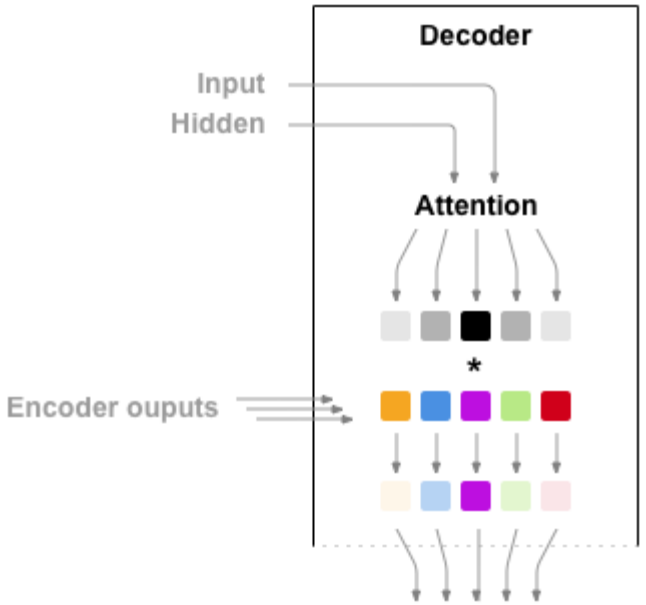
Can be LSTM, GRU, etc.



https://colab.research.google.com/github/bentrevett/pytorch-seq2seq/blob/main/1%20-%20Sequence%20to%20Sequence%20Learning%20with%20Neural%20Networks.ipynb#scrollTo=k6sRrL4wKsmi

I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, 2014, pp. 3104–3112. https://proceedings.neurips.cc/paper_files/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

Review: Attention



Compute a linear combination of the encoder hidden states.

$$\mathbf{c}_t = \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_T$$

Decoder's prediction at position t is based on both the context vector and the hidden state outputted by the RNN at that position.

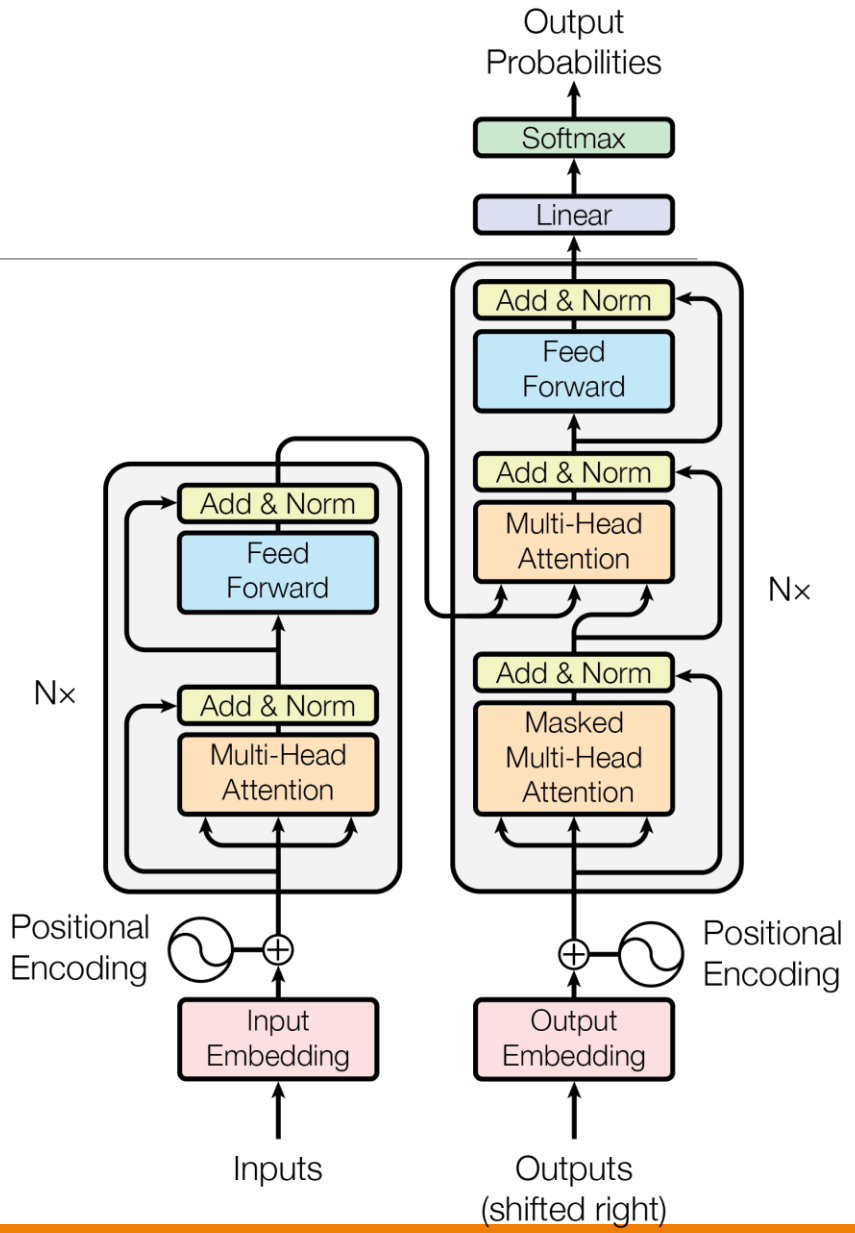
$$\hat{\mathbf{e}}_t = f_\theta(\mathbf{h}_t^{\text{dec}} \quad \mathbf{c}_t)$$

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

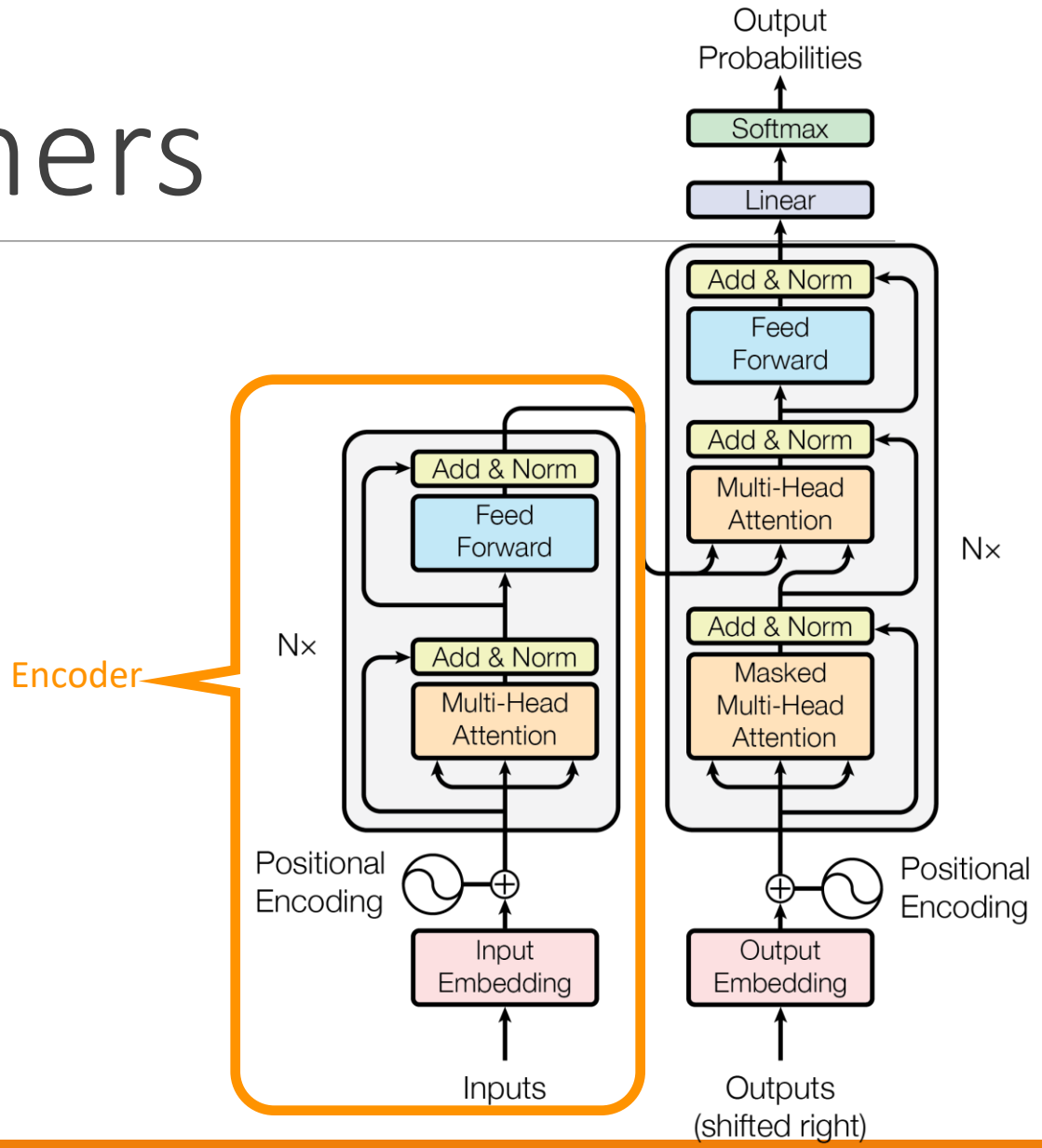
Review: Transformers

The Transformer is a **non-recurrent** non-convolutional (feed-forward) neural network designed for language understanding

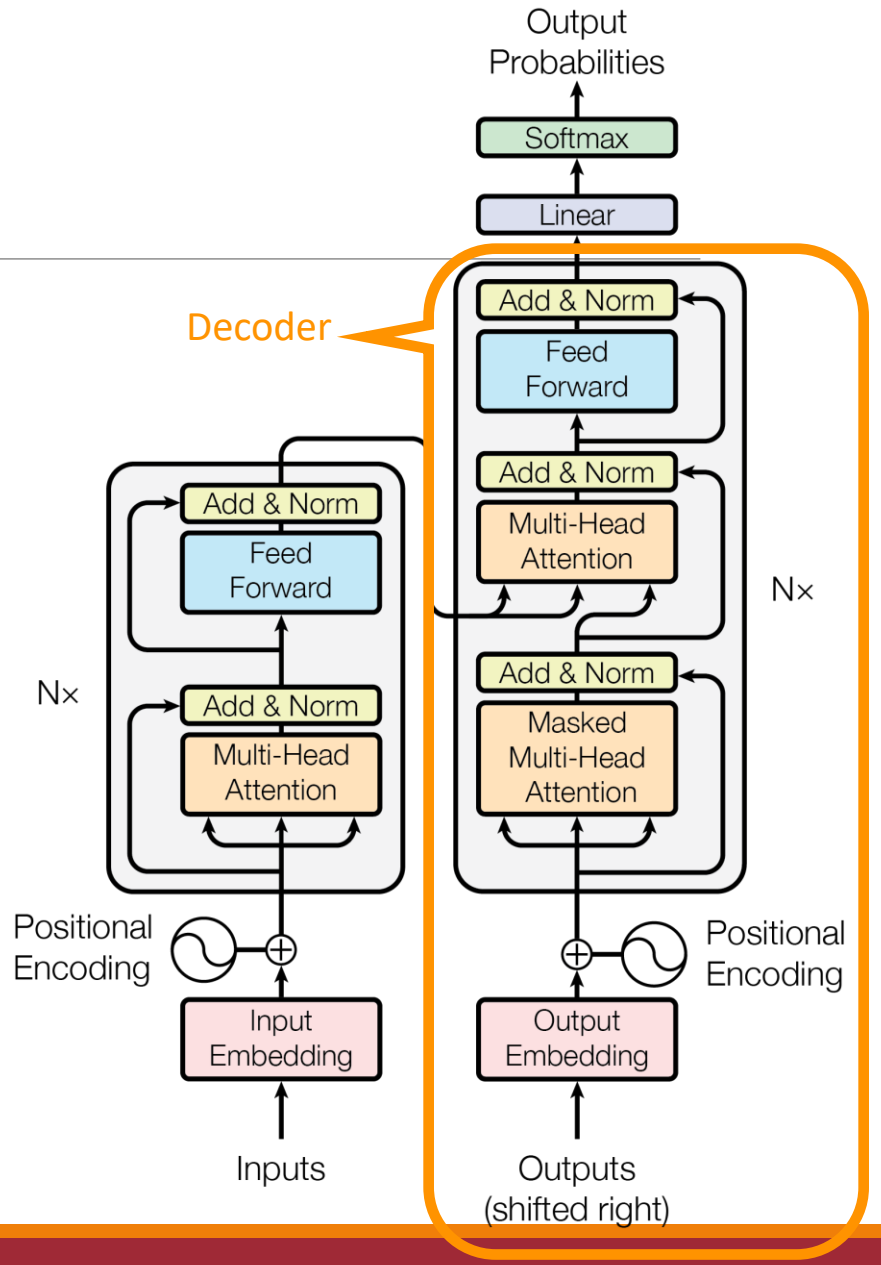
- introduces self-attention in addition to encoder-decoder attention



