DSCI 565: ATTENTION MECHANISMS AND TRANSFORMERS

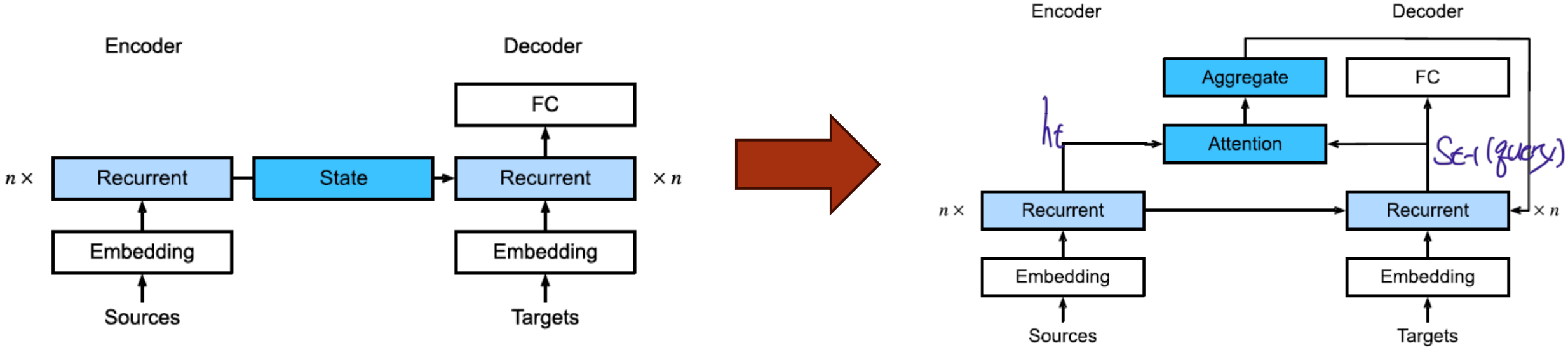
Ke-Thia Yao

Lecture 15: 2025 October 15

Bahdanau Attention Mechanism

- Encoder context is no longer static (e.g., last state of the encoder)
- Context is dependent on the state $\mathbf{s}_{t'}$ of the decoder
- Context is a weighted average of the encoder hidden states

$$\mathbf{c}_{t'} = \sum_{t=1}^{T} \alpha(\mathbf{s}_{t'-1}, \mathbf{h}_t)\mathbf{h}_t$$

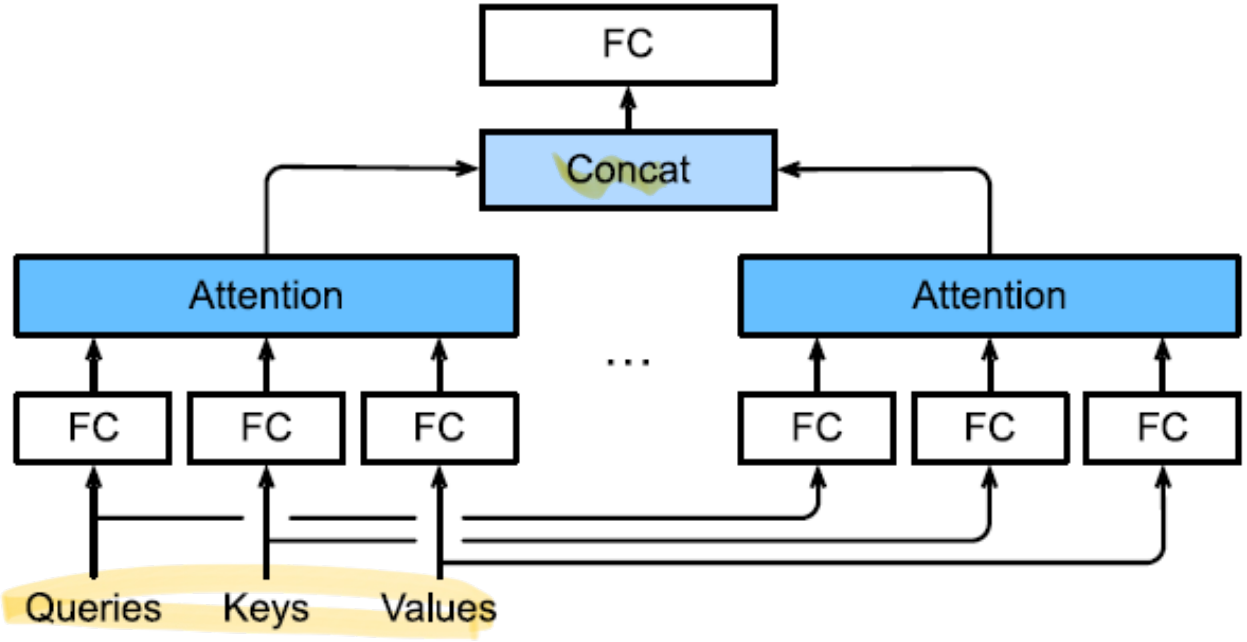


Notebook

- chapter_attention-mechanisms-and-transformers/bahdanau-attention.ipynb

Multi-Head Attention

- In stead of just one attention, we way want multiple attentions
- With each attention focusing on various behavior aspects, e.g.
  - Shorter range vs. longer range
  - Parts of speech vs word meaning
- Use fully-connected layer to learn which aspects to focus using projections of queries, keys and values



Multi-Head Attention

- Multi-head attention transform the queries, keys and values using learned linear projections
- *These projections are feed into h attention poolings in parallel*
- *Then the h outputs $(\boldsymbol{h}_i)$ are* concatenated
- These outputs are called *heads*
- *The concatenate outputs are feed into a fully-connected layer*

- Given a query $\mathbf{q}$, key $\mathbf{k}$ and value $\mathbf{v}$, compute each head:

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v},$$

- Combine

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}.$$

- To avoid extra computational cost, typically $p_v = \frac{p_o}{h}$

Implementation


```python
class MultiHeadAttention(d2l.Module):  #@save
    """Multi-head attention."""
    def __init__(self, num_hiddens, num_heads, dropout, bias=False, **kwargs):
        super().__init__()
        self.num_heads = num_heads
        self.attention = d2l.DotProductAttention(dropout)
        self.W_q = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_k = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_v = nn.LazyLinear(num_hiddens, bias=bias)
        self.W_o = nn.LazyLinear(num_hiddens, bias=bias)
```

On linear project for all heads

```python
def forward(self, queries, keys, values, valid_lens):
    # Shape of queries, keys, or values:
    # (batch_size, no. of queries or key-value pairs, num_hiddens)
    # Shape of valid_lens: (batch_size,) or (batch_size, no. of queries)
    # After transposing, shape of output queries, keys, or values:
    # (batch_size * num_heads, no. of queries or key-value pairs,
    # num_hiddens / num_heads)
    queries = self.transpose_qkv(self.W_q(queries))
    keys = self.transpose_qkv(self.W_k(keys))
    values = self.transpose_qkv(self.W_v(values))

    if valid_lens is not None:
        # On axis 0, copy the first item (scalar or vector) for num_heads
        # times, then copy the next item, and so on
        valid_lens = torch.repeat_interleave(
            valid_lens, repeats=self.num_heads, dim=0)

    # Shape of output: (batch_size * num_heads, no. of queries,
    # num_hiddens / num_heads)
    output = self.attention(queries, keys, values, valid_lens)
    # Shape of output_concat: (batch_size, no. of queries, num_hiddens)
    output_concat = self.transpose_output(output)
    return self.W_o(output_concat)
```

Shape: b, q, d

Map and transpose query.
Shape: $b, q, d \rightarrow b * h, q, d/h$

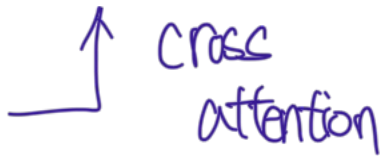
Similar transformation for keys and values

Compute in parallel:
all batches and all heads

Reverse transpose_qkv

Notebook

- chapter_attention-mechanisms-and-transformers/multihead-attention.ipynb



Self-Attention

- Given sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$, self-attention outputs a sequence $\mathbf{y}_1, \ldots, \mathbf{y}_n$

$$\mathbf{y}_i = \text{Attention}(\mathbf{x}_i, \mathcal{D})$$

where $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)\}$

- The queries, keys and values are all $\mathbf{x}_i$
- The idea is to find other parts of the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to which to pay attention

Self-Attention

For self-attention use the same matrix for all three inputs $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$

For example, use the Input sentence embedding matrix for all three inputs

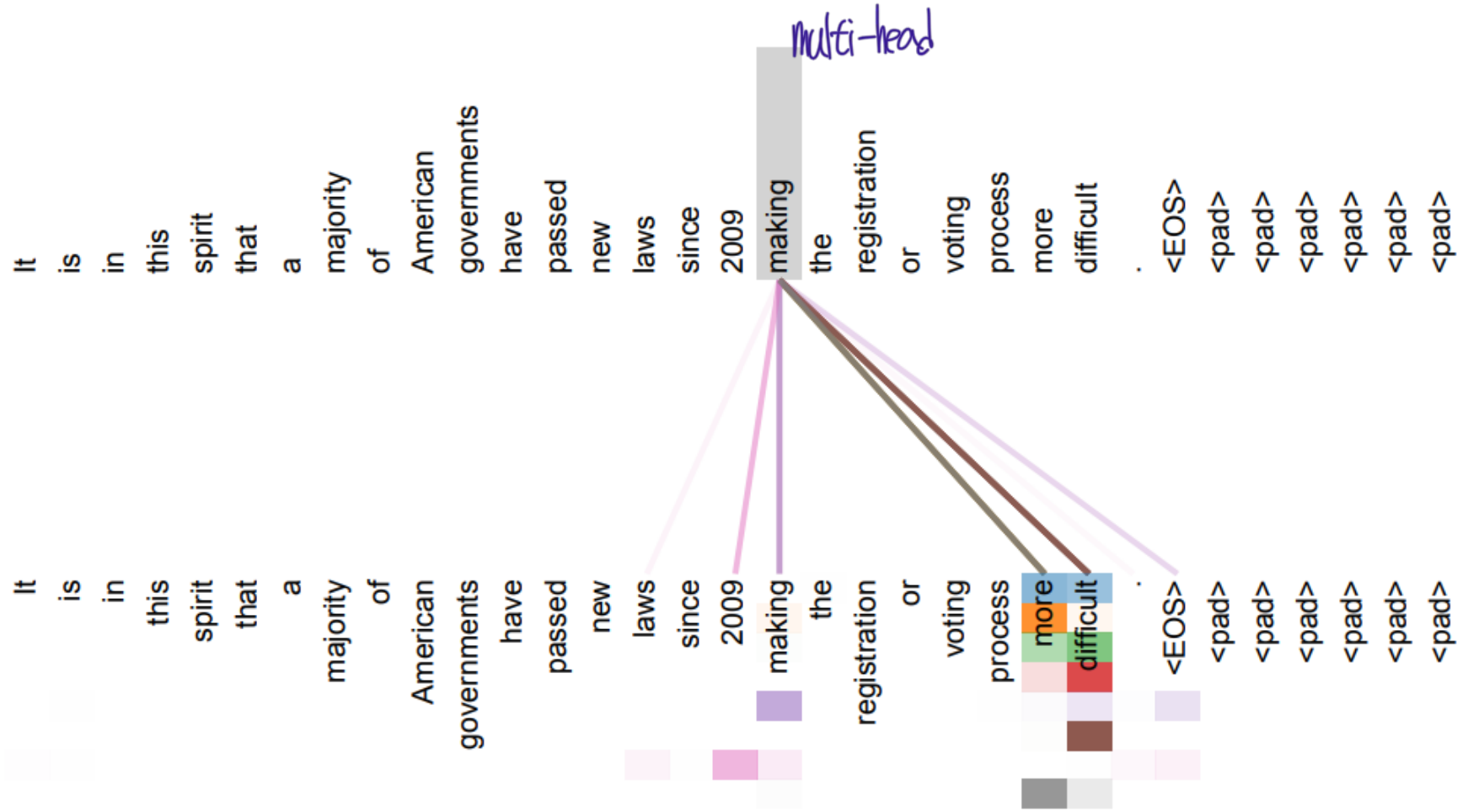
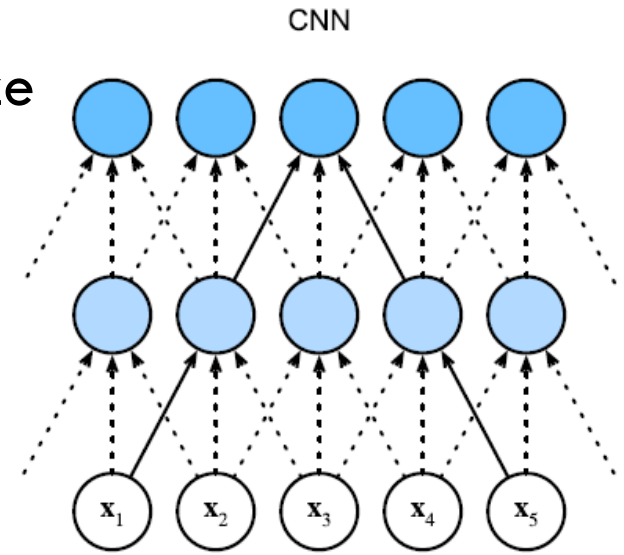


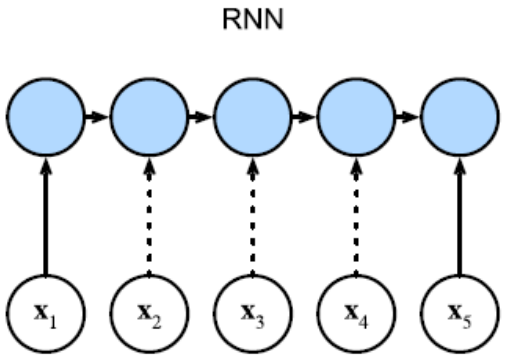
Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

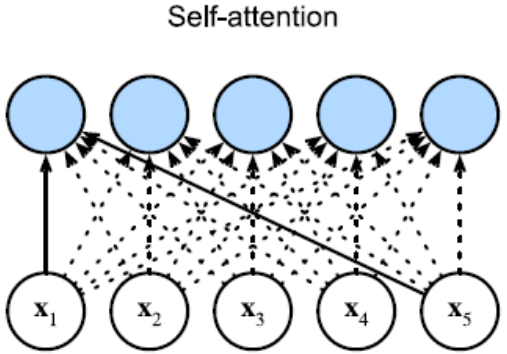
Comparing CNNs, RNNs and Self-Attention

- All can be thought as ways to propagate information along a sequence
- Let n be the sequence length
  d be the input/output channel, hidden size
  k be the kernel size

	CNN	RNN	Self-Attention
Complexity (per layer)	$\mathcal{O}(knd^2)$	$\mathcal{O}(nd^2)$	$\mathcal{O}(n^2d)$
Sequential Ops	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Max path length	$\mathcal{O}(n/k)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$







Positional Encoding

- Attention does not preserve order information
- The vector $\mathbf{y}_i$ is just a weighted average of other vectors
- What if order information is needed for the translation task?
- Could just concatenate an index (in binary) to each word embedding
- Positional encoding does something similar, except
  - It uses sine and cosine functions
  - Adds index to embedding, instead of concatenation

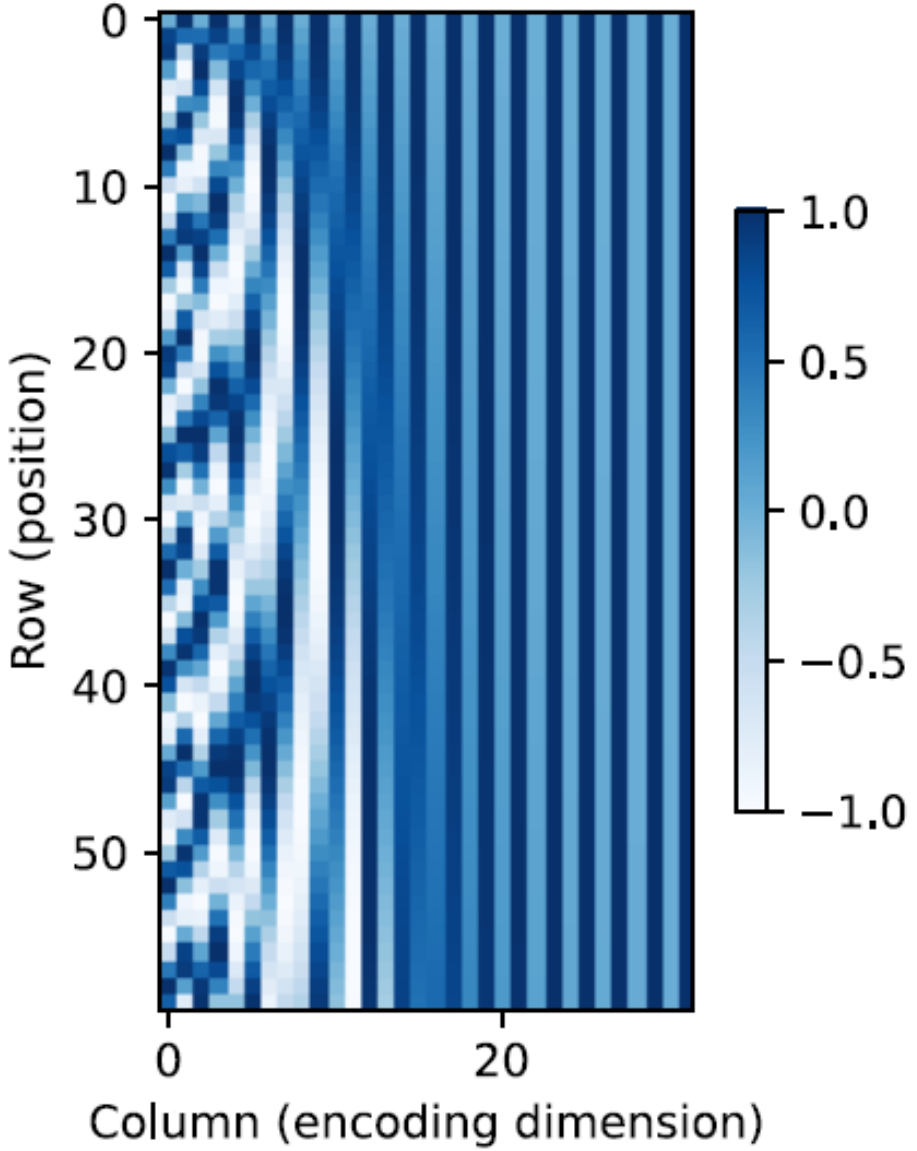
```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1101
1110
1111
```

Positional Encoding

- Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the input sequence and $\mathbf{P} \in \mathbb{R}^{n \times d}$ be in position encoding, the positional encoded output is $\mathbf{X} + \mathbf{P}$
- Where $i \in [0, n)$ and $j \in \left[0, \frac{d}{2}\right]$

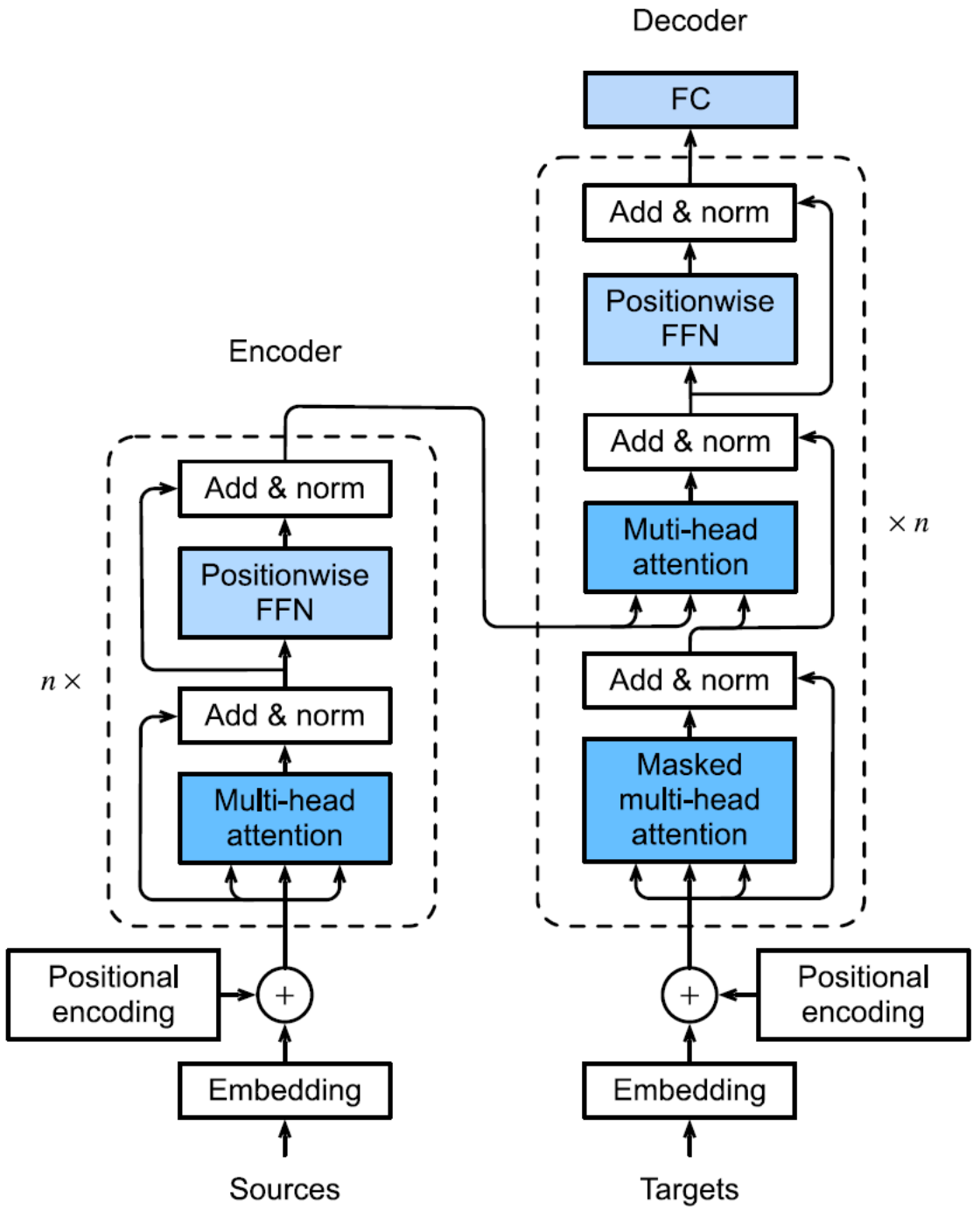
$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$$