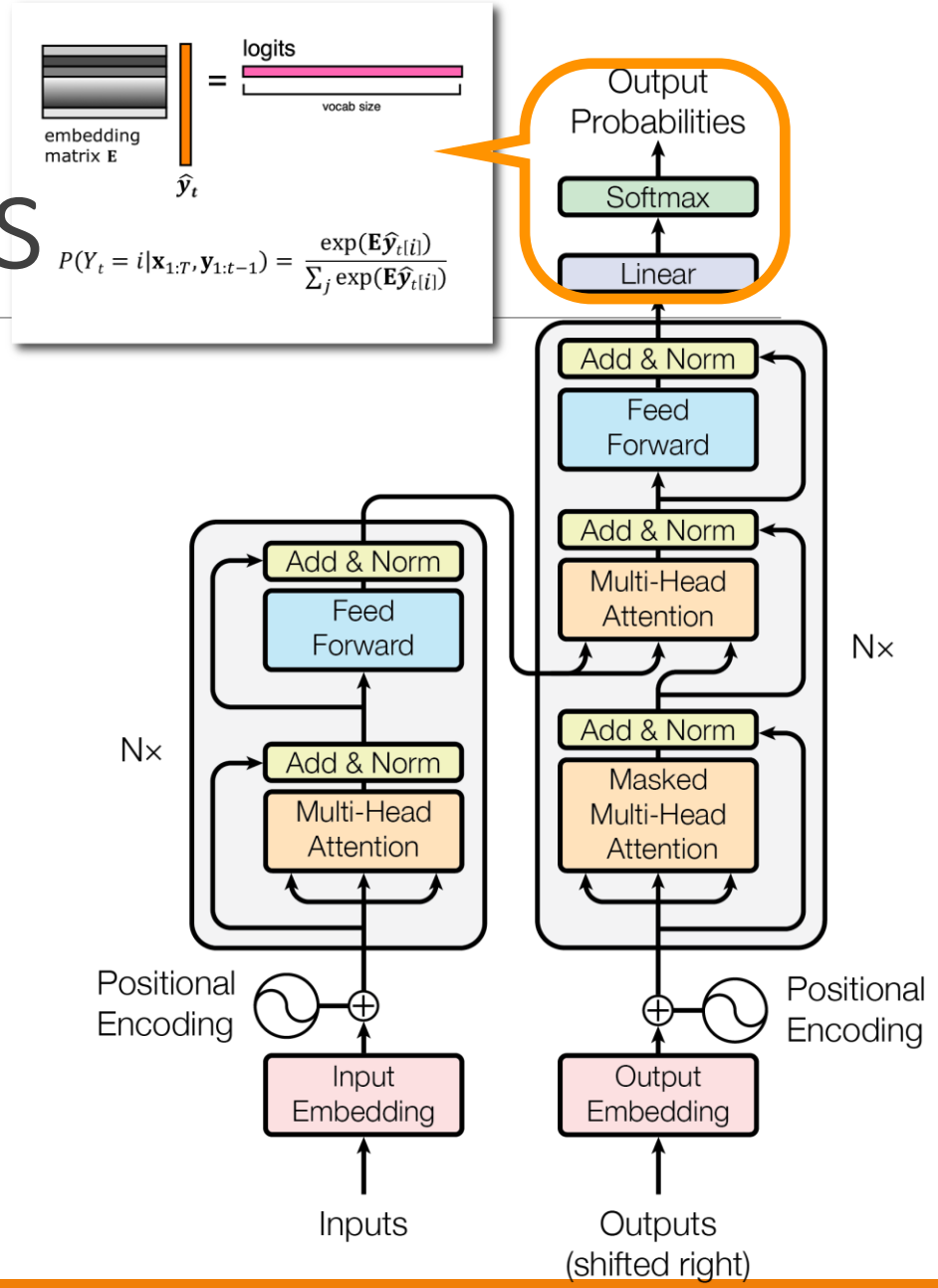
Review: Transformers



Review: Transformers

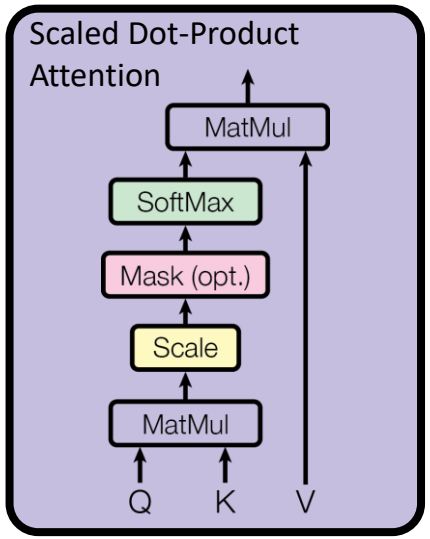


Review: Transformers



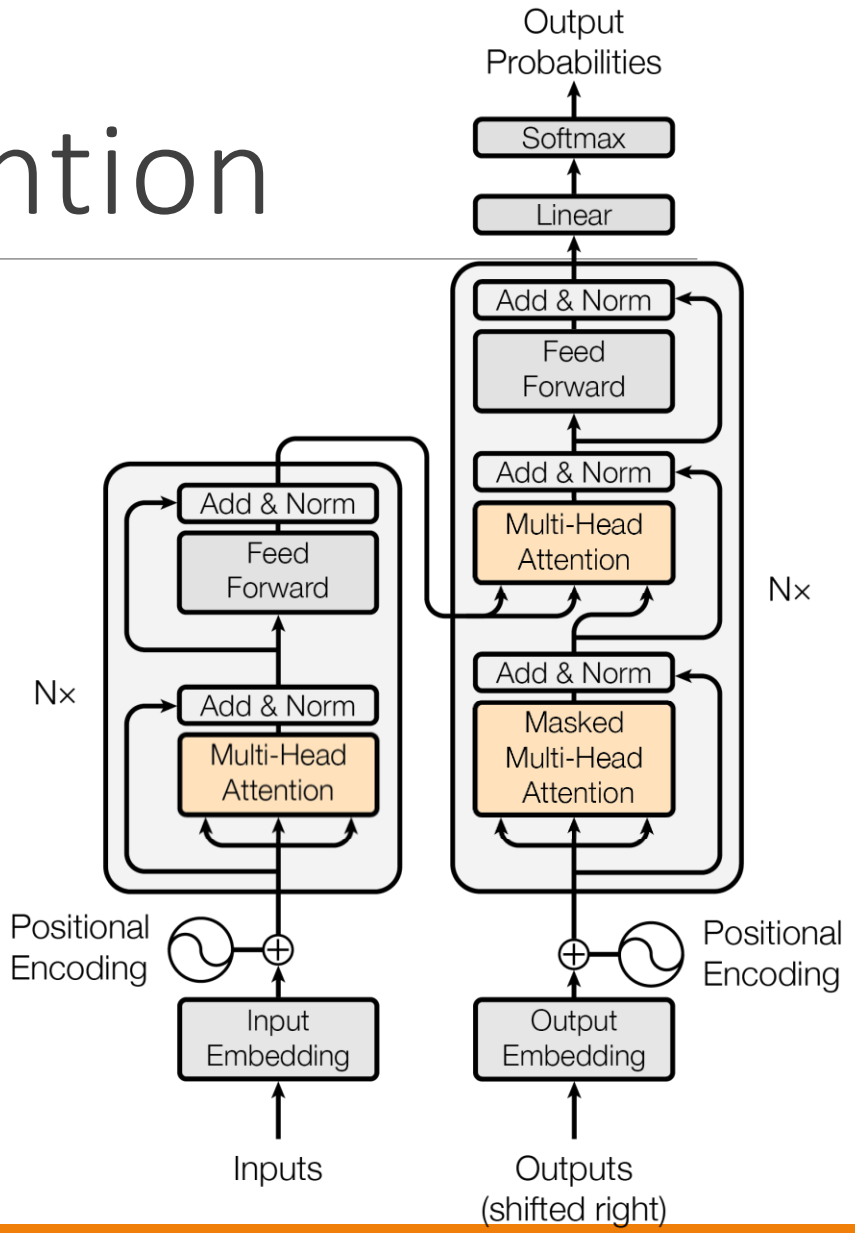
Review: Scaled Dot-Product Attention

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



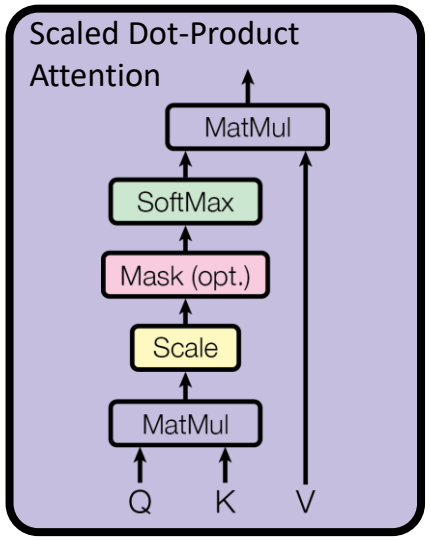
<u>The rough algorithm:</u>

- For each vector in Q (query matrix), take the linear sum of the vectors in V (value matrix)
- The amount to weigh each vector in V is dependent on how "similar" that vector is to the query vector
- "Similarity" is measured in terms of the dot product between the vectors



Scaled Dot-Product Attention

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

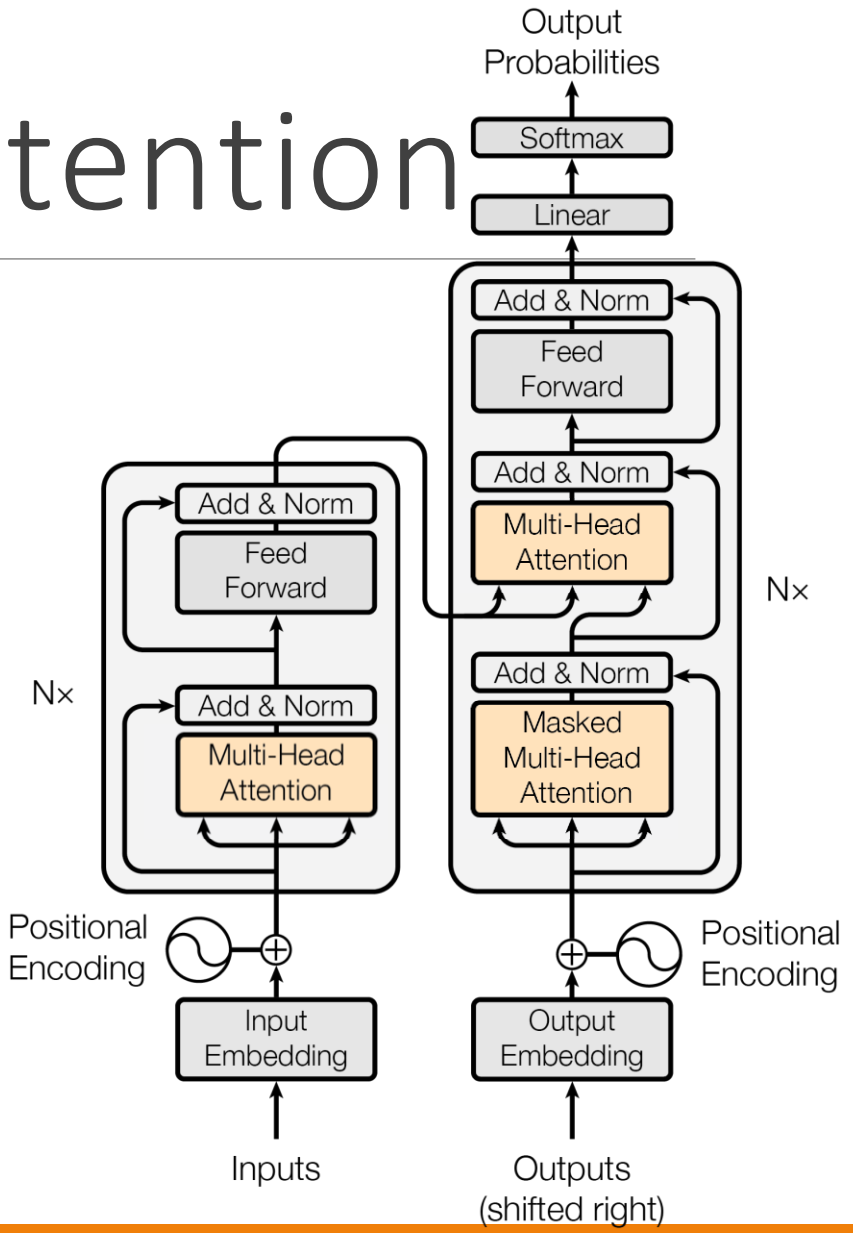


For self-attention:

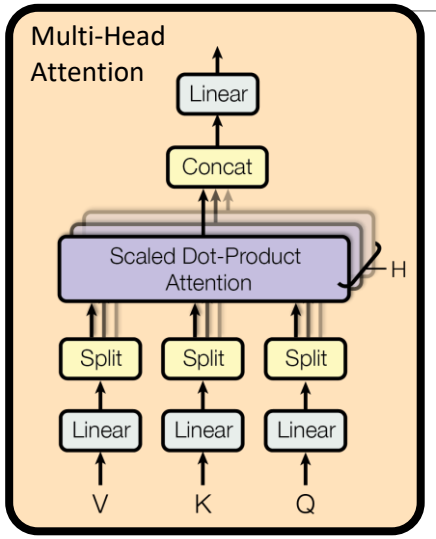
Keys, queries, and values all come from the outputs of the previous layer

For encoder-decoder attention:

Keys and values come from encoder's final output. Queries come from the previous decoder layer's outputs.



Multi-Head Attention



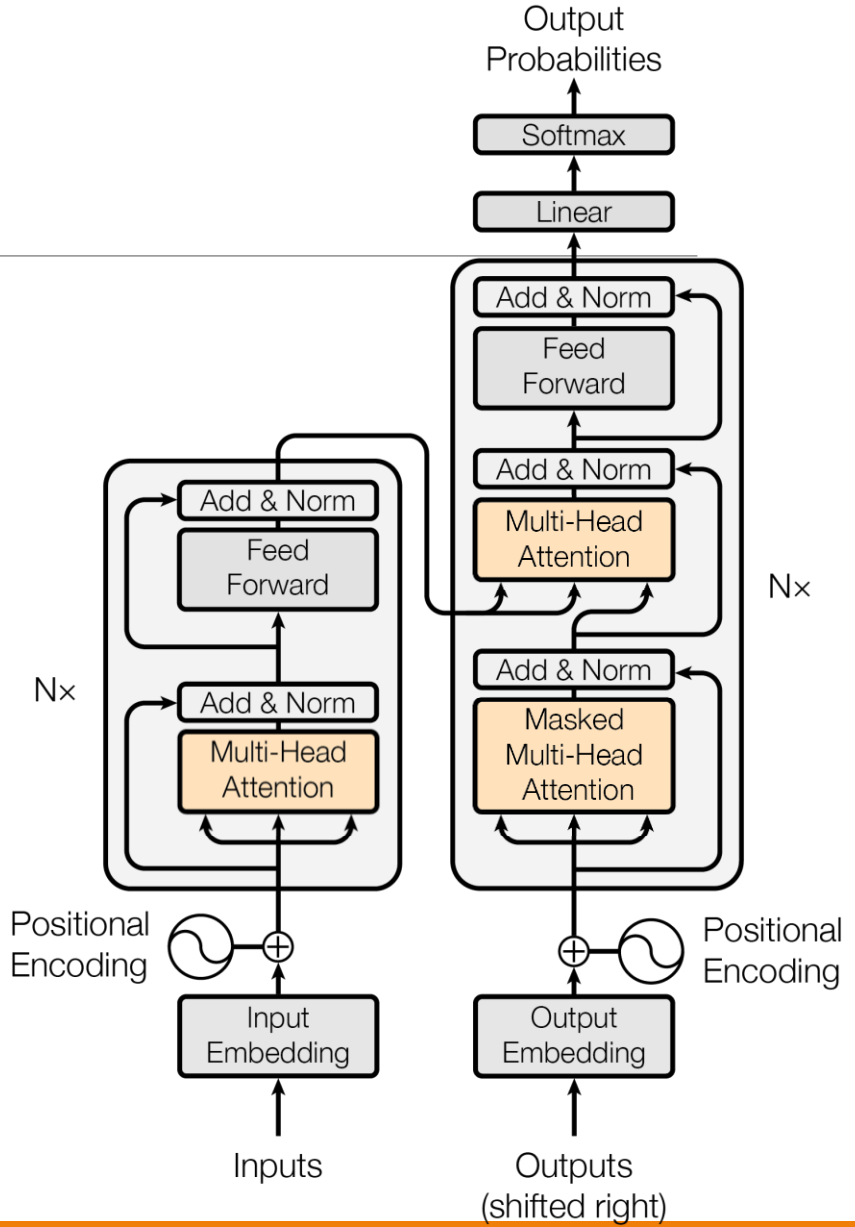
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$$\text{MultiHeadAtt}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1, \ldots \text{head}_h)\mathbf{W}^O$$

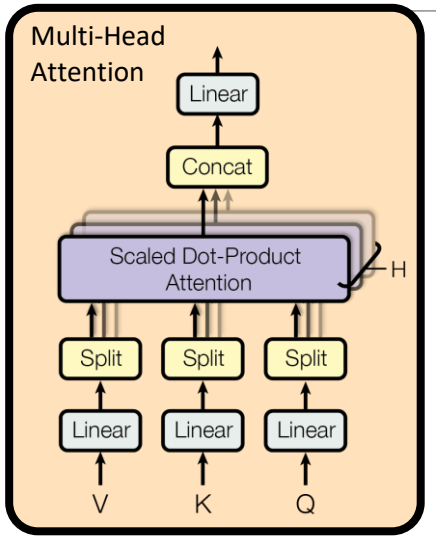
Instead of operating on **Q**, **K**, and **V** mechanism projects each input into a smaller dimension. This is done h times.

The attention operation is performed on each of these "heads," and the results are concatenated.

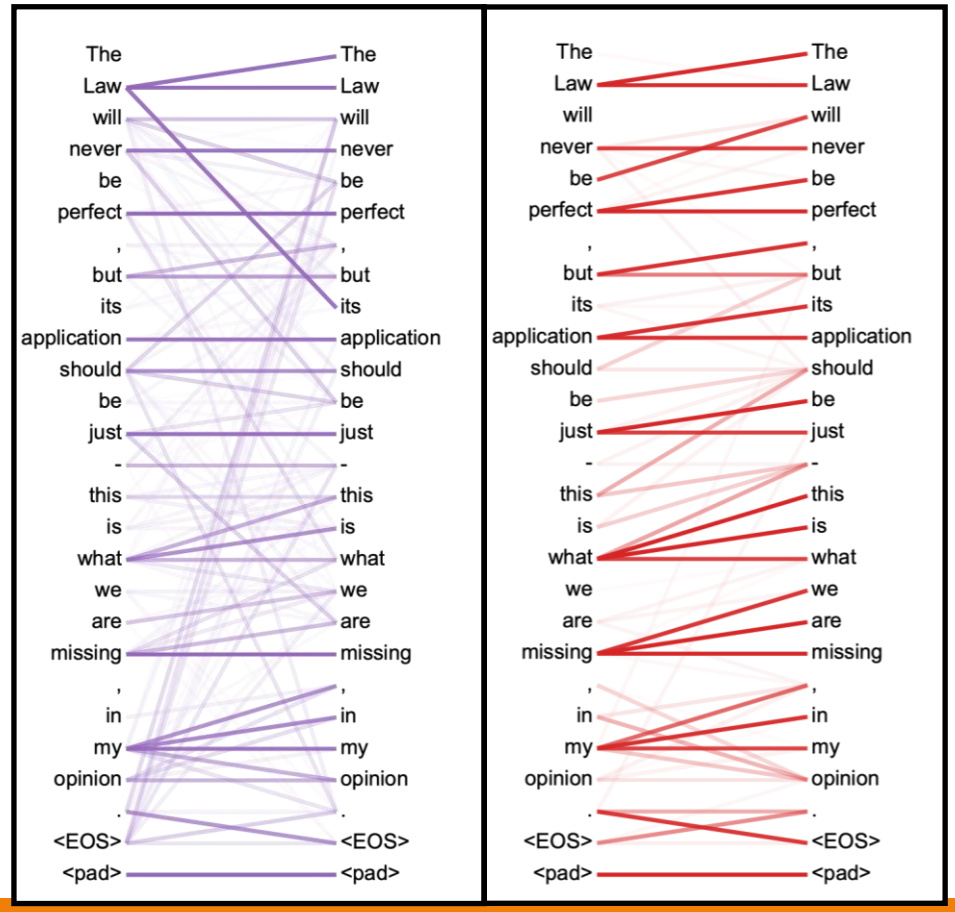
Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

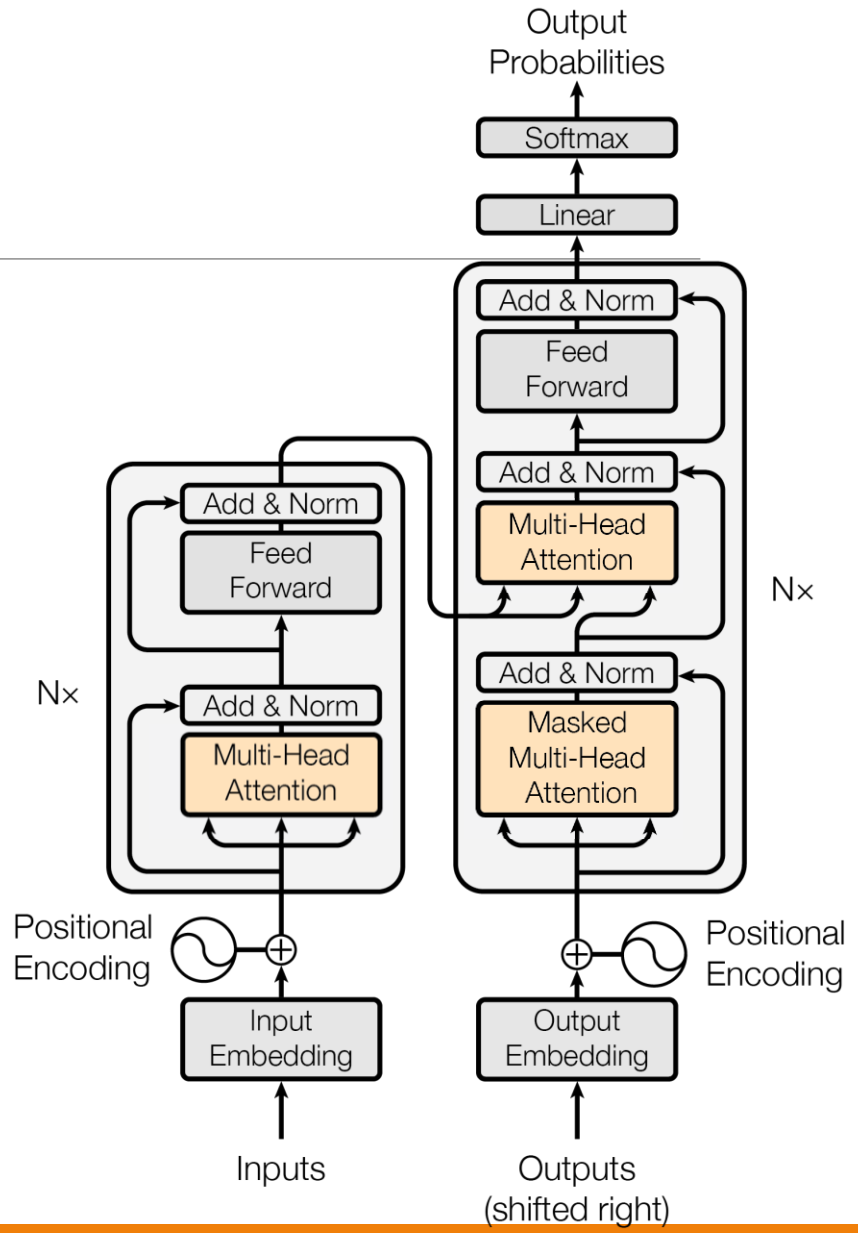


Multi-Head Attention



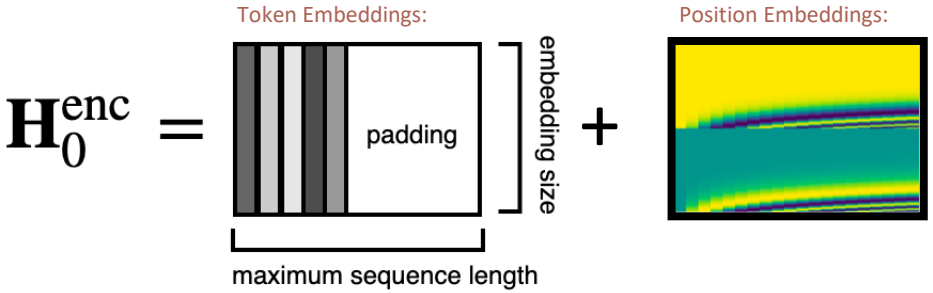
Two different self-attention heads:





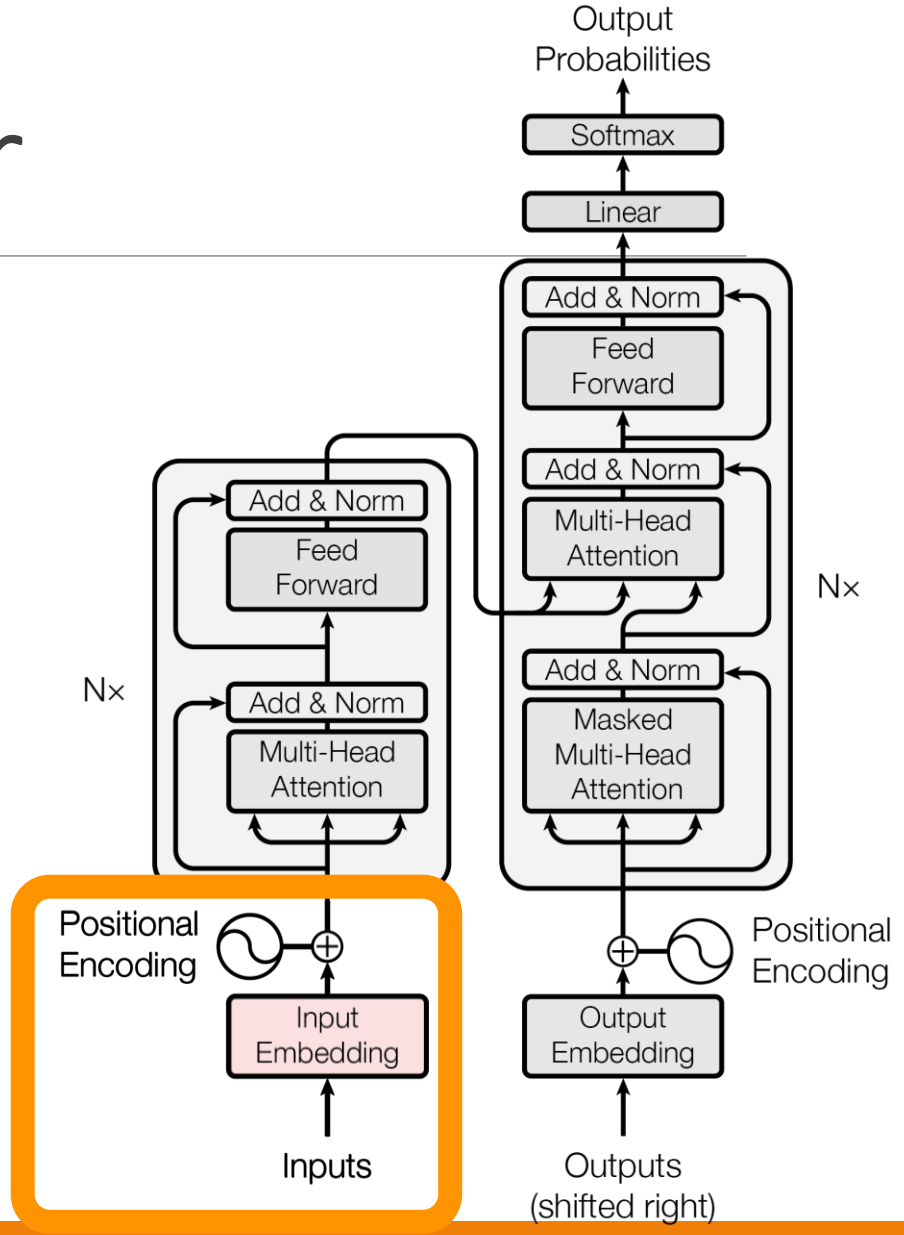
Inputs to the Encoder

The input into the encoder looks like:



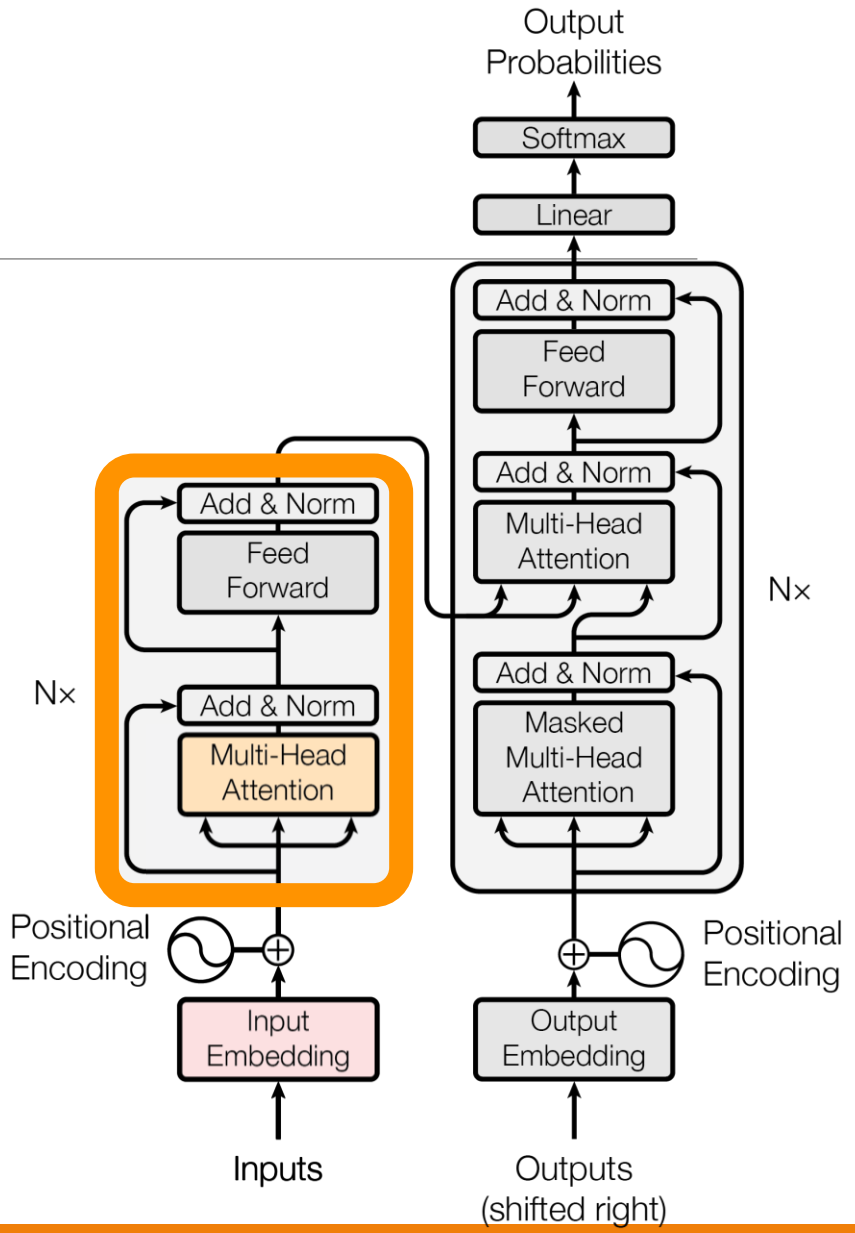
$$\mathbf{H}_0^{enc} =$$

= token embeddings + position embeddings



The Encoder

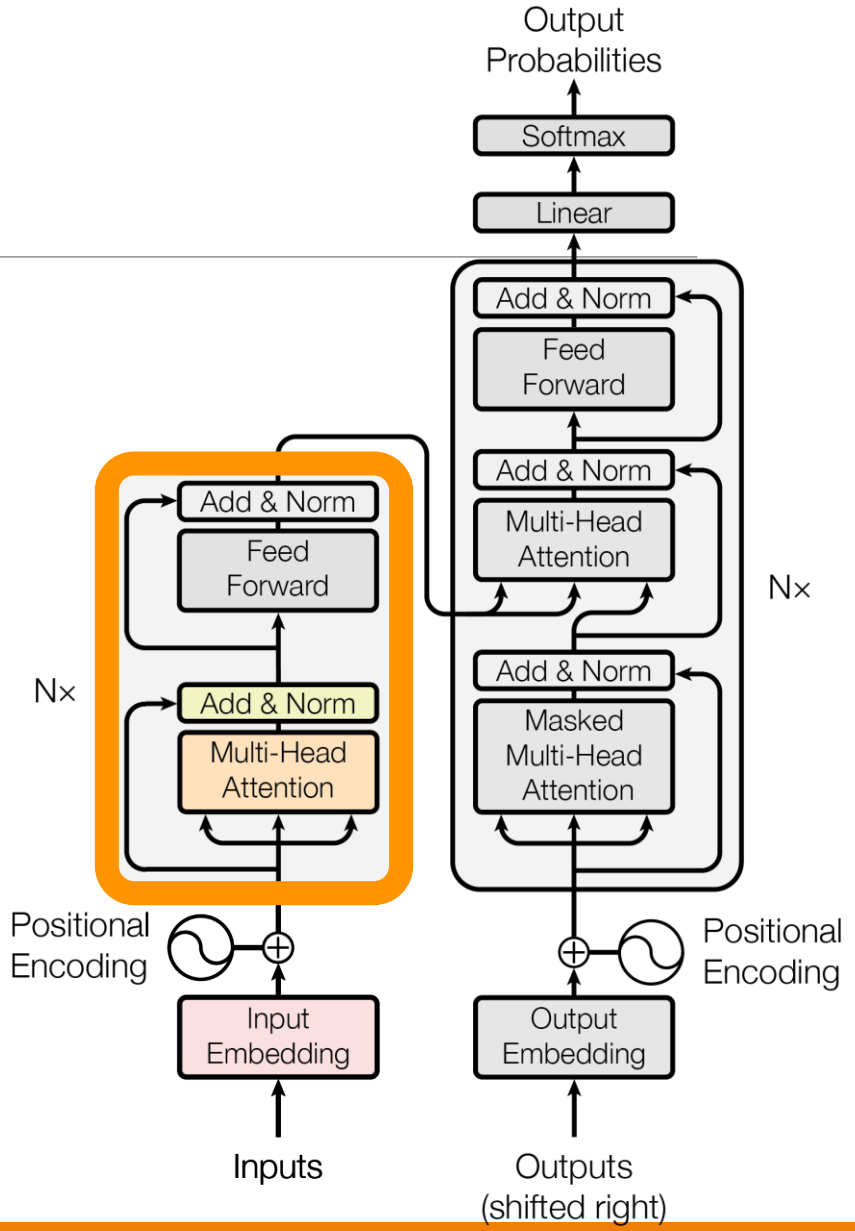
Multi-Head Attention = $\text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$