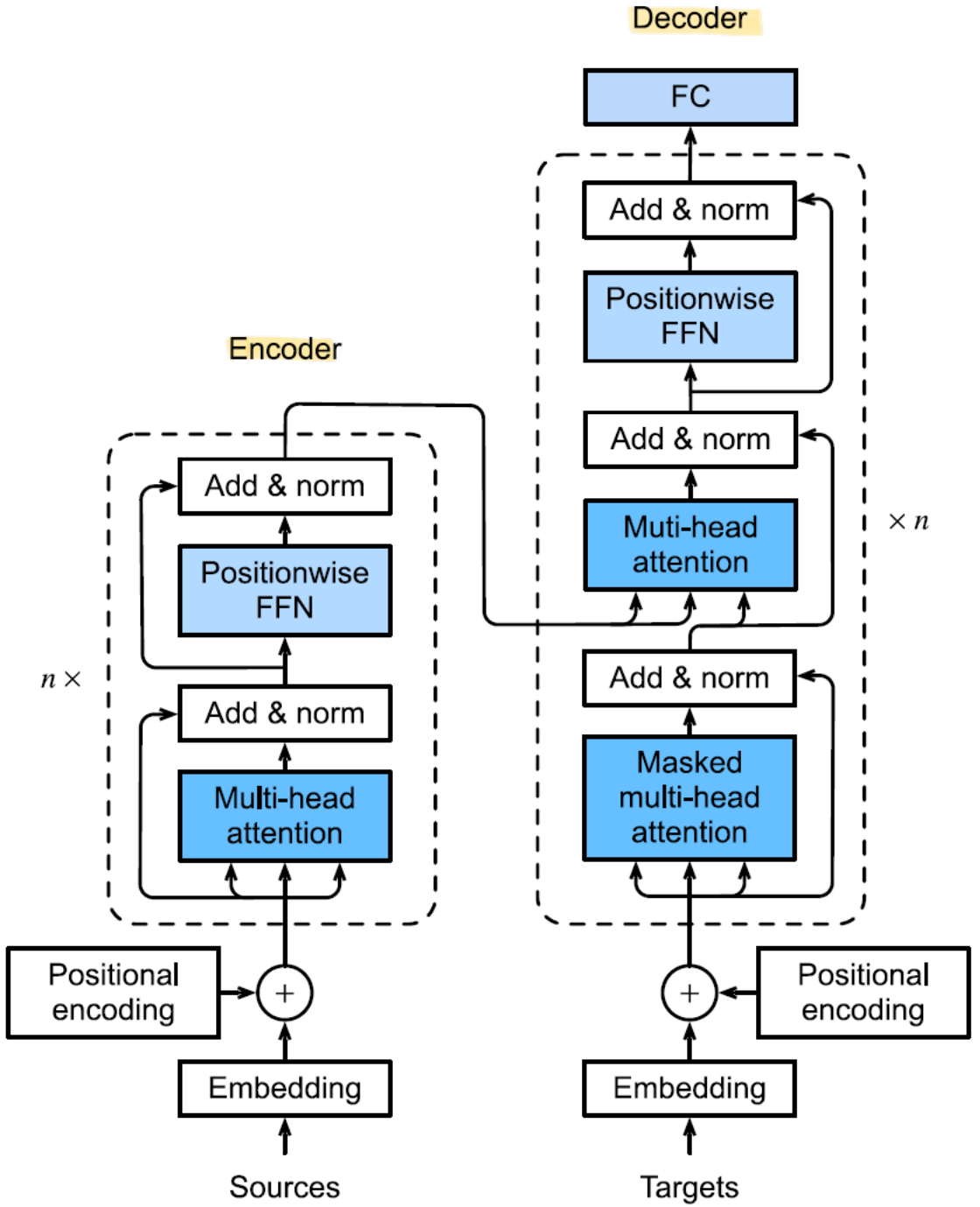
Transformer Architecture

- "Attention is All You Need," Vaswani *et al.*, 2017
- Regular neural network with attention mechanism
- No recurrent units
  - Easier to train
  - No vanishing gradients
  - Can be parallelized
- Pervasive in deep learning applications: language, vision, speech and reinforcement learning

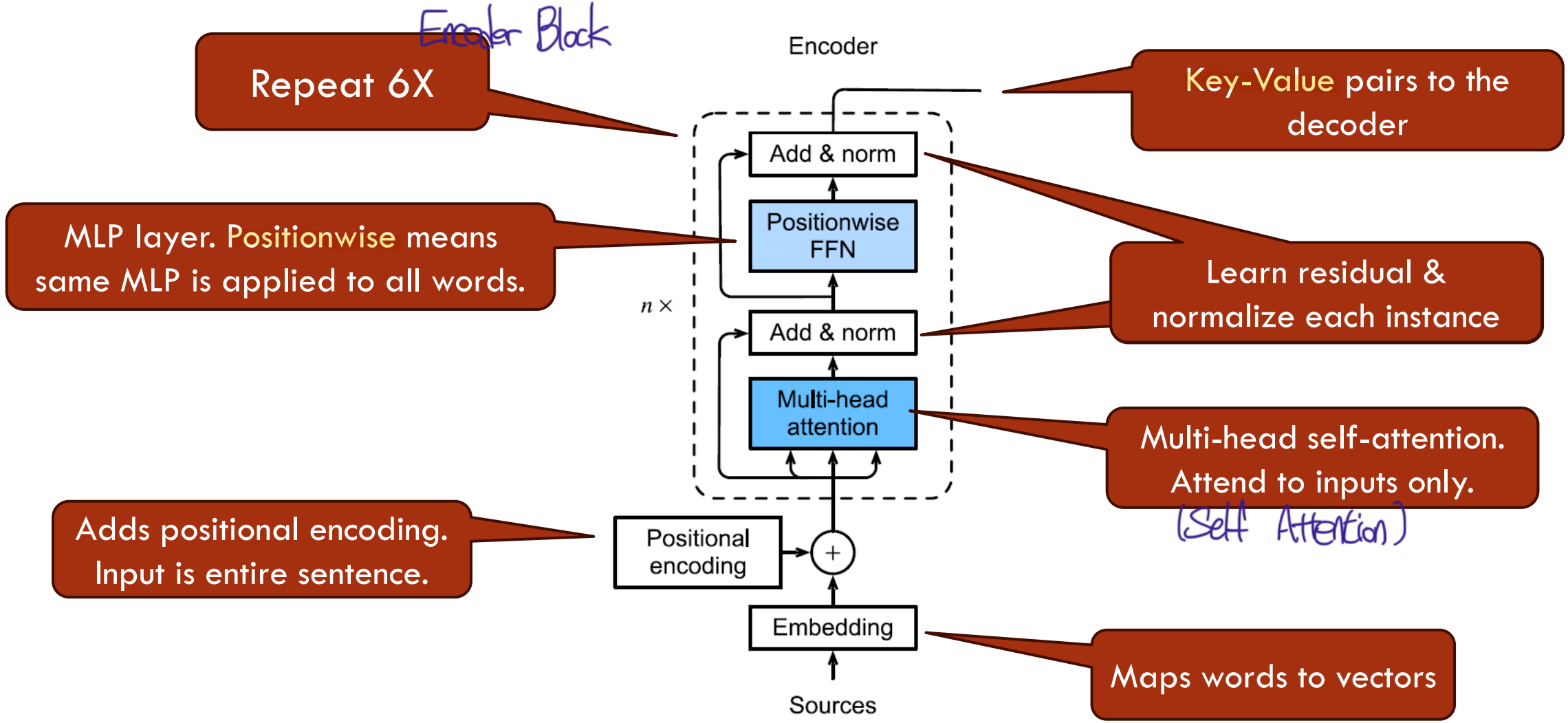


Training and Prediction

- Training phase
  - Send entire English sentence to Encoder, and Spanish sentence to decoder
  - "I like soccer" and "<sos> Me gusta el futbol"
- During translation phase Inference phase
  - Call transformer multiple times
  - "I like soccer" and "<sos>"
  - "I like soccer" and "<sos> Me"
  - "I like soccer" and "<sos> Me gusta"
  - "I like soccer" and "<sos> Me gusta el"



Encoder


Encoder Block
Repeat 6X
Encoder
Key-Value pairs to the decoder
Add & norm
Positionwise FFN
MLP layer. Positionwise means same MLP is applied to all words.
Learn residual & normalize each instance
$n \times$
Add & norm
Multi-head attention
Multi-head self-attention. Attend to inputs only.
(Self Attention)
Adds positional encoding. Input is entire sentence.
Positional encoding
+
Embedding
Maps words to vectors
Sources

Decoder

Decoder Block

Repeat 6X

MLP layers. Positionwise means same MLP is applied to all words.

Learn residual & normalize each instance

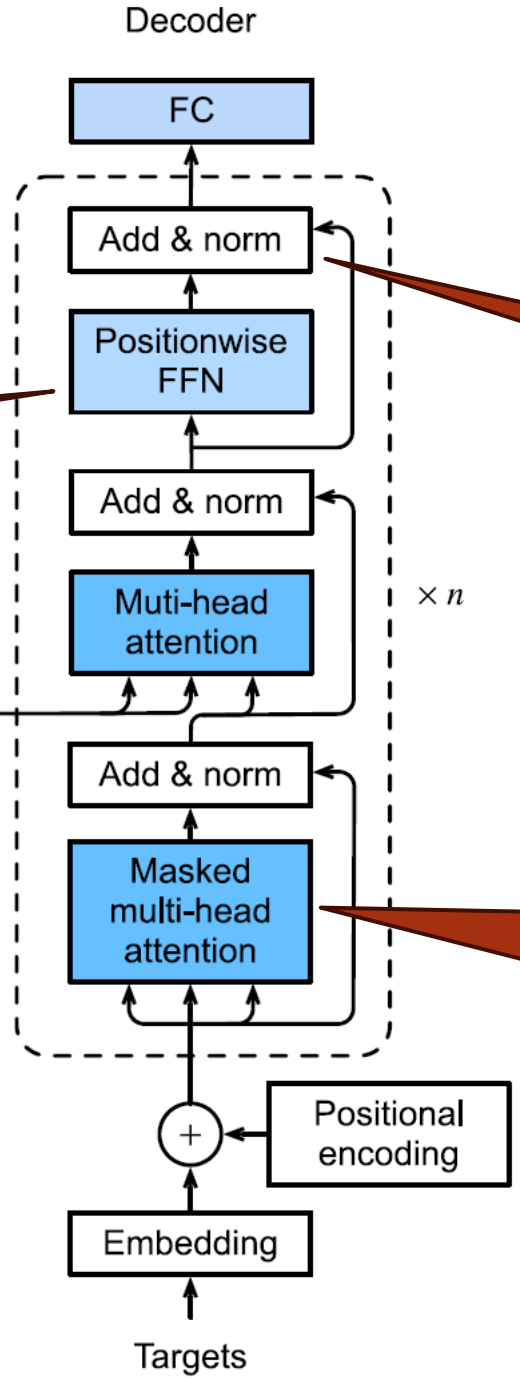
key, value from encoder

Cross attention. Use decoder states to attend to encoder states.