The Encoder

Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$

Add & Norm $= \text{LayerNorm}(\text{Multi-Head Attention} + \mathbf{H}_i^{enc})$



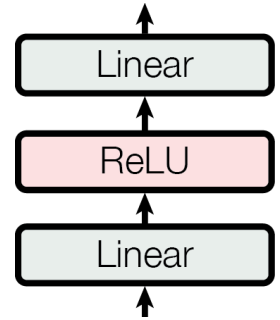
The Encoder

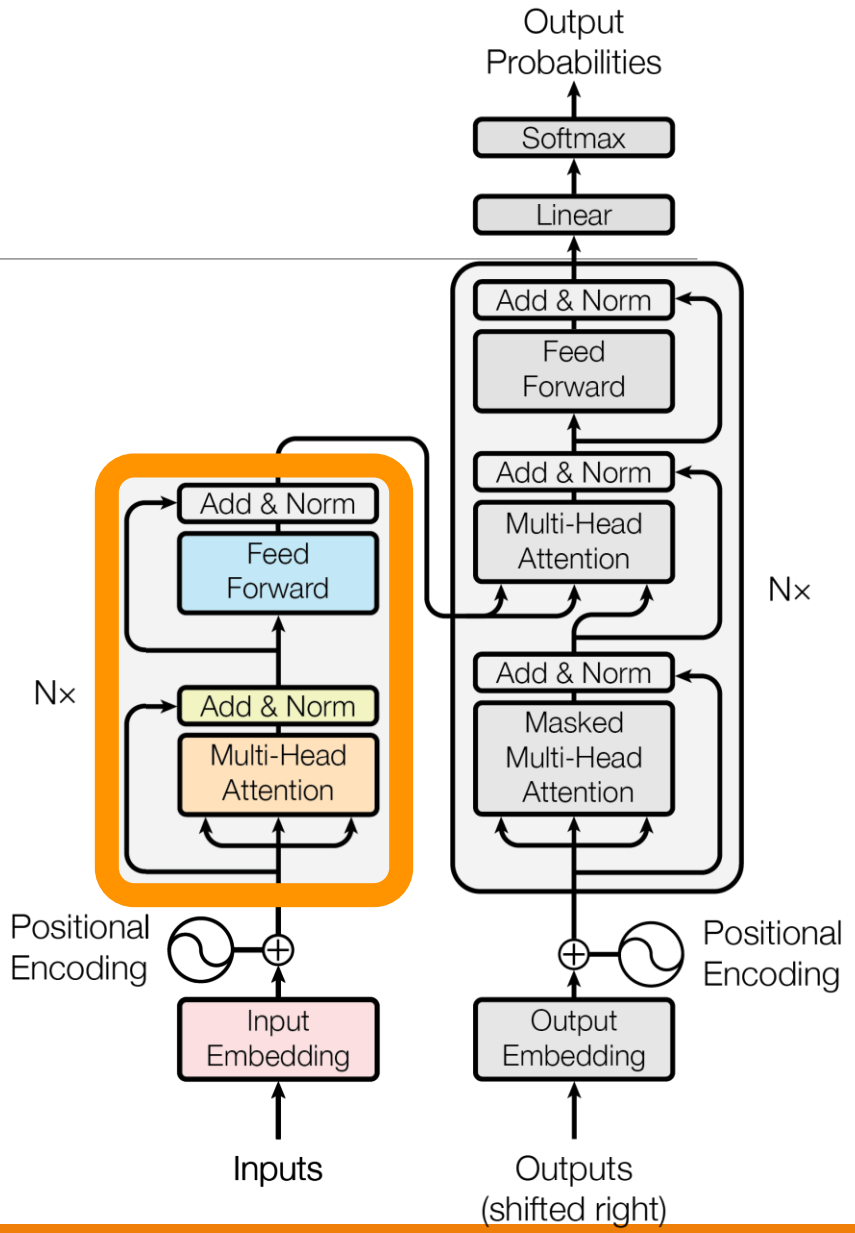
Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)

Add & Norm = LayerNorm(Multi-Head Attention + $\mathbf{H}_i^{enc}$)

Feed Forward = $max(0,$ Add & Norm $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

Feed Forward <=>





The Encoder

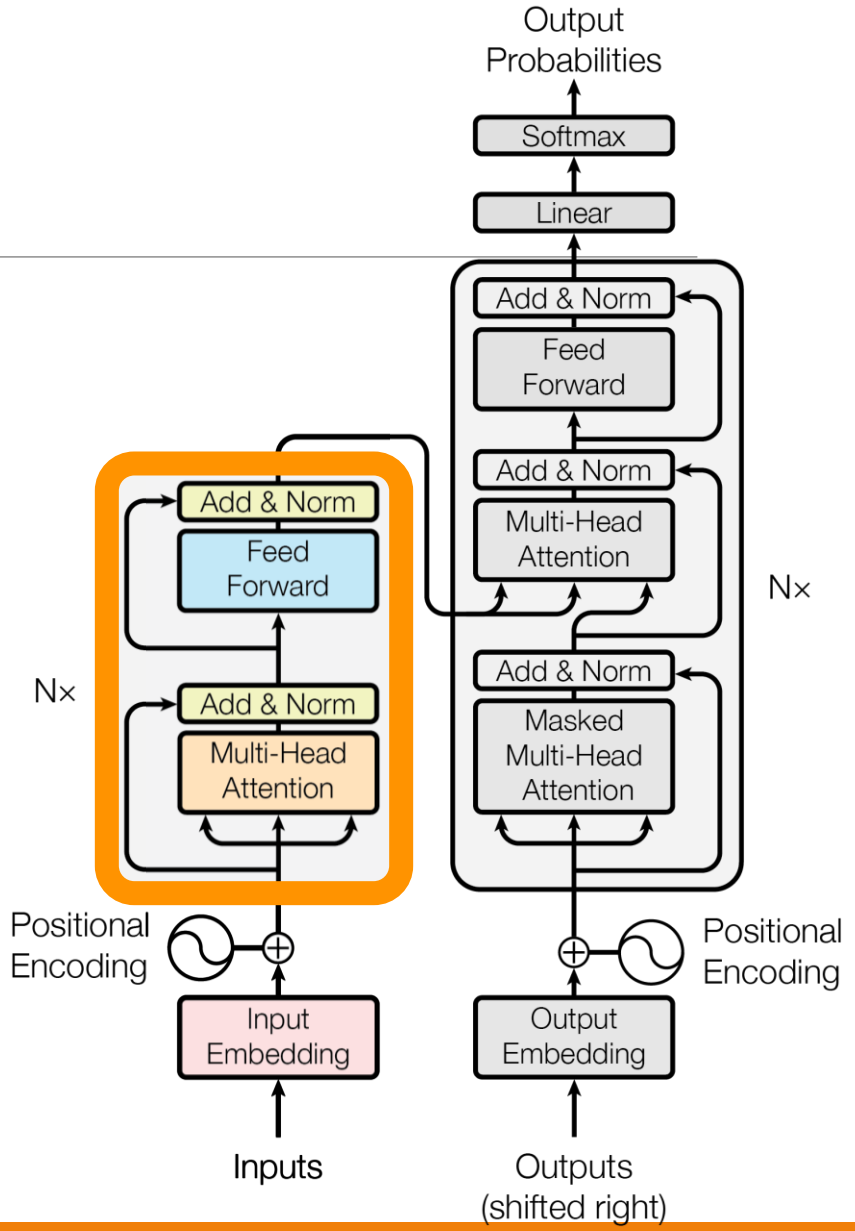
Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$

Add & Norm $= \text{LayerNorm}(\text{Multi-Head Attention} + \mathbf{H}_i^{enc})$

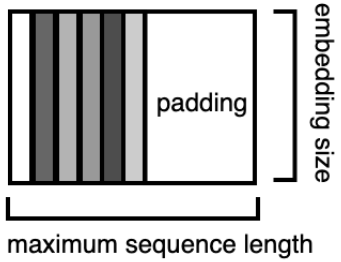
Feed Forward $= max(0, \text{Add & Norm}\,\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

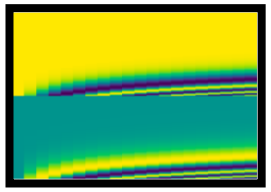
Add & Norm (2) $= \text{LayerNorm}(\text{Feed Forward} + \mathbf{H}_i^{enc})$

$\mathbf{H}_{i+1}^{enc} =$ Add & Norm (2)

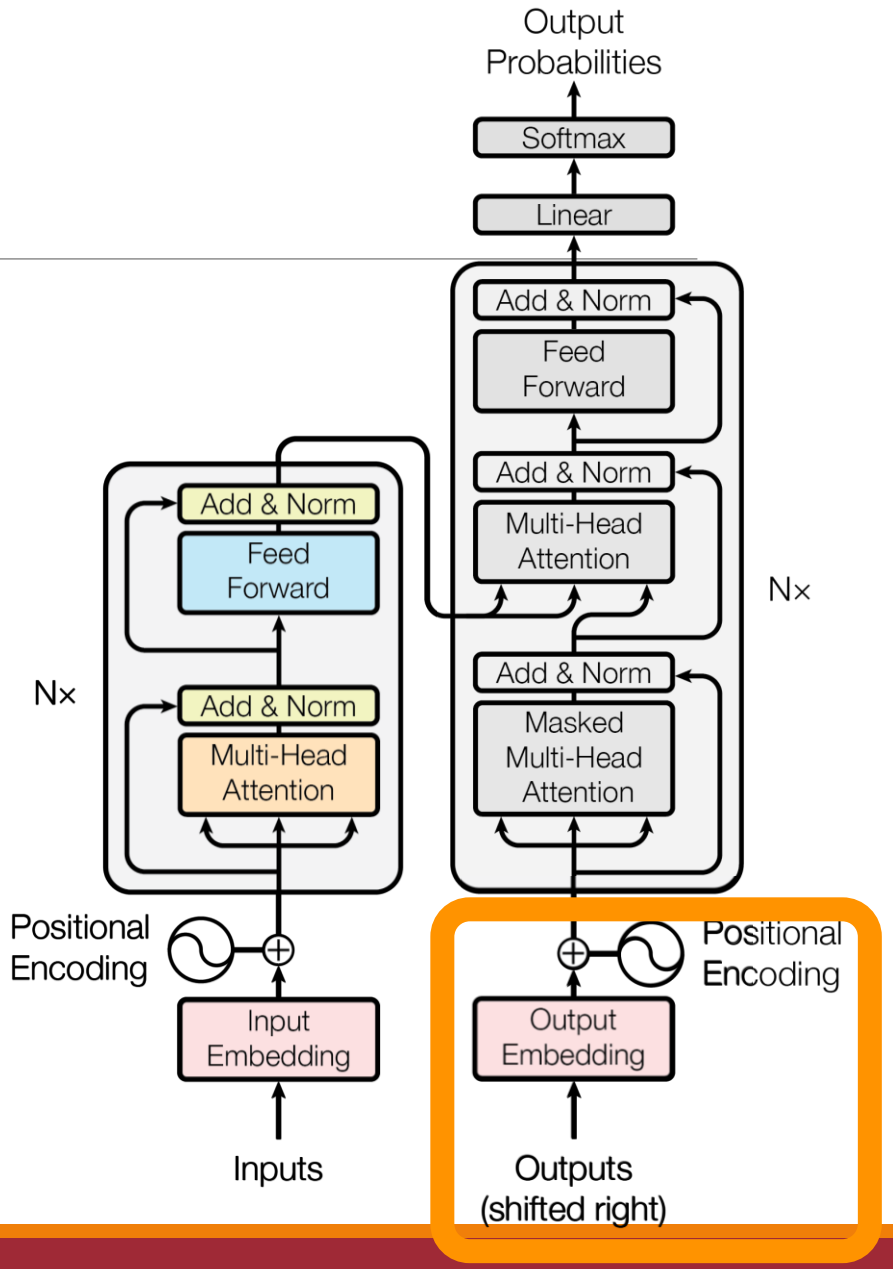


The Decoder

$$\mathbf{H}_0^{dec} =$$ 

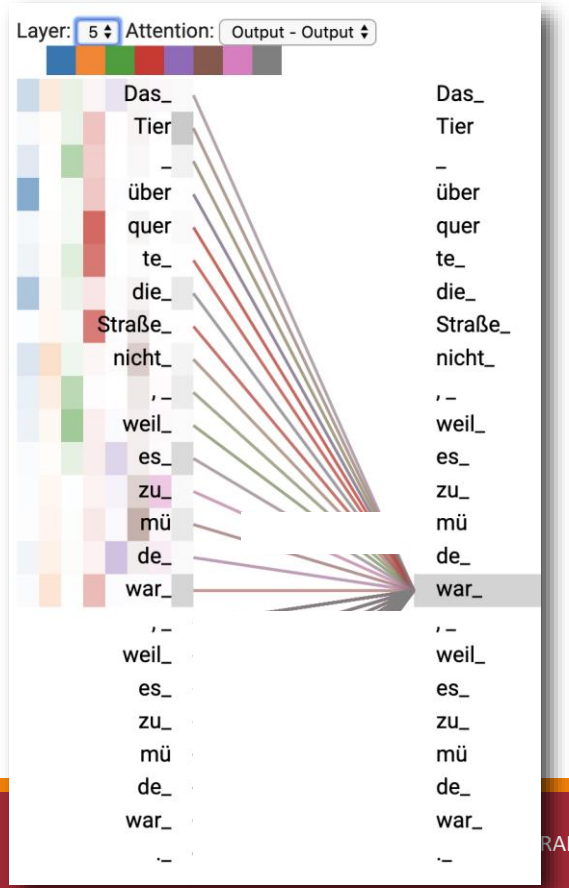
+ 

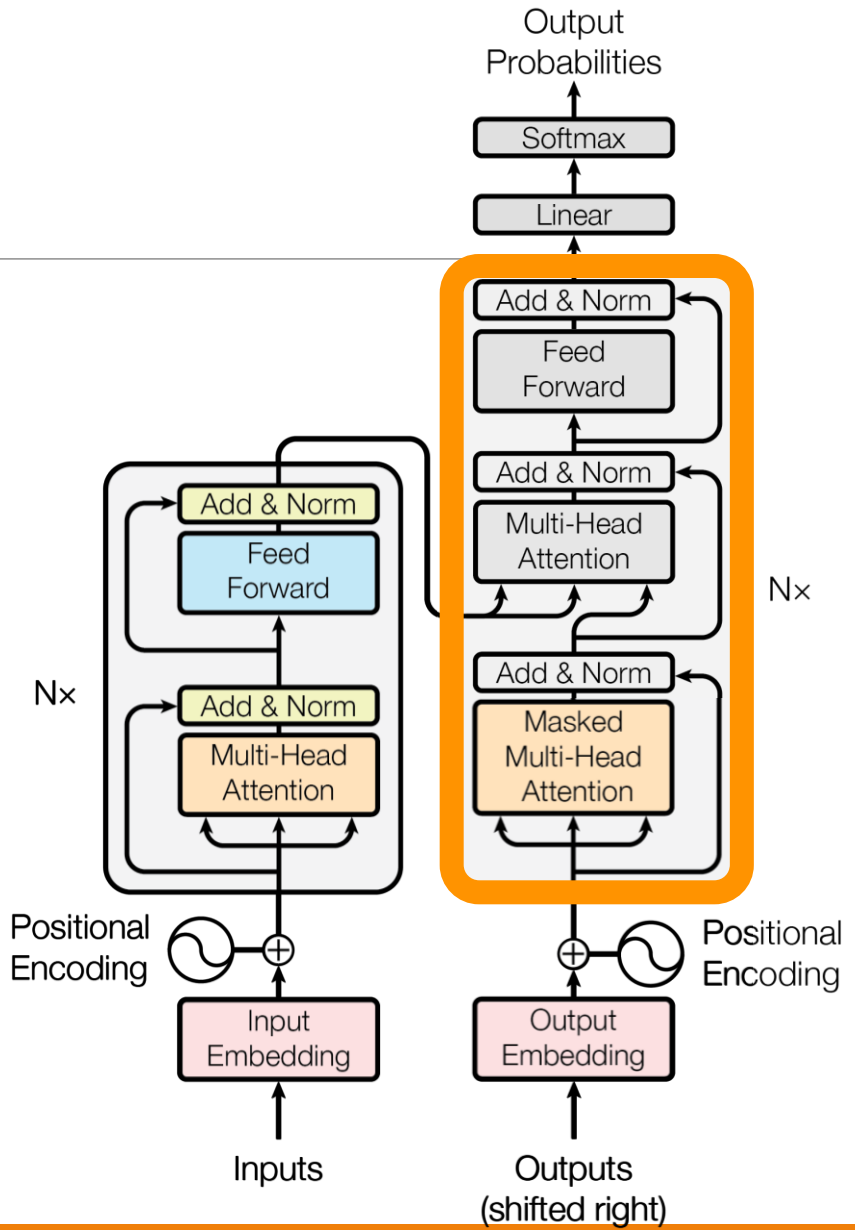
= token embeddings + position embeddings



The Decoder

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

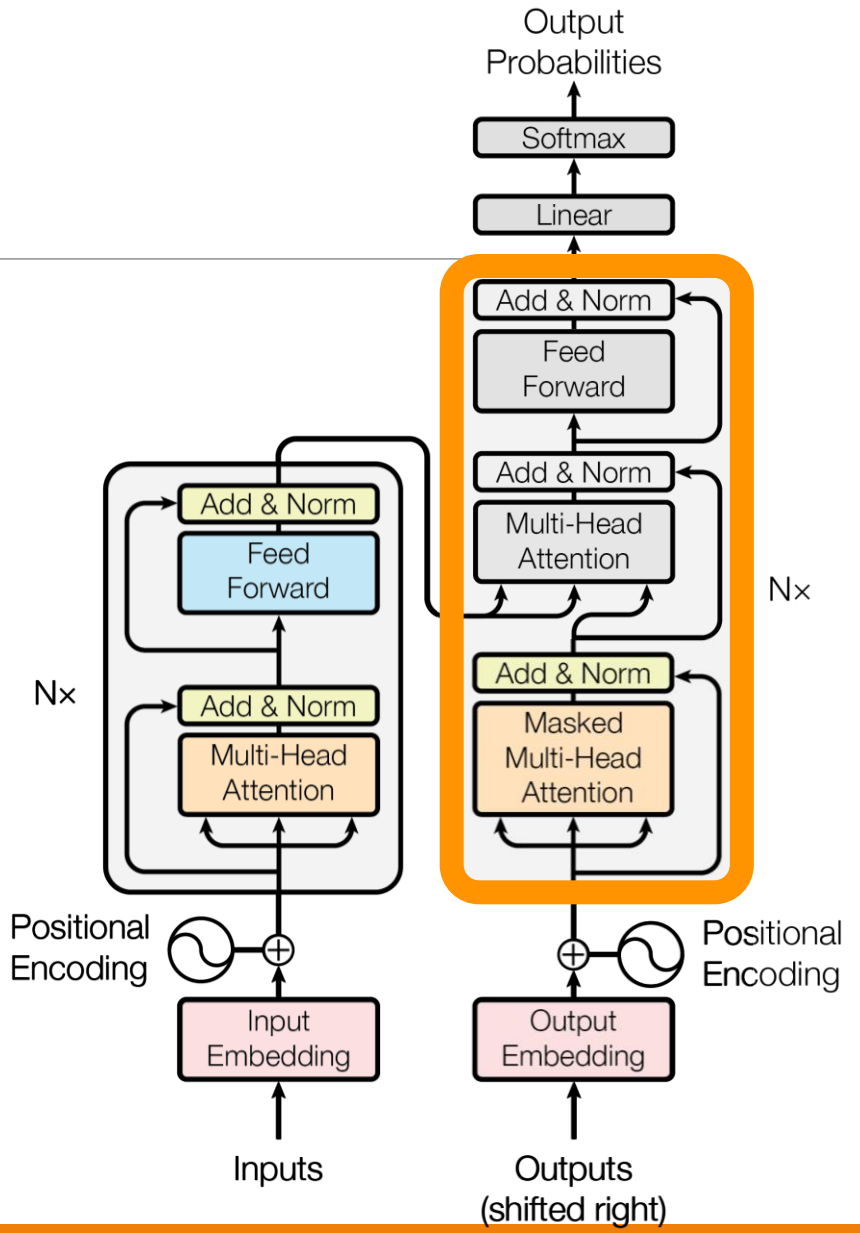




The Decoder

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

Add & Norm $= \text{LayerNorm}(\text{Multi-Head Attention} + \mathbf{H}_i^{dec})$

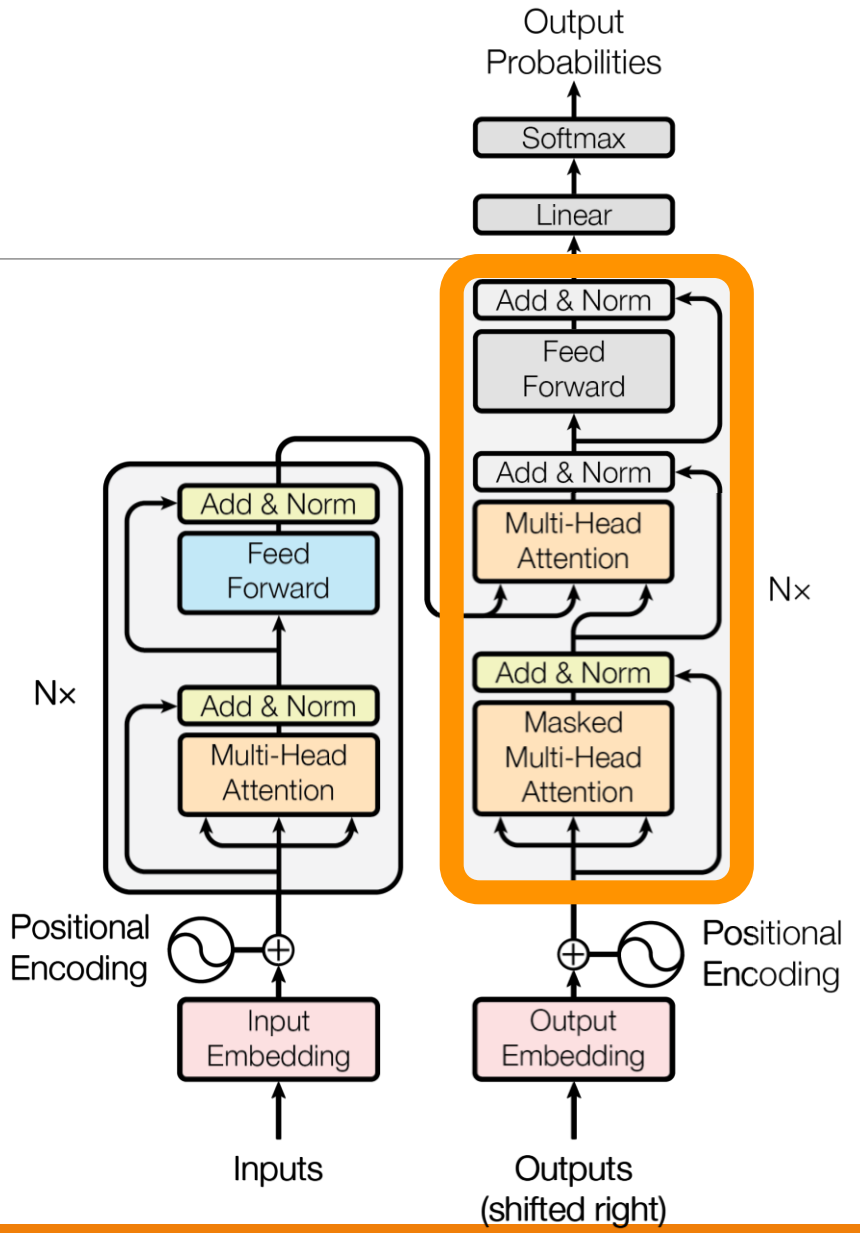


The Decoder

Masked Multi-Head Attention = MaskedMultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}$)

Add & Norm = LayerNorm(Multi-Head Attention + $\mathbf{H}_i^{dec}$)

Enc-Dec Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)



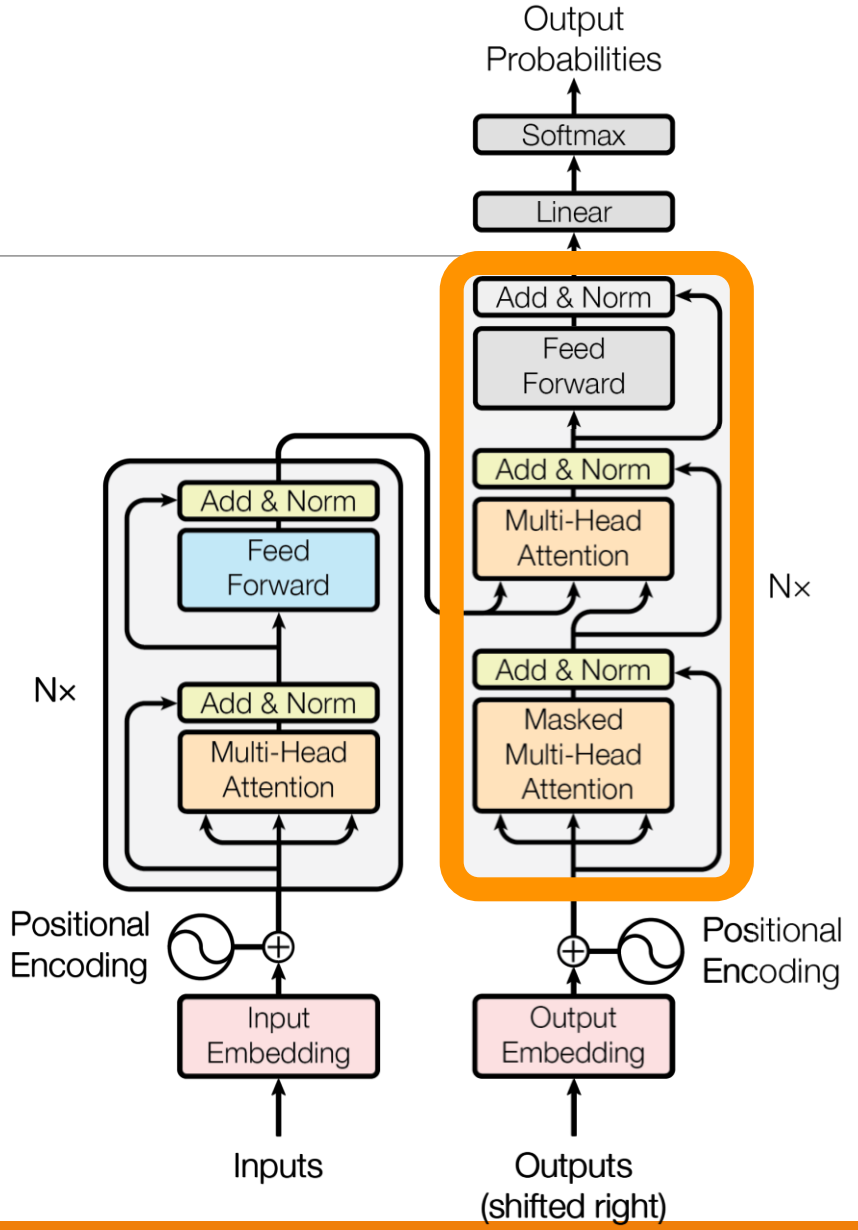
The Decoder

Masked Multi-Head Attention $= \text{MaskedMultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec})$

Add & Norm $= \text{LayerNorm}(\text{Multi-Head Attention} + \mathbf{H}_i^{dec})$

Enc-Dec Multi-Head Attention $= \text{MultiHeadAtt}(\mathbf{H}_i^{dec}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc})$

Add & Norm (2) $= \text{LayerNorm}(\text{Multi-Head Attention} + \text{Add \& Norm})$



The Decoder

Masked Multi-Head Attention = MaskedMultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}, \mathbf{H}_i^{dec}$)

Add & Norm = LayerNorm(**Multi-Head Attention** + $\mathbf{H}_i^{dec}$)

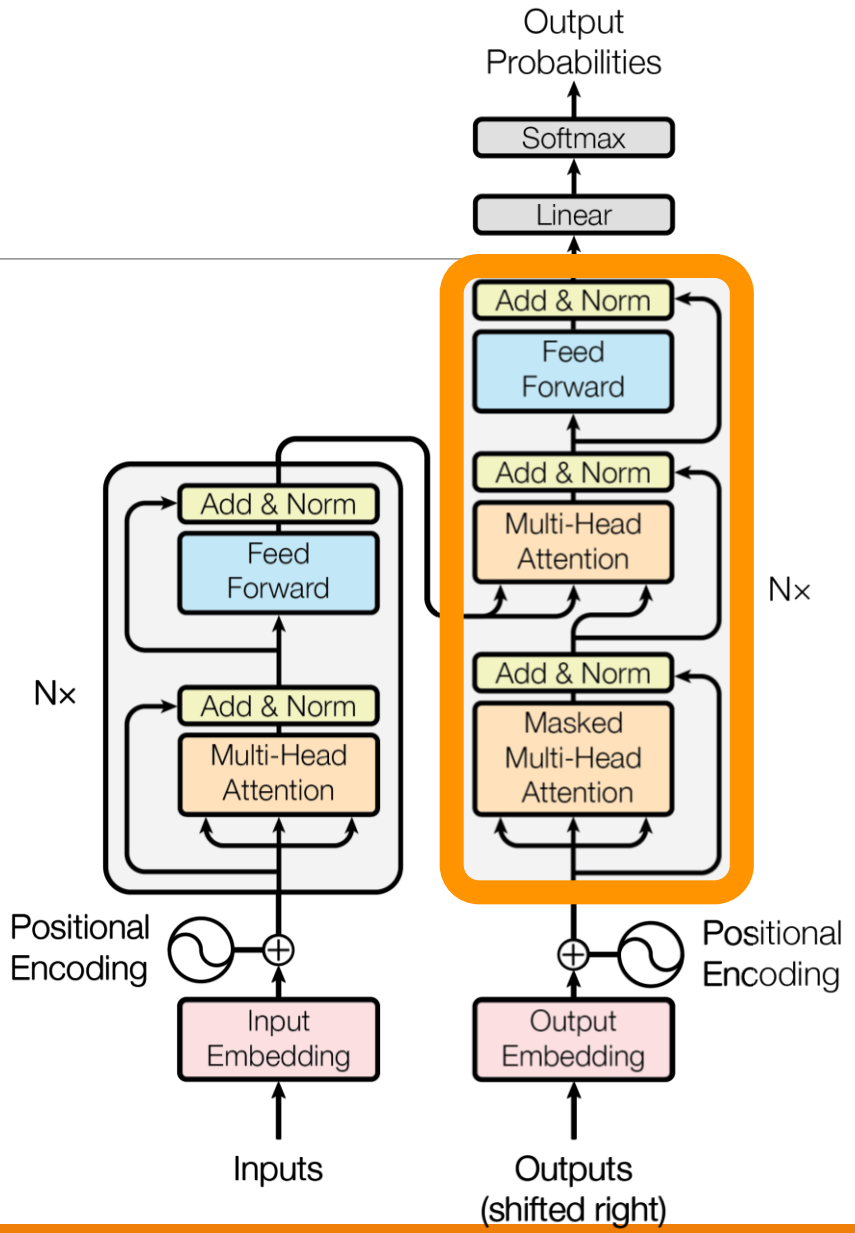
Enc-Dec Multi-Head Attention = MultiHeadAtt($\mathbf{H}_i^{dec}, \mathbf{H}_i^{enc}, \mathbf{H}_i^{enc}$)

Add & Norm (2) = LayerNorm(**Multi-Head Attention** + **Add & Norm**)

Feed Forward = $max(0,$ **Add & Norm (2)** $\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$

Add & Norm (3) = LayerNorm(**Feed Forward** + **Add & Norm (2)**)

$\mathbf{H}_{i+1}^{dec}$ = **Add & Norm (3)**



Strengths of the Transformer Architecture

Training is easily parallelizable

- Larger models can be trained efficiently.