(Cross attention)

Masked self-attention. Attend to outputs only. During training mask words forward in time t'

(self attention)

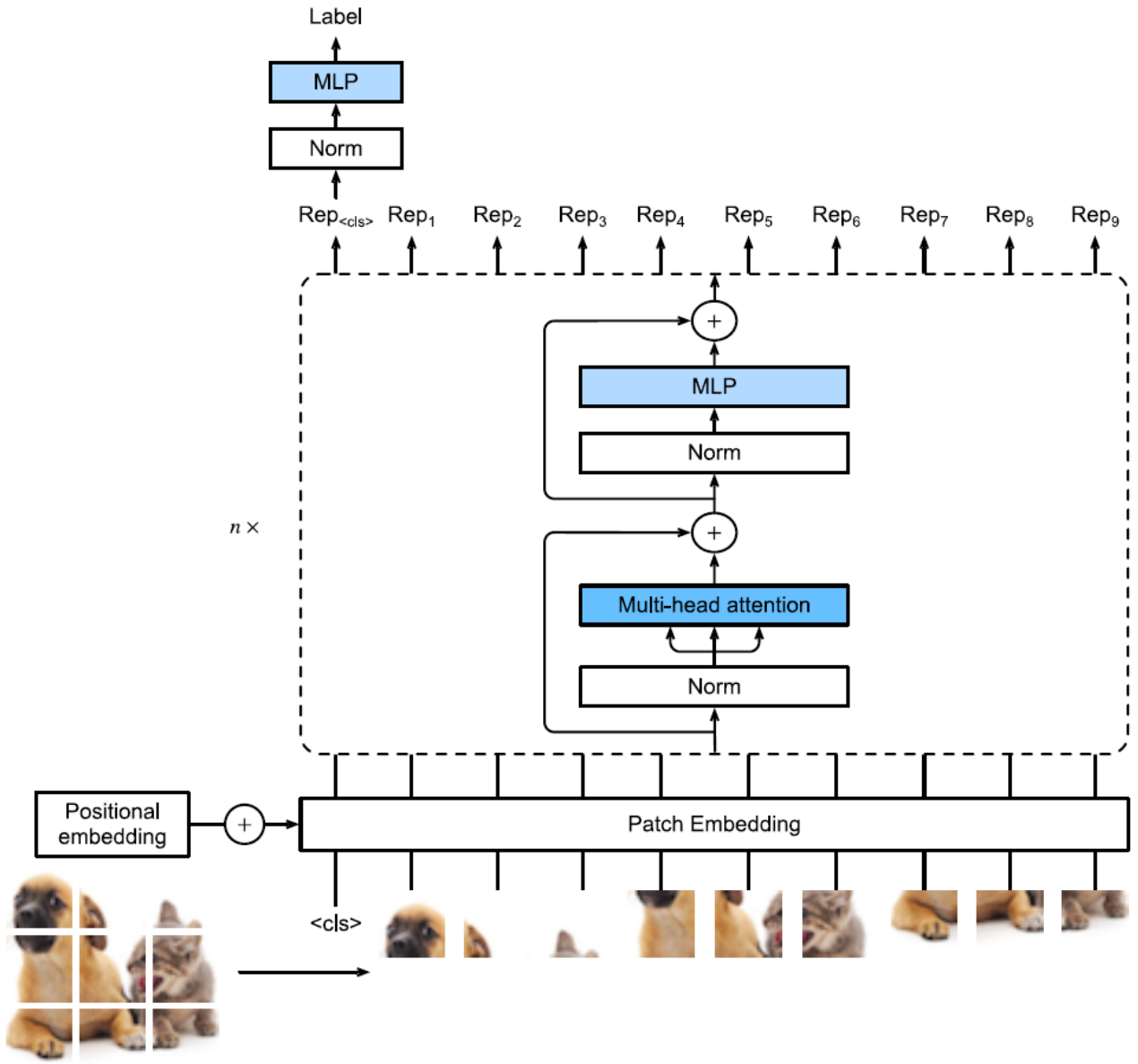


Notebook

- chapter_attention-mechanisms-and-transformers/transformer.ipynb

Vision Transformers

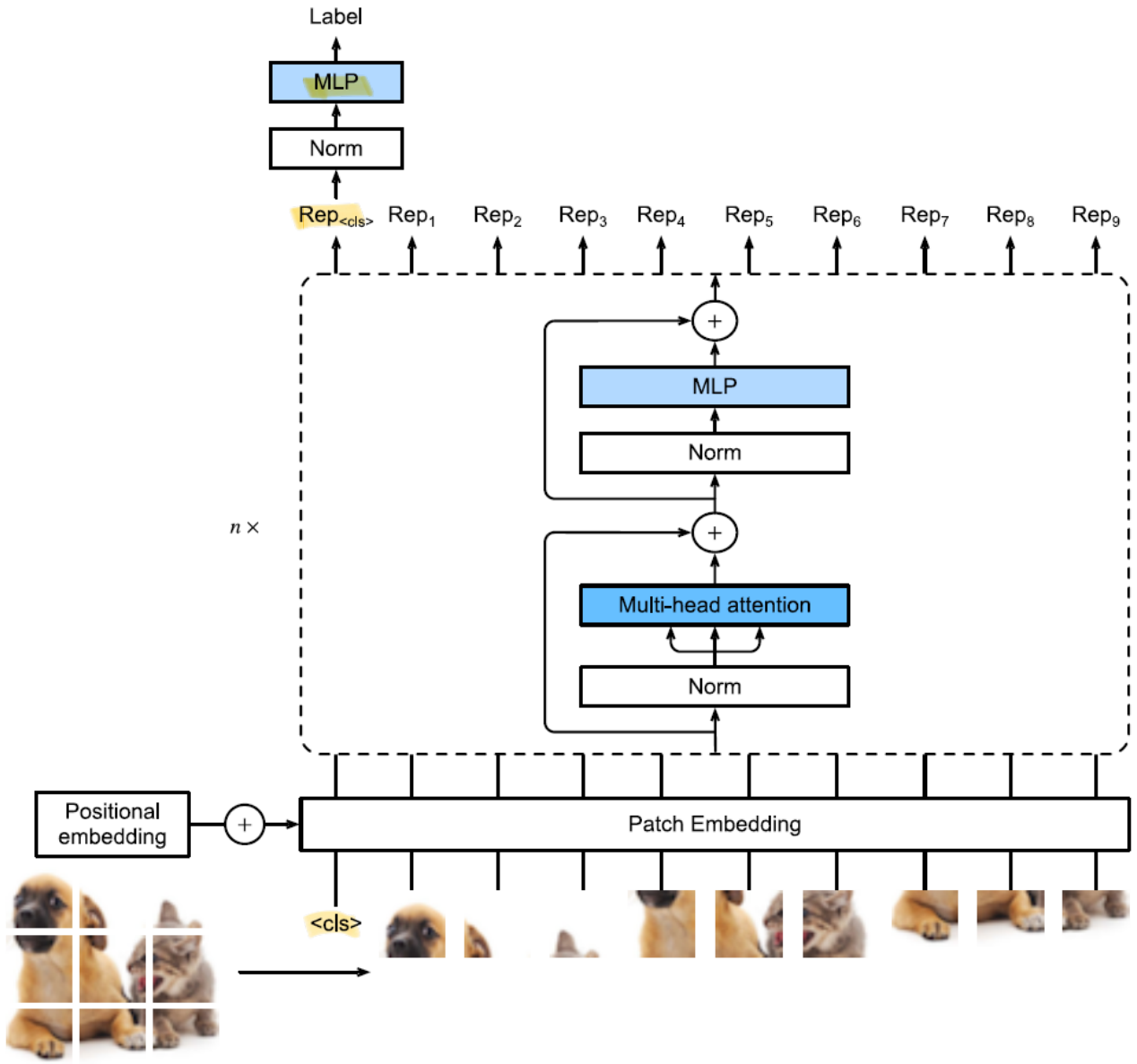
- Can the transformer architecture be applied to computer vision?
- Vision Transformers (ViTs), Dosovitskiy *et al.*, 2021
  - Better scalability than CNN
  - Performs better than ResNets on larger datasets



Vision Transformers

only encoder

- Partition image into non-overlapping patches
  - Conv with kernel size = stride
- Embed patches as vectors
- Add horizontal and vertical positional embedding
- Self-attention across all patches
- Layer normalization with learning residuals
- <cls> token with corresponding $\text{Rep}_{<cls>}$ for classification



Improving Vision Transformers

- Improving ViT on smaller datasets
  - DeiT (Touvron *et al.*, 2021) more data efficient training strategies
  - T2T ViT (Yuan *et al.*, 2021) use token-to-token mappings
- Scaling ViT to larger images
  - Swin Transformers (Liu *et al.*, 2021) hierarchical transformers with shifted windows to avoid global self-attention

Notebook

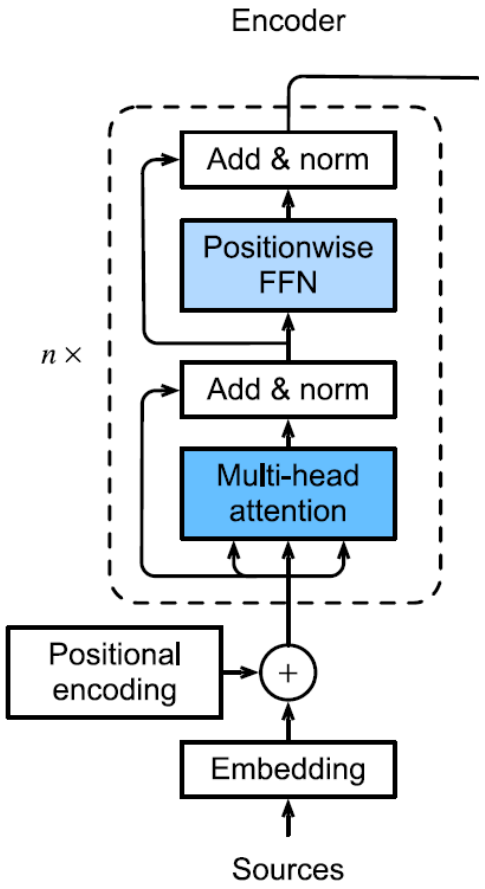
- chapter_attention-mechanisms-and-transformers/vision-transformer.ipynb

Large-Scale Pretraining with Transformers

- To build a machine learning model for a specific tasks, we could train the model *from scratch* using from a task-specific dataset
- A potential problem of the from-scratch approach is the resulting model may be sensitive to shifts in the data distribution
- Another approach is to first train a *general model* on a large dataset. Then, use *transfer learning* to adapted the pretrained model to the specific task
- This second approach is used by researcher with CNN (e.g., EfficientNet, ResNet)
- This is the approach taken by researcher with transformers as well

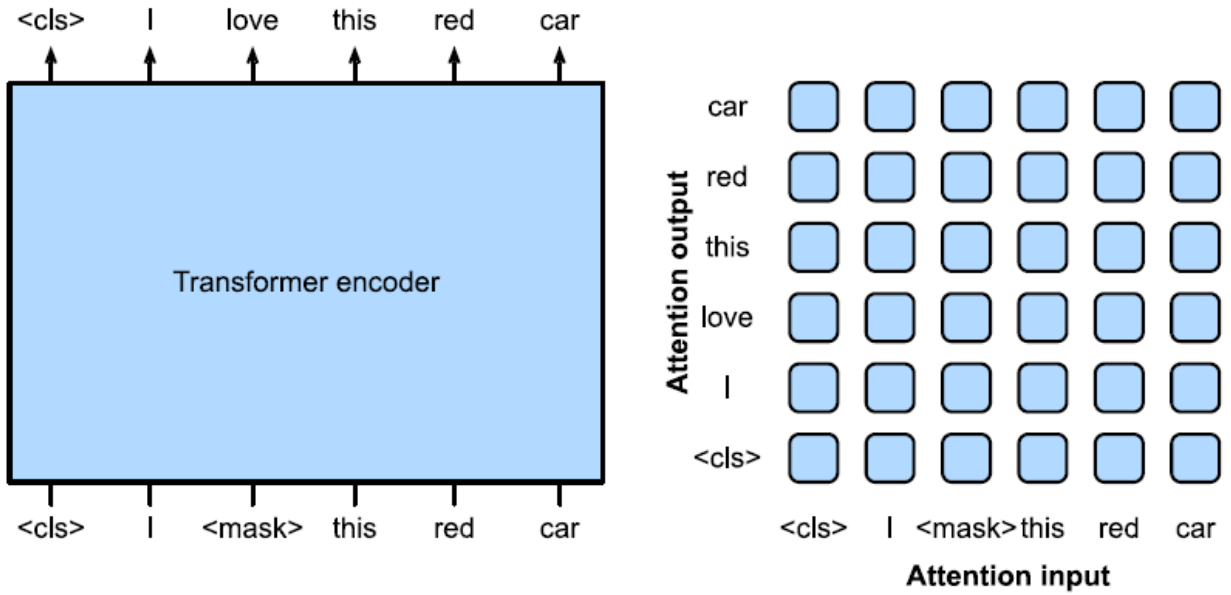
Encoder-Only Pretrained Model

- The Transformer encoder converts a sequence of input tokens into a vector representation
- Then the representation can be used further for learning, such as generated a classifier
- The vision Transformer is an encoder-only architecture with the representation of the special token <cls> used for classification



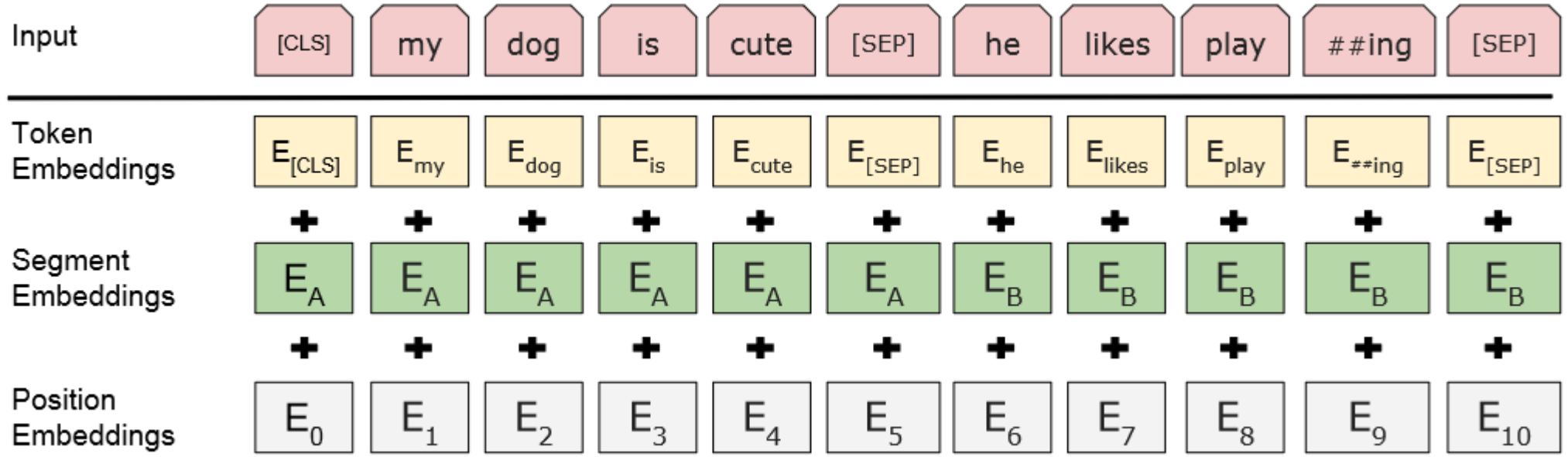
BERT: Bidirectional Encoder Representations from Transformers

- BERT is pre-trained model on text sequences using *masked language modeling*
- BERT has 350M parameters
- BERT is pre-trained on BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Pre-training Task #1: Masked LM. Tokens from the input text sequences are randomly masked
  - Masked input: `<cls> I <mask> this red car`
- During training BERT is given the entire masked input. This allows all tokens to attend to each other (i.e., "bidirectional")



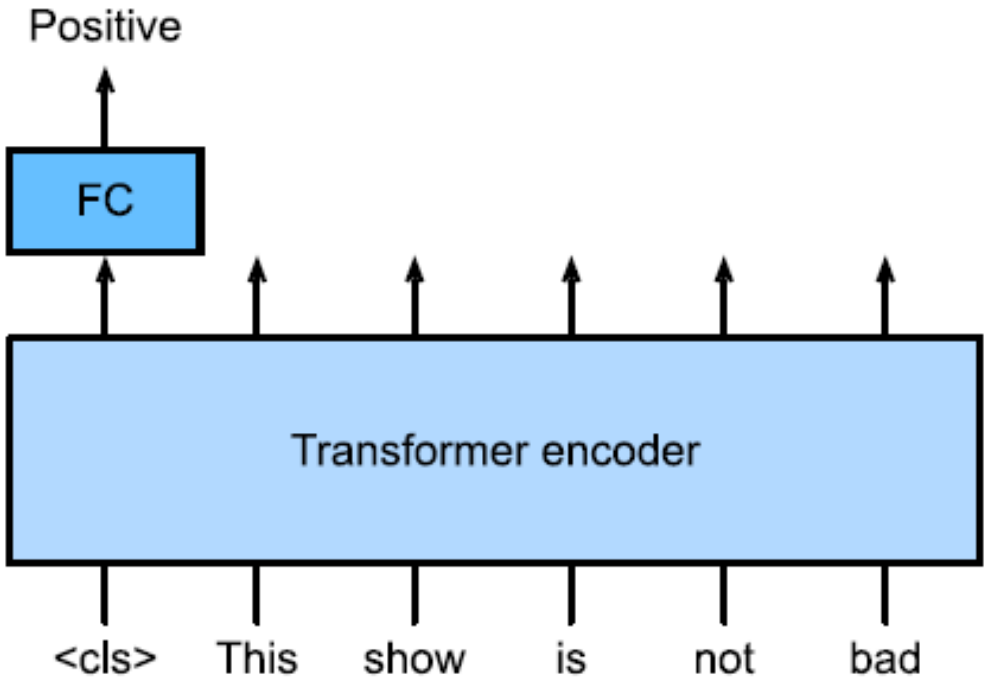
BERT

- Task #2: Next Sentence Prediction (NSP)
- Training set consists of pairs of sentences
  - 50% the second sentence is from the actual next sentence
  - 50% the second sentence is replaced by a randomly chosen sentence



Fine-Tuning BERT

- To fine-tune BERT, add additional layers with randomized parameters
- Then train to update the new parameters as well pretrained BERT parameters
- Right shows fine-tuning BERT for sentiment analysis



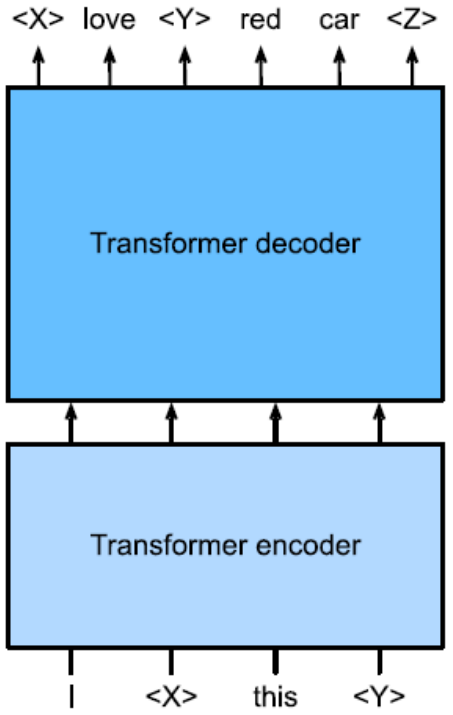
Fine Tuning BERT

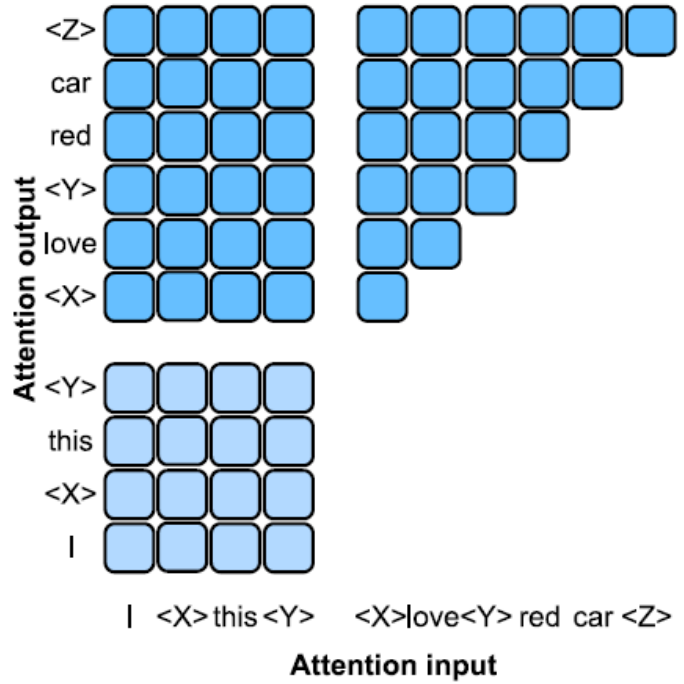
- The BERT paper (Devlin *et al., 2019*) fine tuned BERT on four tasks
  - GLUE benchmark for language understanding modeled as classification problem
  - SQuAD v1.1 question answering problem. Given a question and a passage from Wikipedia containing the answer, the task is to predict the answer text span in the passage.
  - SQuAD v2.0 extends v1.1 with problems with passages that do not contain the answer
  - SWAG common sense reasoning. Given sentence and choose the most plausible among four possible continuation sentences
- Required 2 to 4 epochs to fine tune for a task