Does not “forget” information from earlier in the sequence.

- Any position can attend to any position.

What are some of its weaknesses?

For more information

The Illustrated Transformer

Foundation Models

Types of Foundation Models

Encoder Only

Decoder Only

Encoder-Decoder Models

What is a foundation model?

A model that captures “foundation” or core information about a modality (e.g., text, speech, images)

Pretrained on a large amount of data & able to *be* finetuned on a particular task

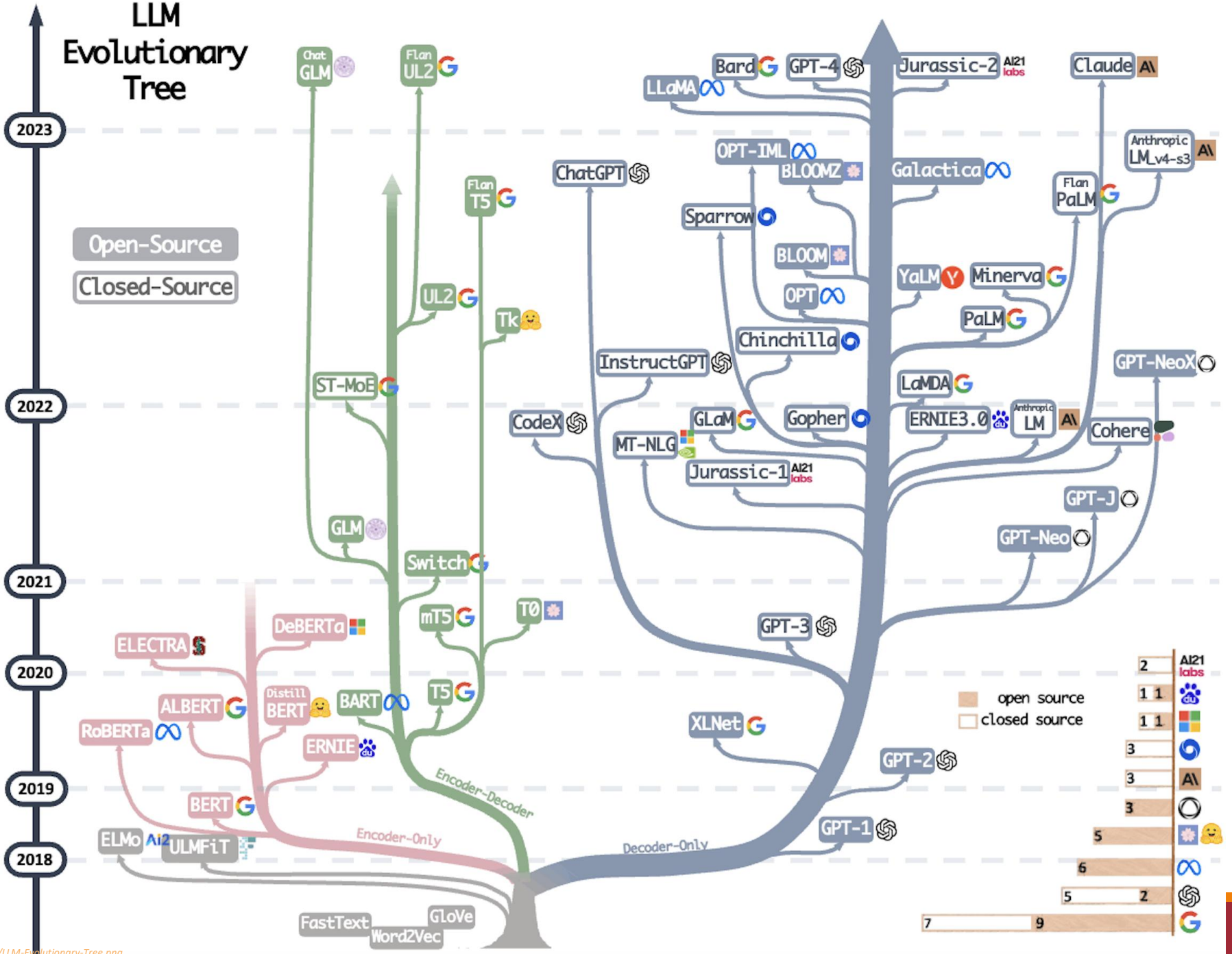
Self-supervised

All non-finetuned large language models (LLMs) are foundation models

Some Models Come Fine-tuned

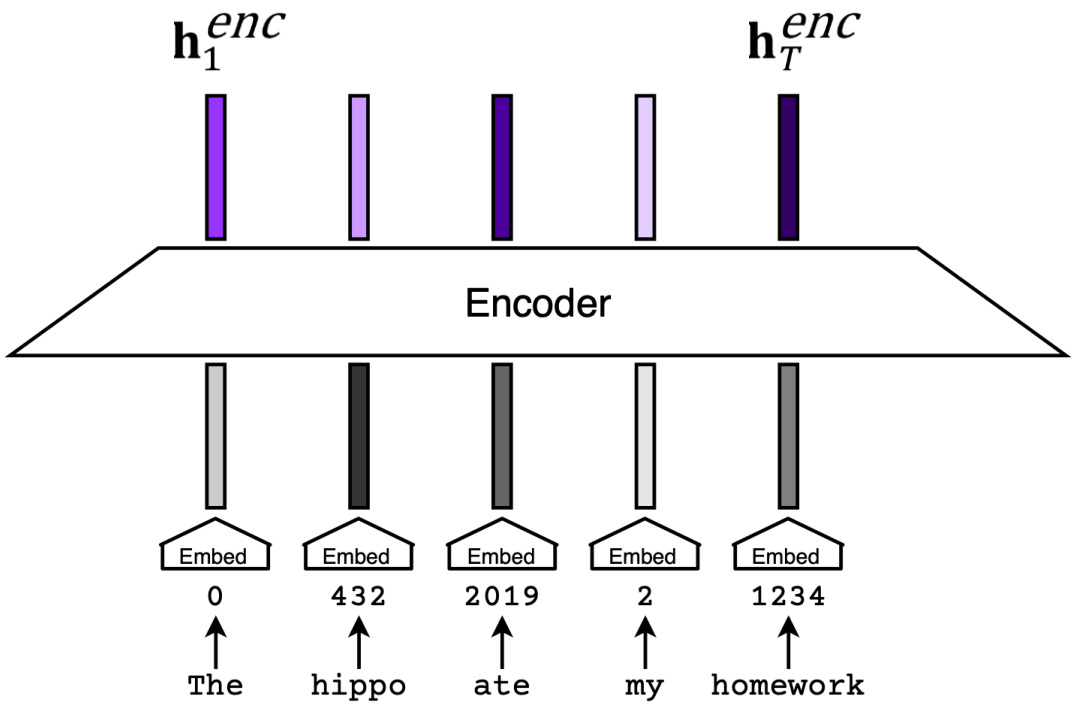
ChatGPT/InstructGPT

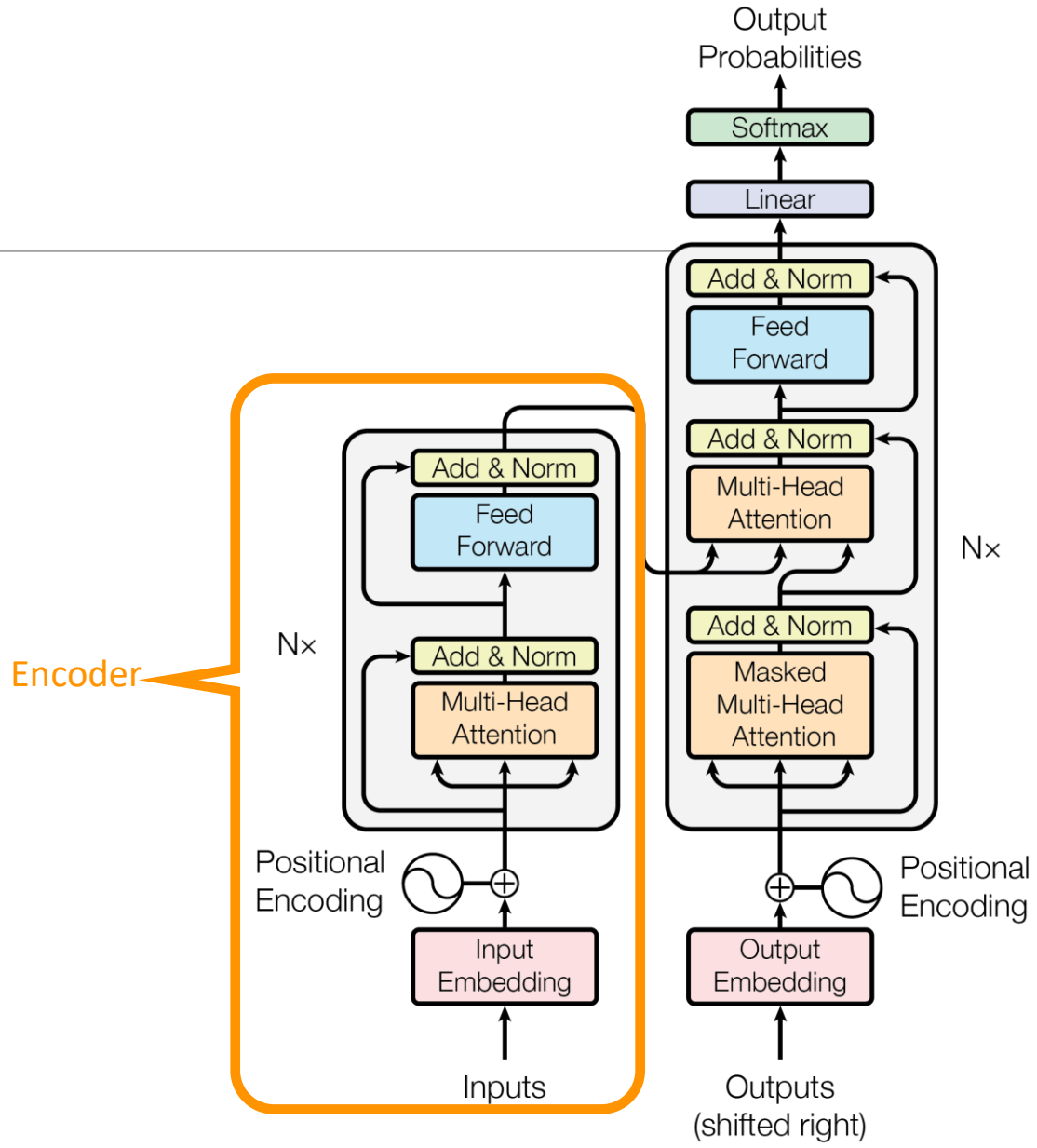
Most/all “Instruct” or “Chat” models

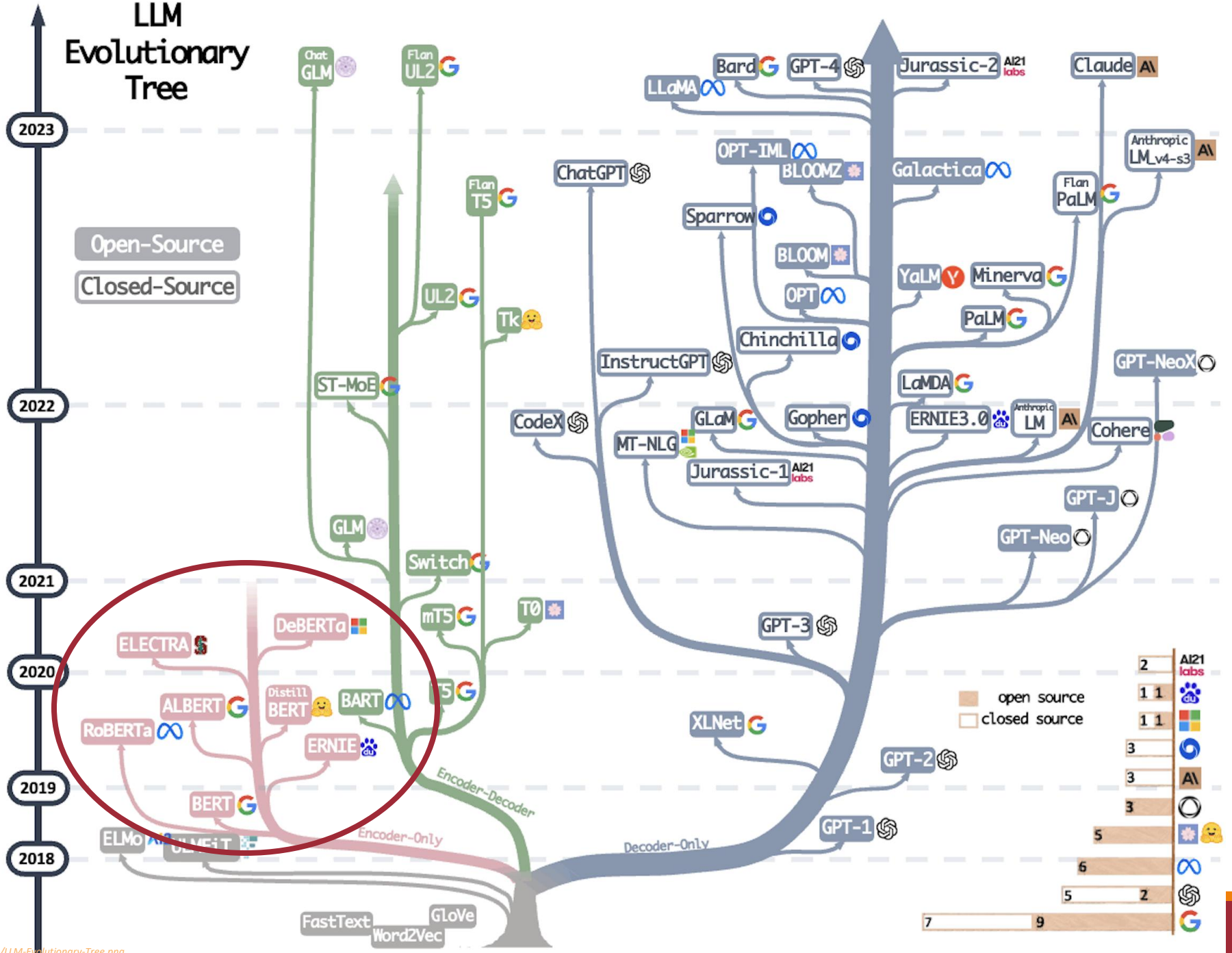


https://blog.biocomm.ai/wp-content/uploads/2023/05/LLM-Evolutionary-Tree.png

Encoder-only models

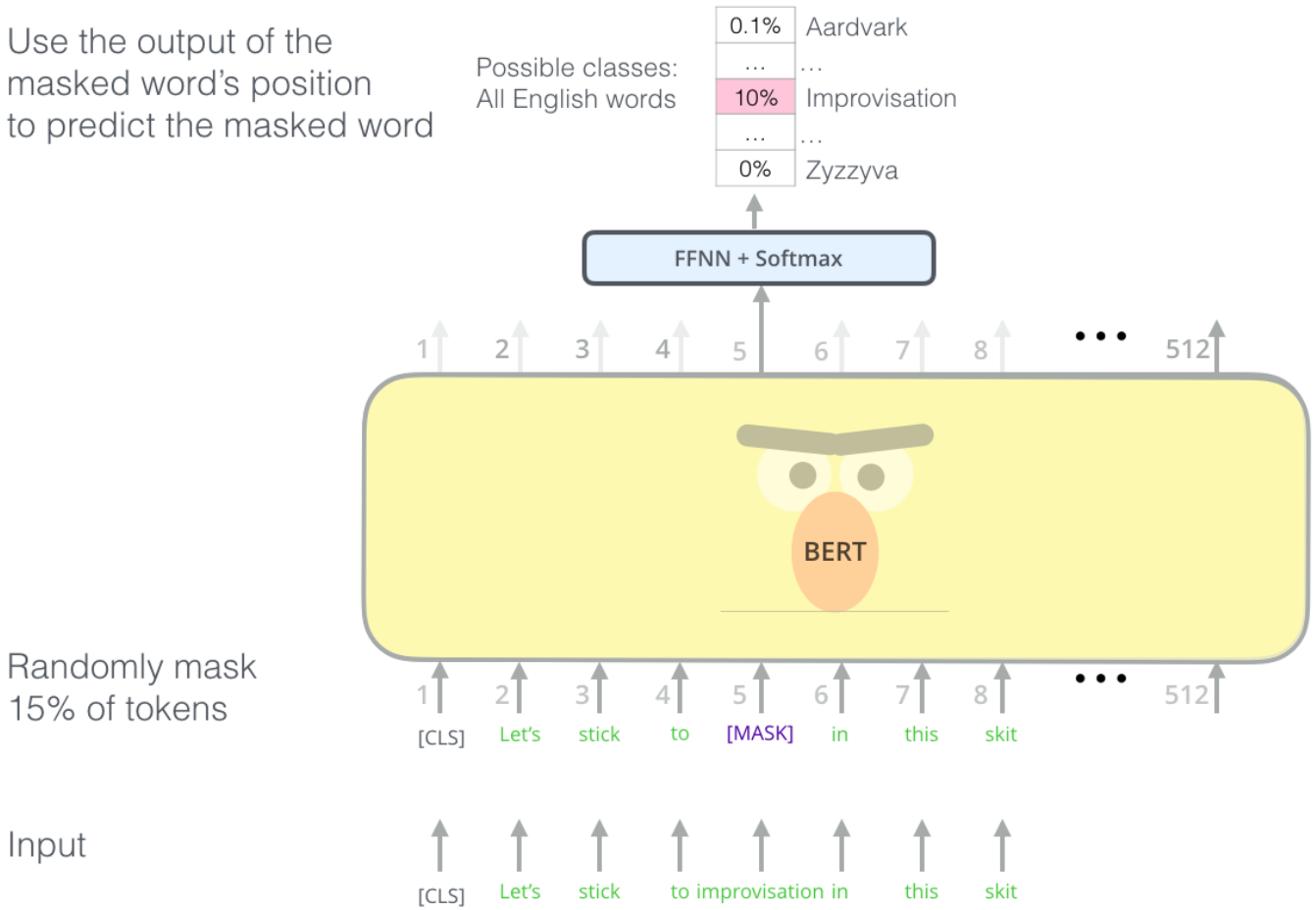






https://blog.biocomm.ai/wp-content/uploads/2023/05/LLM-Evolutionary-Tree.png

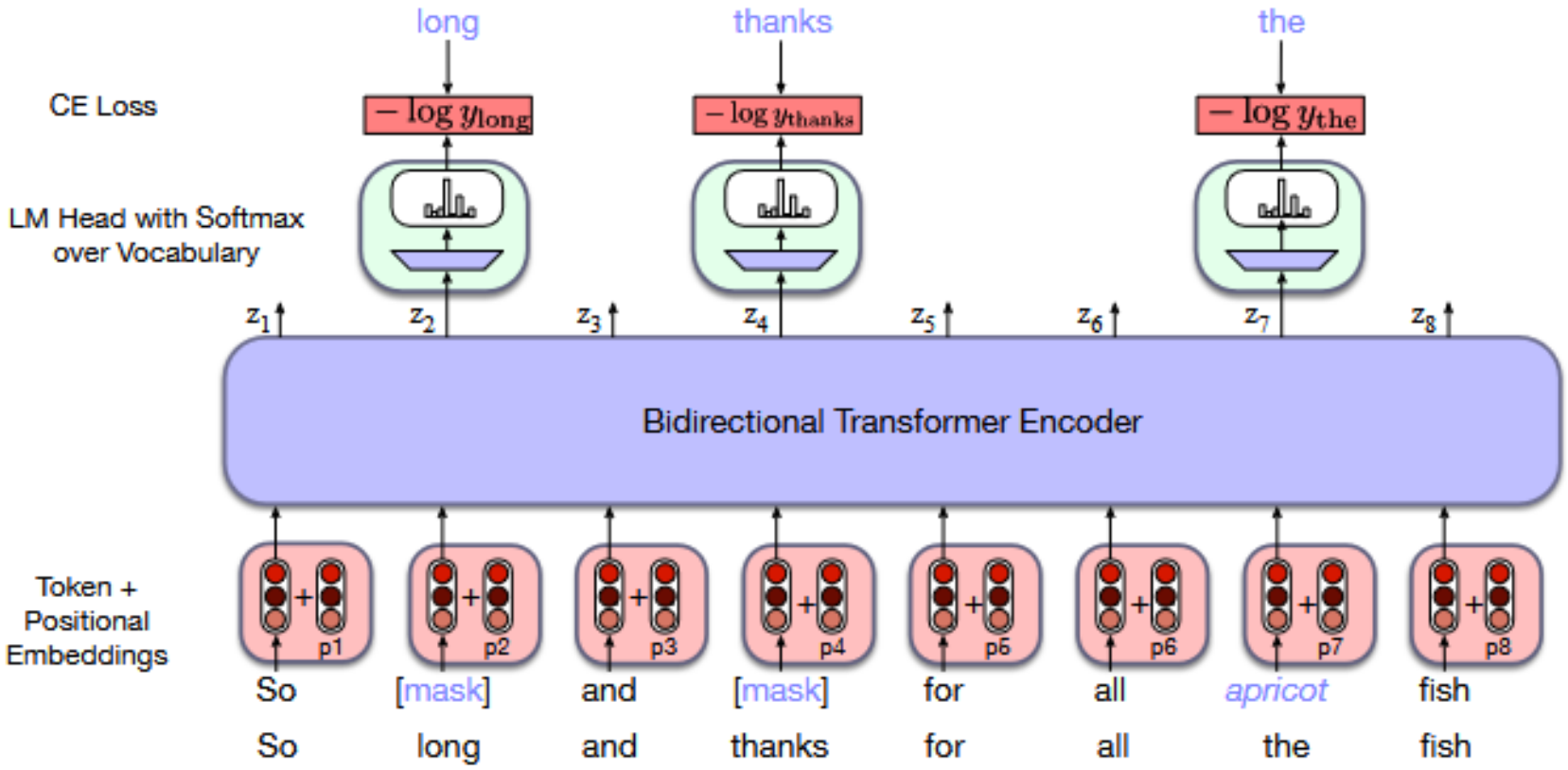
BERT (Devlin et al. 2019)



http://jalammar.github.io/illustrated-bert/

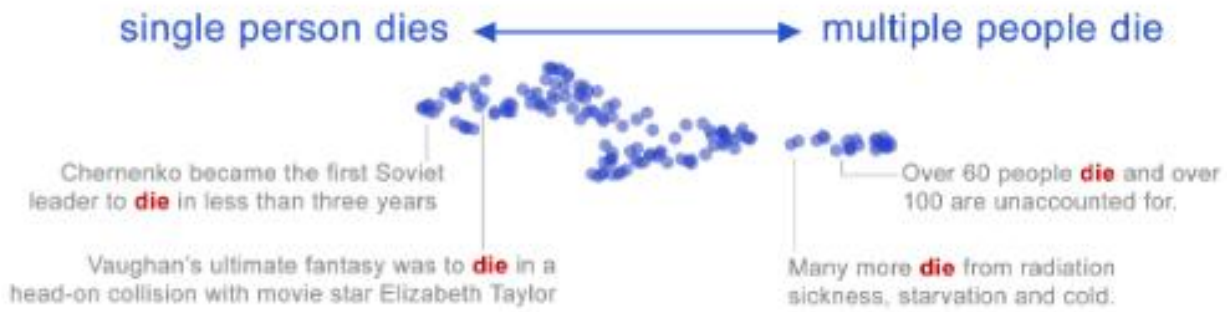
Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Masked Language Models



From Jurafsky & Martin's Speech and Language Processing, 3rd Edition, Chapter 11

Contextual Embeddings