mid term

Encoder-Decoder Pretrained Model

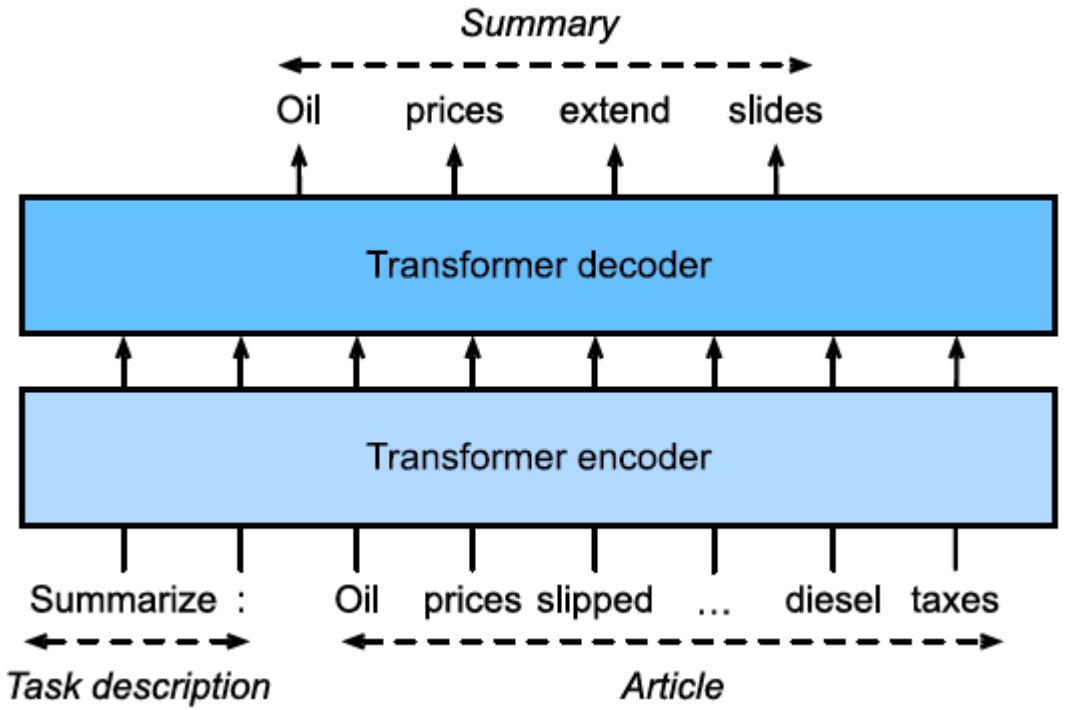
- Examples include BART (Lewis *et al.*, 2019) from Meta, and T5 (Raffel *et al.*, 2020) from Google
- T5 trained to generate missing spans of words
- Given
  - Original sentence: I love this red car
  - Input: I <X> this <Y>
  - Target: <X> love <Y> red card <Z>





Fine Tuning T5

- Fine tuning T5
  - Add task description to input
  - No additional layers are needed
- T5 can generate output of arbitrary length
- After fine tuning T5 (11 billion parameters), it achieved state-of-the-art performance in encoding/classification tasks and generation/summarization tasks



T5 Encoder

Teddy bears swimming at the Olympics 400m Butterfly event.

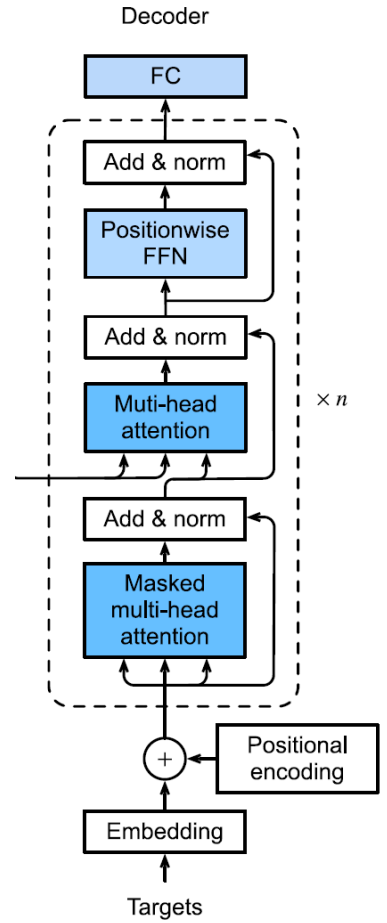
A cute corgi lives in a house made out of sushi.

A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

Text-to-image examples by the Imagen model, whose text encoder is from T5 (figures taken from Saharia et al. (2022)).

Decoder-Only Pretrained Models

- Examples include GPT (100 million parameters), GPT-2 (1.5 billion parameters) and GPT-3 (175 billion parameters)
- GPT-2 was trained on 40GB of text and obtained state-of-the-art results multiple language benchmarks *without updating the parameters or architecture*



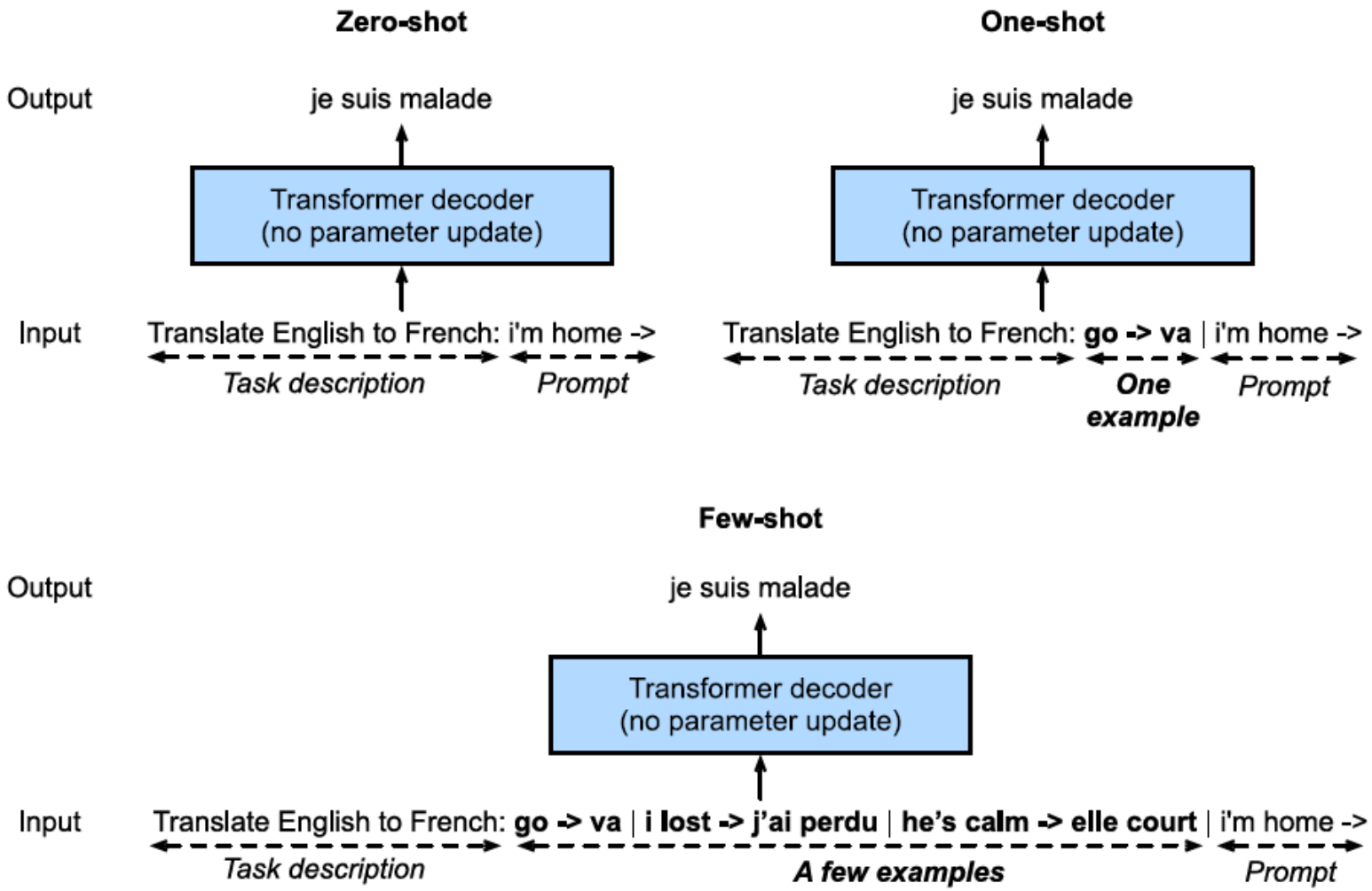
In-Context Learning

Input consist of:

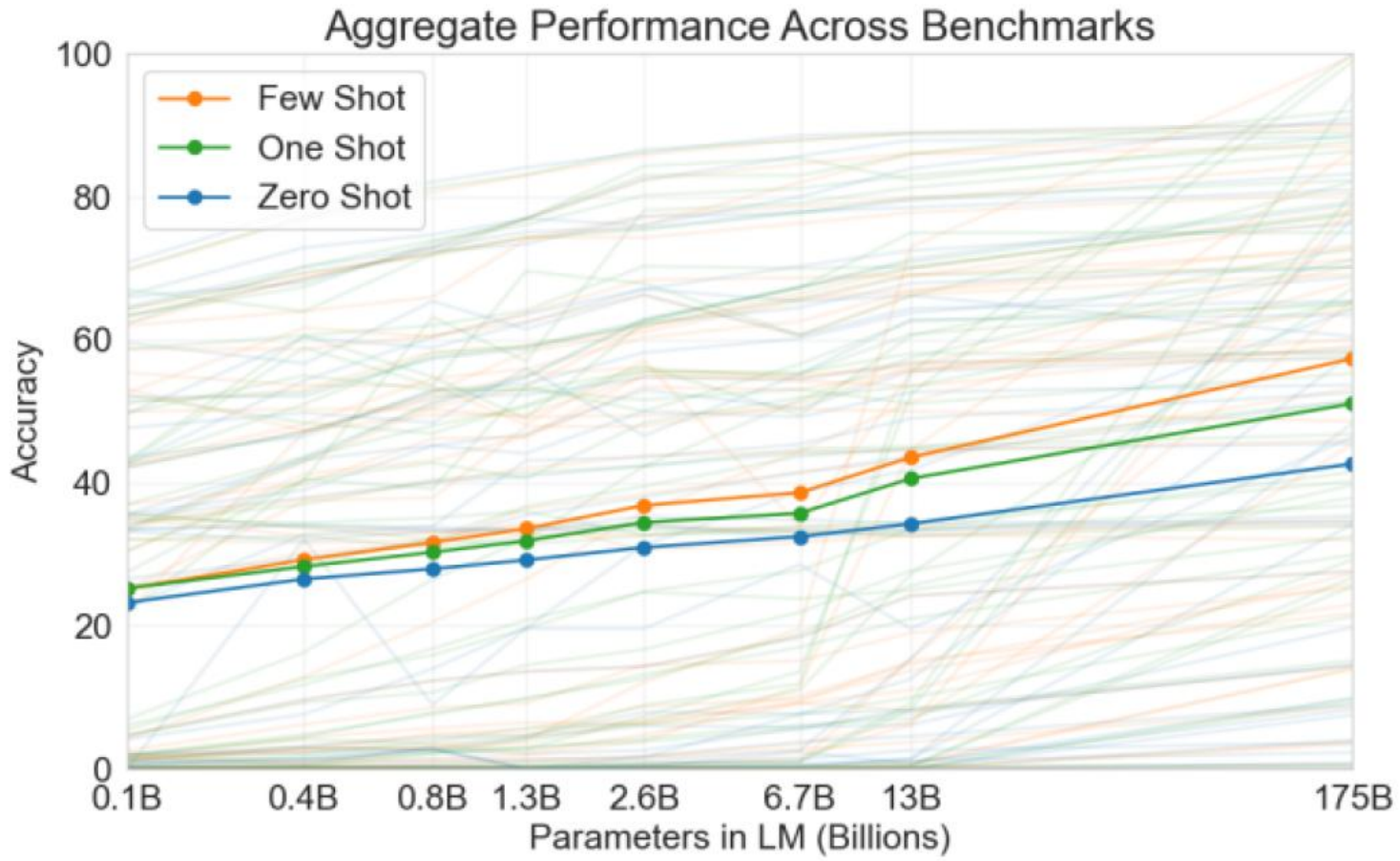
Task description

Zero or more examples

A Prompt



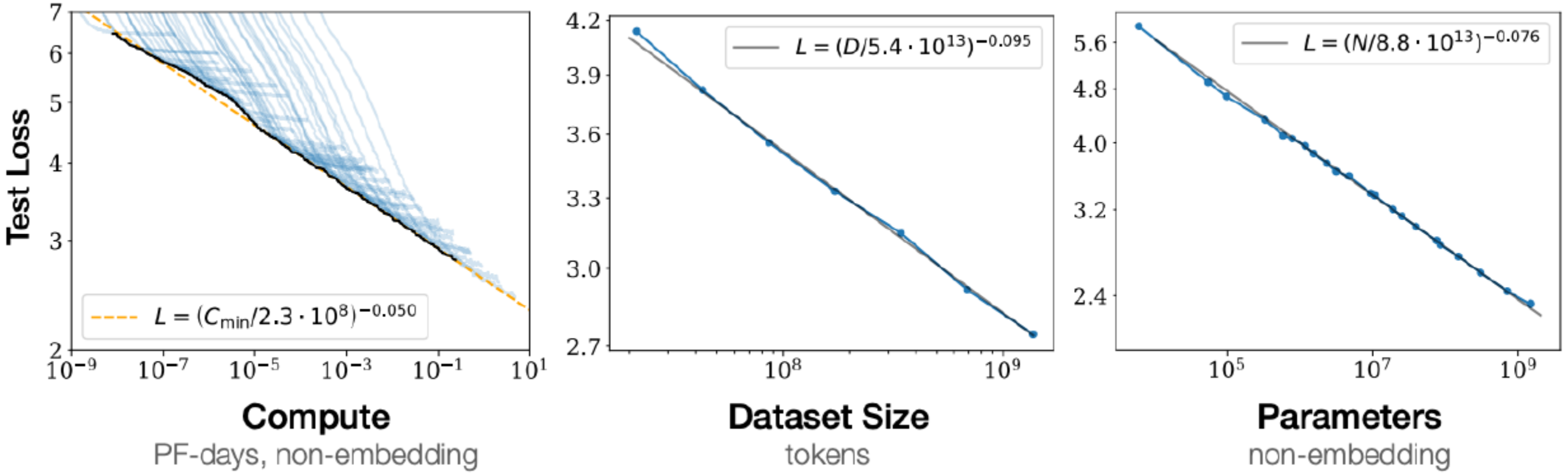
GPT-3 Performance



Aggregate performance of GPT-3 for all 42 accuracy-denominated benchmarks (caption adapted and figure taken from Brown et al. (2020)).

Scalability

- Power-law scaling of Transformers in GPT language model (Kaplan *et al.*, 2020)
- Test loss decreases smoothly with compute, data size and parameters



Scalability

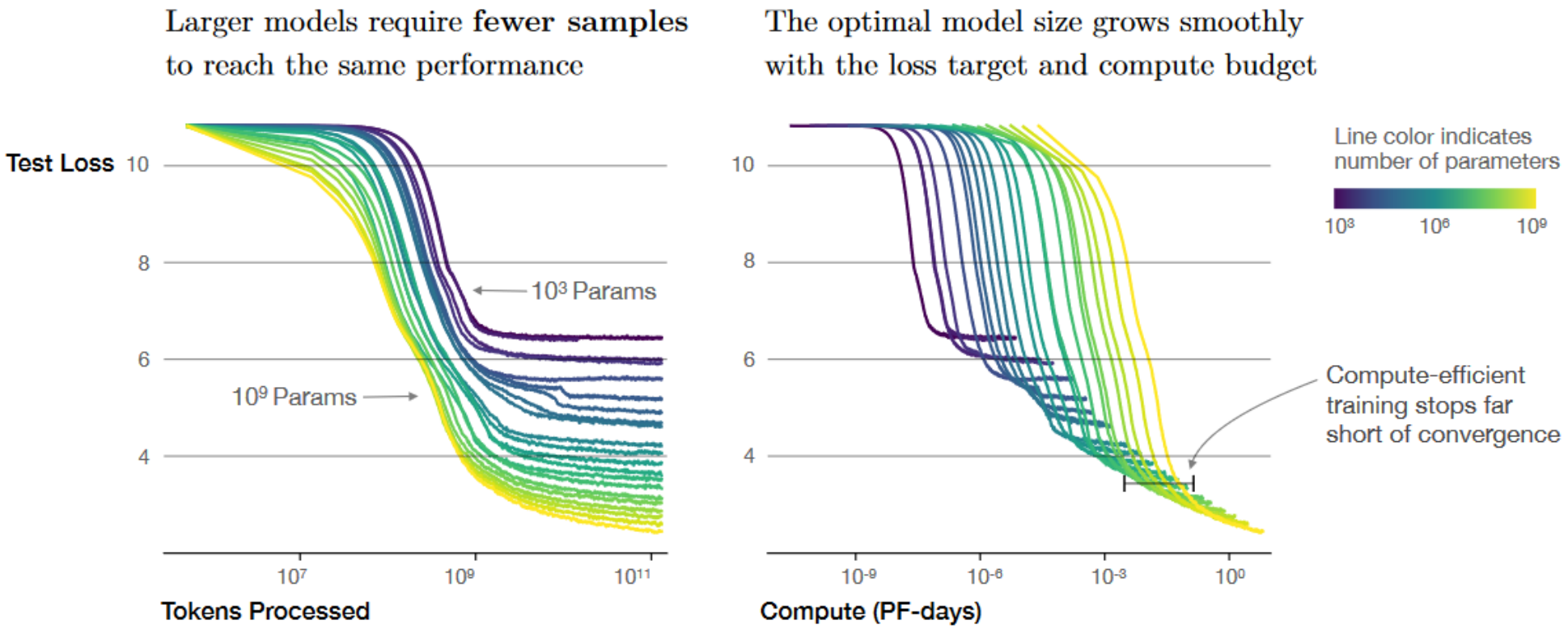


Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

Scalability

- Power law scaling persist for GTP-3 sized networks

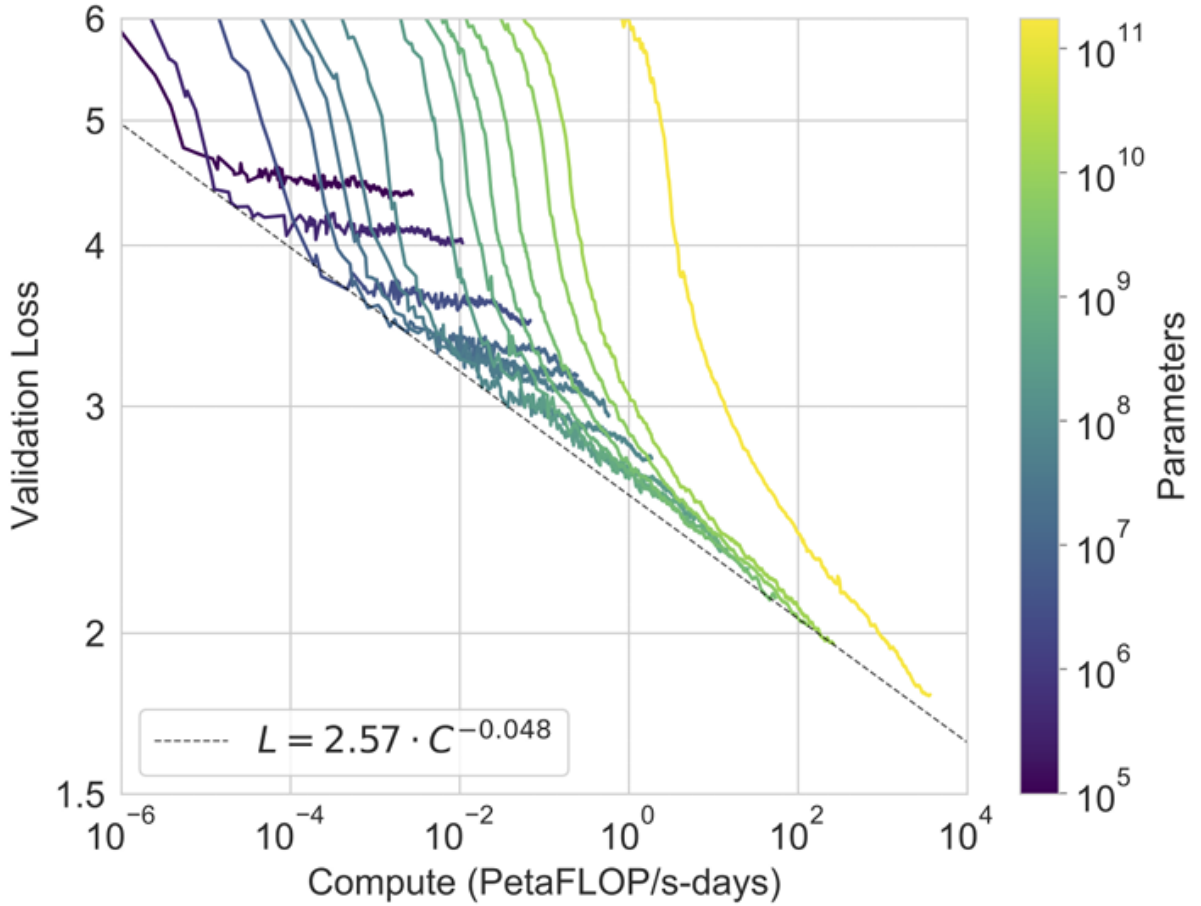


Figure 3.1: Smooth scaling of performance with compute. Performance (measured in terms of cross-entropy validation loss) follows a power-law trend with the amount of compute used for training. The power-law behavior observed in [KMH+20] continues for an additional two orders of magnitude with only small deviations from the predicted curve. For this figure, we exclude embedding parameters from compute and parameter counts.