From Jurafsky & Martin's Speech and Language Processing, 3rd Edition, Chapter 11

Uses of Encoder-Only Models

Classification tasks

Sentence embeddings

Context-dependent word embeddings

Any type of fill-in-the-blank tasks

BERT Question

Consider the highlighted words. Which two words would contextual word embeddings from BERT say are closest?

A. I am so excited to use my new **bat** at the baseball game tomorrow.

B. The favorite food of this species of **bat** is mosquitoes.

C. The **cardinal** isn't just a lawn decoration; the species makes themselves useful by eating mosquitoes.

PollEv.com/laramartin527

Word2Vec Question

Remember: word2vec is a dense vector embedding

Consider the highlighted words. Which two words would word2vec say are closest?

A. I am so excited to use my new **bat** at the baseball game tomorrow.

B. The favorite food of this species of **bat** is mosquitoes.

C. The **cardinal** isn't just a lawn decoration; the species makes themselves useful by eating mosquitoes.

PollEv.com/laramartin527

BERT Family of Models

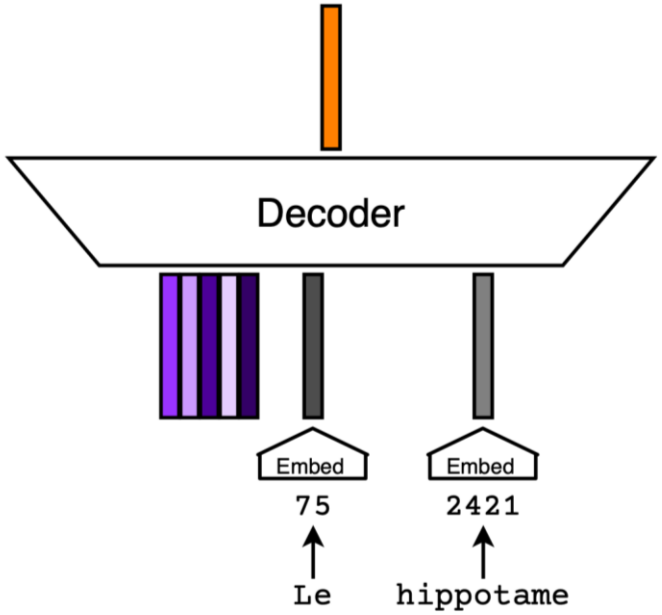
- Encoder-only
  - Input: Corrupted version of text sequence
  - Goal: Produce an uncorrupted version of text sequence
- How to use:
  - Finetune for a classification task
  - Extract word/sentence embeddings

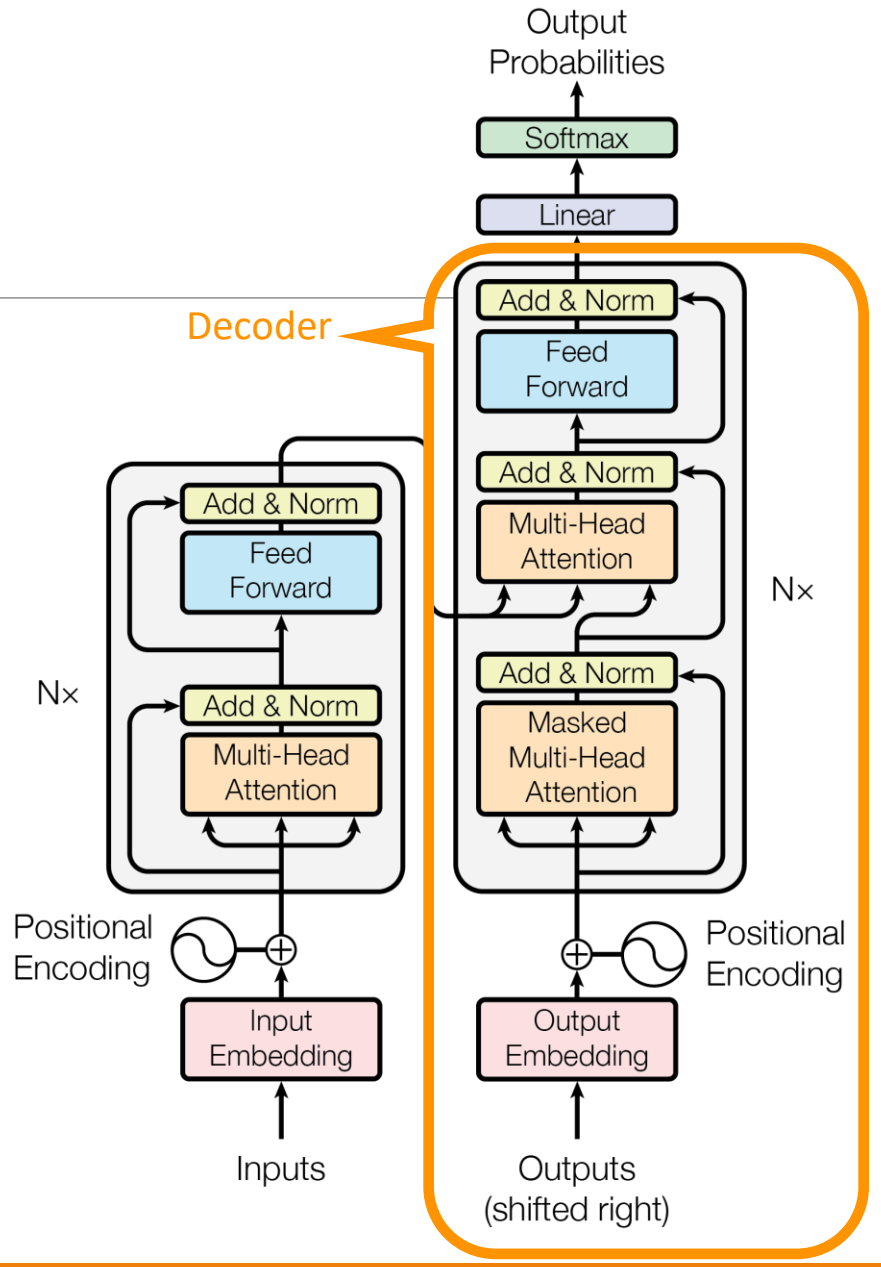
Some important BERT family members

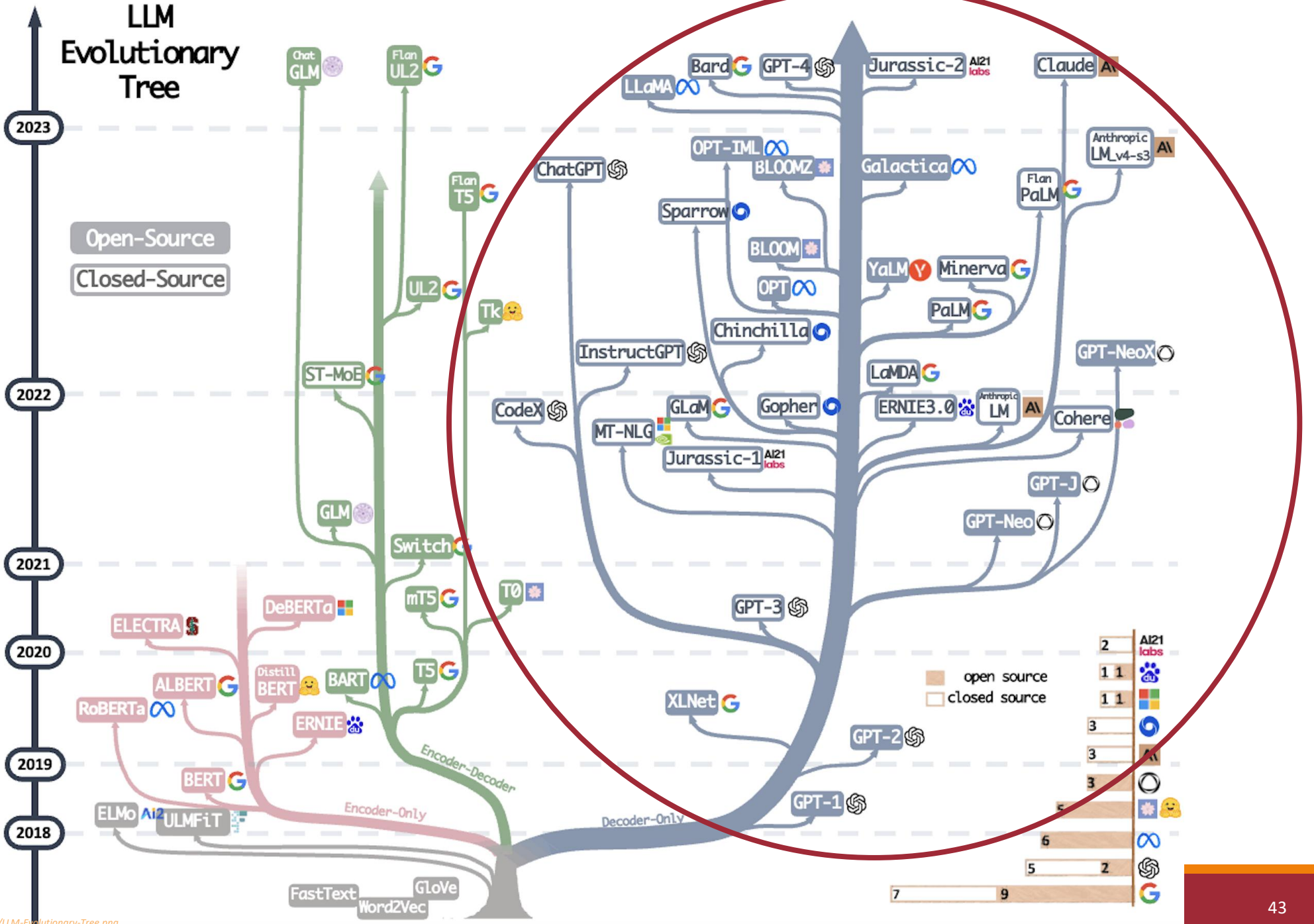
(in my opinion)