Large Language Models

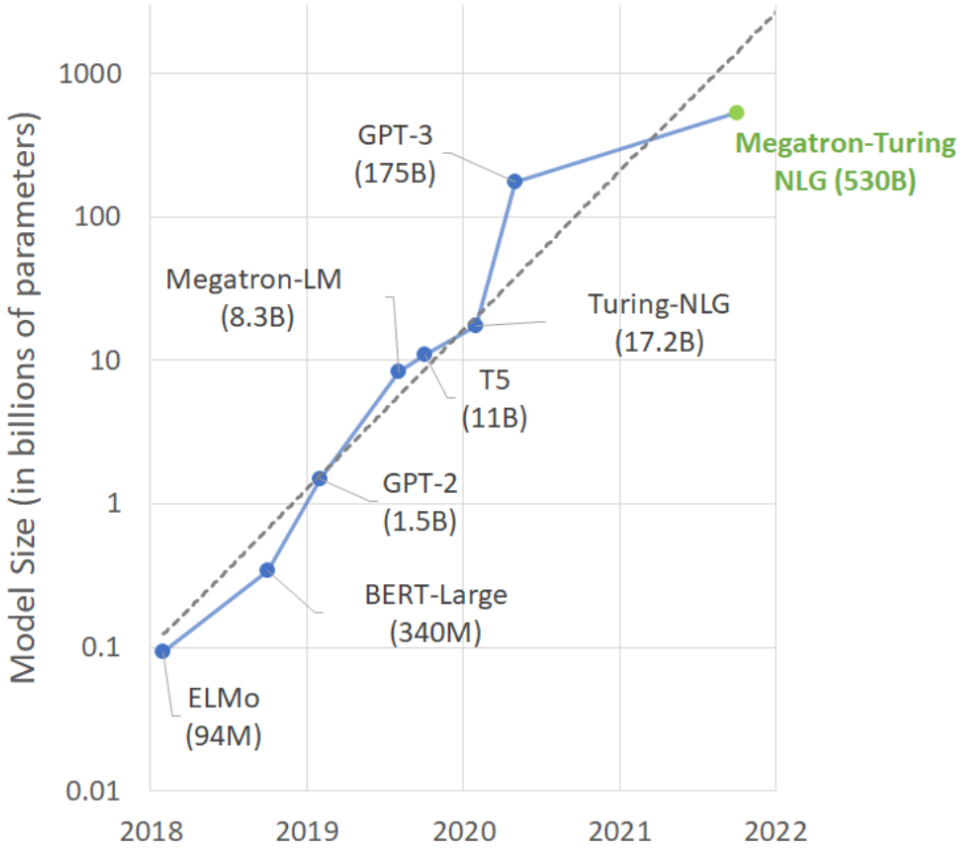
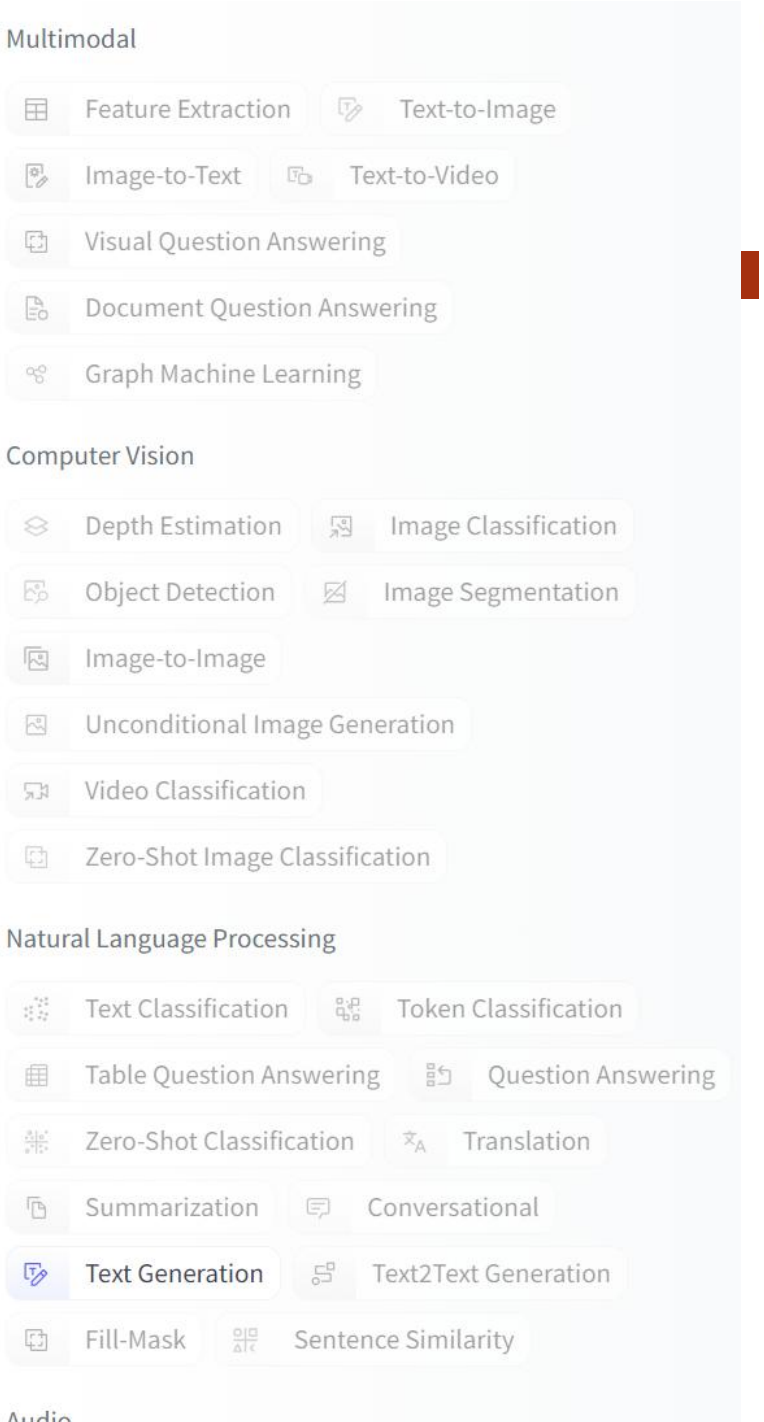
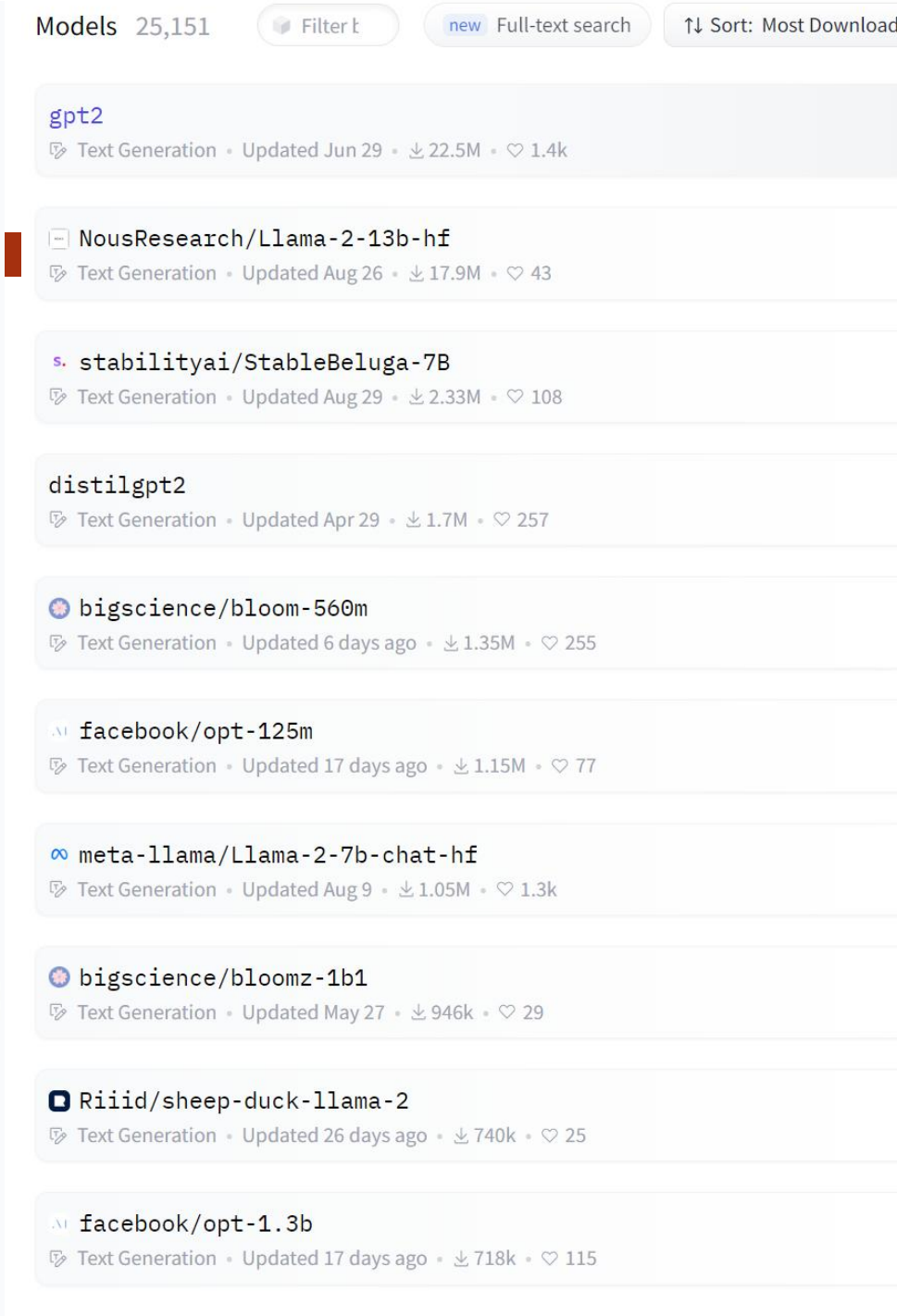


Figure 1. Trend of sizes of state-of-the-art NLP models with time

- The trend in training ever large models continues
- Gopher has 270B parameters, trained on 300B tokens (Rae *et al, 2021)*
- Megatron Turing NLG has 530B parameters, trained on 270B tokens (Smith *et al.*, 2022)
- Chinchilla has 70B parameters trained on 1.4T tokens (4X) outperformed Gopher (Hoffmann *et al., 2022)*
- PaLM has 540B parameters trained on 780B tokens (Chowdhery *et al.*, 2022)

Hugging Face Open-Sourced Models





TinyStories: How Small Can Language Models Be and Still Speak Coherent English?

- Use GPT4 generated children's stories to train small networks, Eldan and Li, 2023

Prompt	2.5M, 8 layers	33M, 4 layers	21M, 1 layer	GPT2-XL (1.5B)
Alice was so tired when she got back home so she went	to bed.	straight to bed.	to bed with a big smile on her face.	outside
Lily likes cats and dogs. She asked her mom for a dog and her mom said no, so instead she asked	her mom if she could have a dog.	her dad for a cat.	her mom again.	her dad. They brought her a Chihuahua, and she took to them immedi-ately.
Alice and Jack walked up the street and met a girl in a red dress. The girl said to them, "Hi, I'm Jane. What are your names?"	Alice smiled and said, "My name is Daisy. What's your name?"	Alice said, "I'm Alice and this is Jack."	Jack smiled and said, "I'm Jane. Nice to meet you!"	Jane said, "Jack and Kate"