- RoBERTa (better version of the original BERT) – Liu et al. 2019 (Facebook)
- Sentence-BERT (BERT fine-tuned to give good sentence embeddings) – Reimers & Gurevych 2019 (Technische Universität Darmstadt)
- DistilBERT (lite BERT) – Sanh et al. 2019
- ALBERT (lite BERT) – Lan et al. 2020
- HuBERT (BERT for speech embeddings) – Hsu et al. 2021

Decoder-Only Models






LLM Evolutionary Tree
2023
2022
2021
2020
2019
2018
Open-Source
Closed-Source
Chat GLM
Flan UL2
Flan T5
UL2
Tk
ST-MoE
GLM
Switch
mT5
T0
DeBERTa
ELECTRA
ALBERT
Distill BERT
BART
T5
RoBERTa
ERNIE
BERT
ELMo
ULMFiT
FastText
GloVe
Word2Vec
Encoder-Only
Encoder-Decoder
Decoder-Only
LLaMA
Bard
GPT-4
Jurassic-2
Claude
OPT-IML
BLOOMZ
ChatGPT
Galactica
Anthropic LM_v4-s3
Flan PaLM
Sparrow
BLOOM
YaLM
Minerva
OPT
PaLM
Chinchilla
InstructGPT
GPT-NeoX
LaMDA
CodeX
GLaM
Gopher
ERNIE3.0
Anthropic LM
Cohere
MT-NLG
Jurassic-1
GPT-J
GPT-Neo
GPT-3
XLNet
GPT-2
GPT-1
open source
closed source
AI21 labs

GPT Family

- Decoder-only
  - Input: Text sequence
  - Goal: Predict the next word given the previous ones
- How to use:
  - Ask GPT* to continue from a prompt.
  - Finetune smaller GPTs for more customized generation tasks.
    - ChatGPT cannot be finetuned since it is already finetuned
  - Use OpenAI's API to get them to fine-tune GPT* for you.
- Around GPT-2 was when pre-trained models became popular
- Around GPT-3 was when *just* prompting became a thing

Other Decoder-Only Models

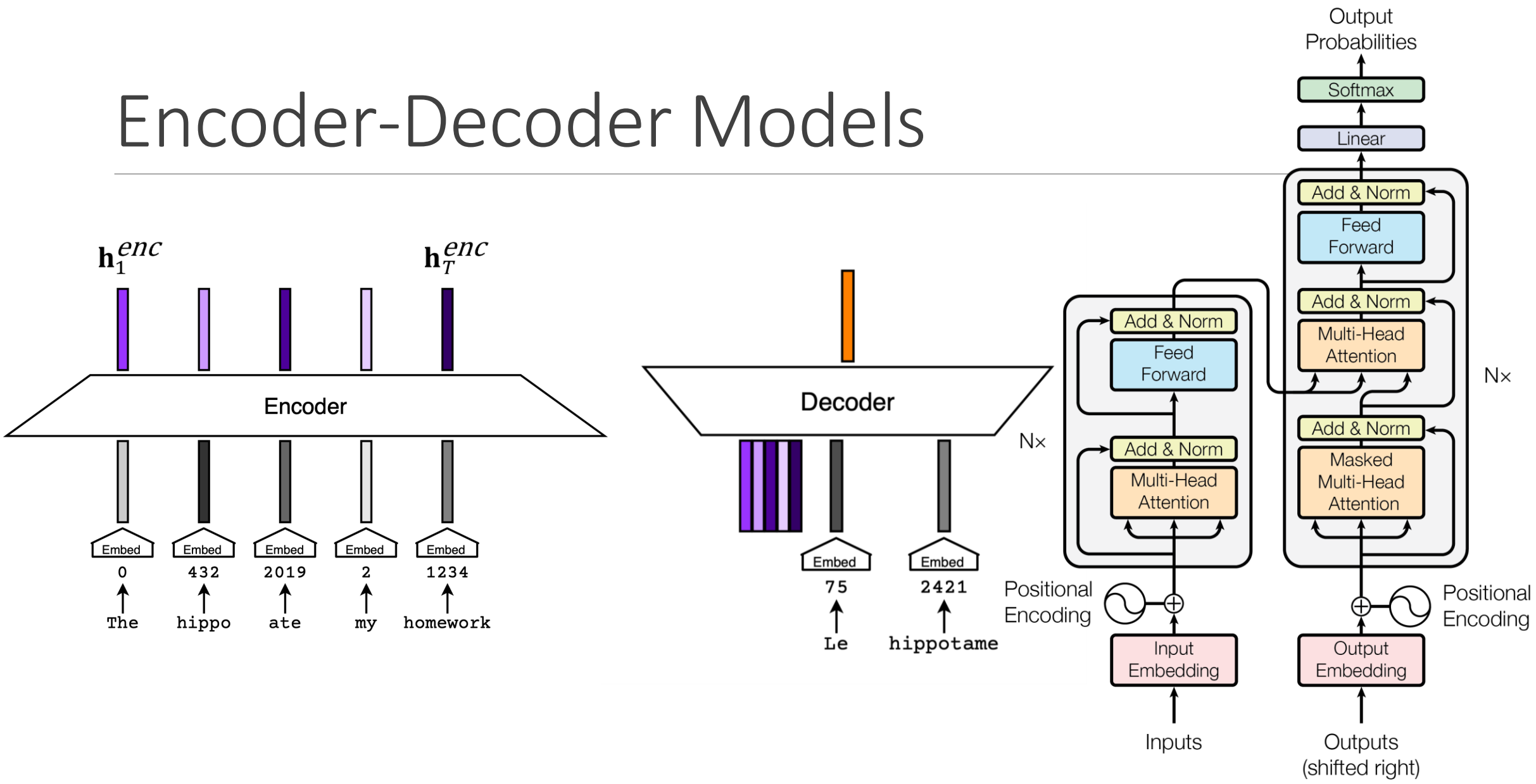
LLaMA 3/4 (Meta)

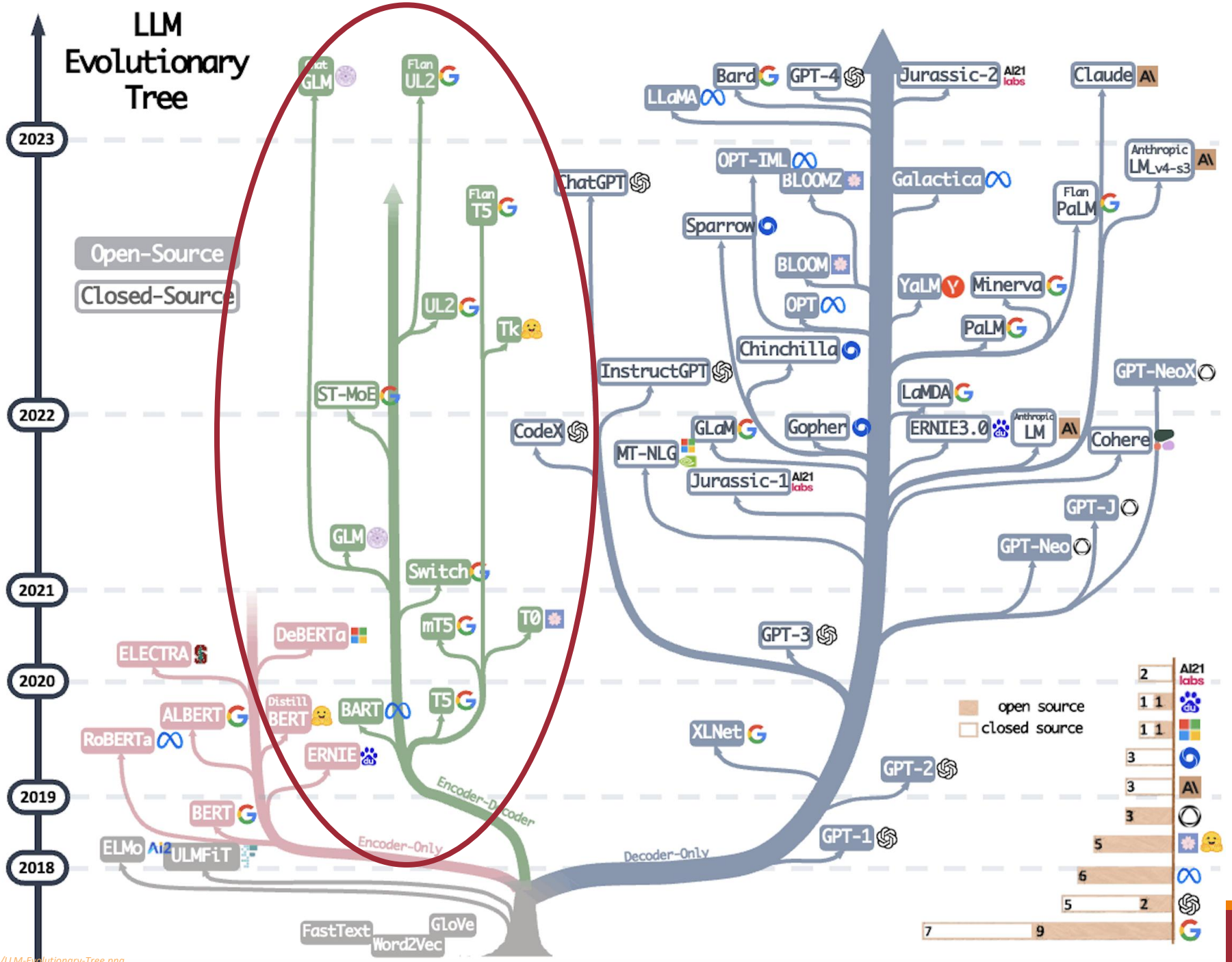
Claude 3 (Anthropic)

Gemma (Google)

OLMo 2 (AI2)

Encoder-Decoder Models





https://blog.biocomm.ai/wp-content/uploads/2023/05/LLM-Evolutionary-Tree.png

Enc-Dec Family of Models

- Encoder-decoder
  - Input: Text sequence with random word spans deleted
  - Goal: Generate the deleted word spans
- How to use:
  - Finetune smaller ones for either generation or classification tasks.
  - Prompt tuning (train a sequence of embedding which get prefixed to the input)

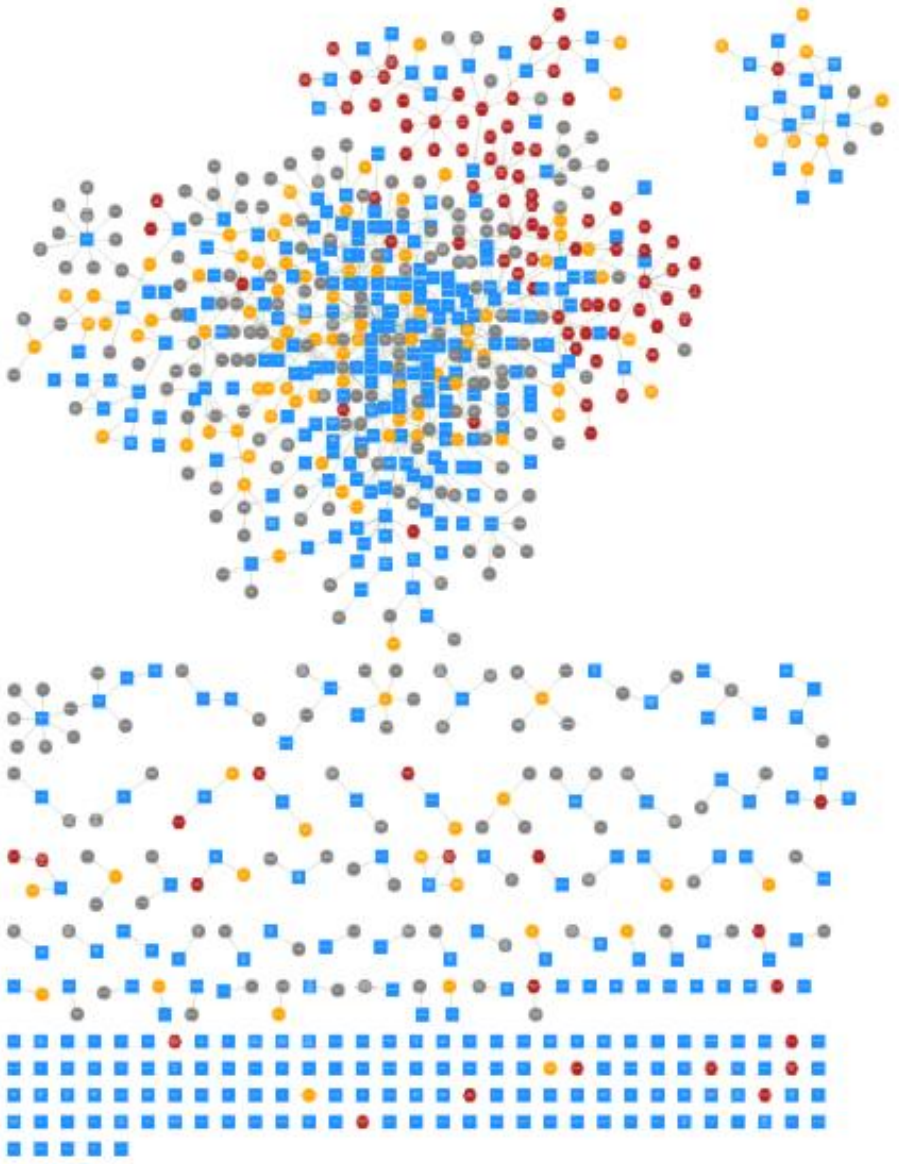
Some Enc-Dec family members

- T5 (Google)
- BART (combo of GPT and BERT) – (Facebook)

- DALL-E 2 (for caption prediction)

All the models!

https://crfm.stanford.edu/ecosystem-graphs/index.html?mode=home



For next lecture!!

Please be prepared to ask questions for the review.

Go through the list of topics on Blackboard and come up with ~3 questions about things you want clarification on.