# Deep Learning Practice - NLP

## Language Modelling, GPT, Pretraining using HF Transformers

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

# So far

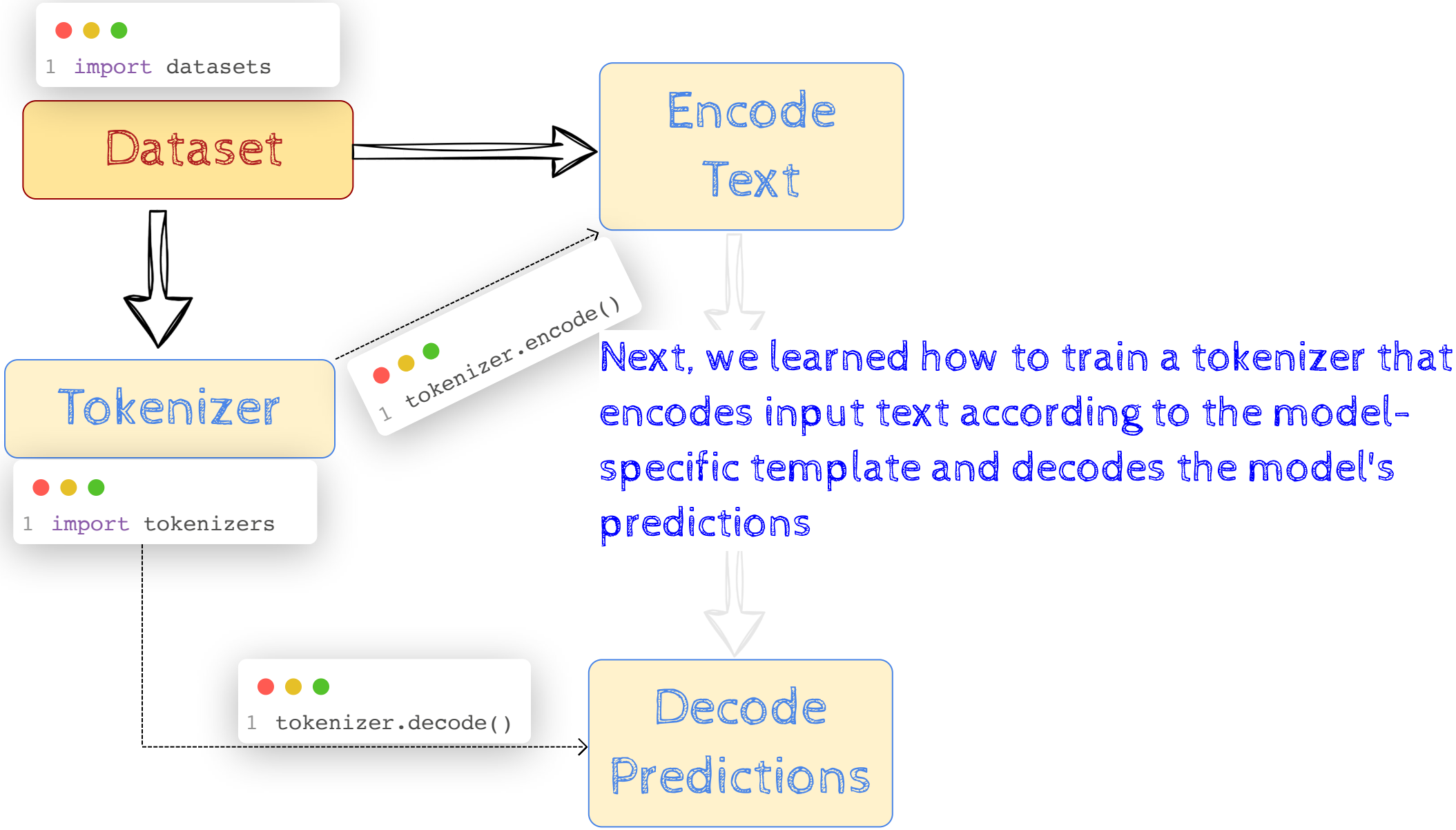
```
1 import datasets
```

Dataset

We learned how to load any dataset from the HF hub, access and modify samples, and save them to disk using the HF datasets module.

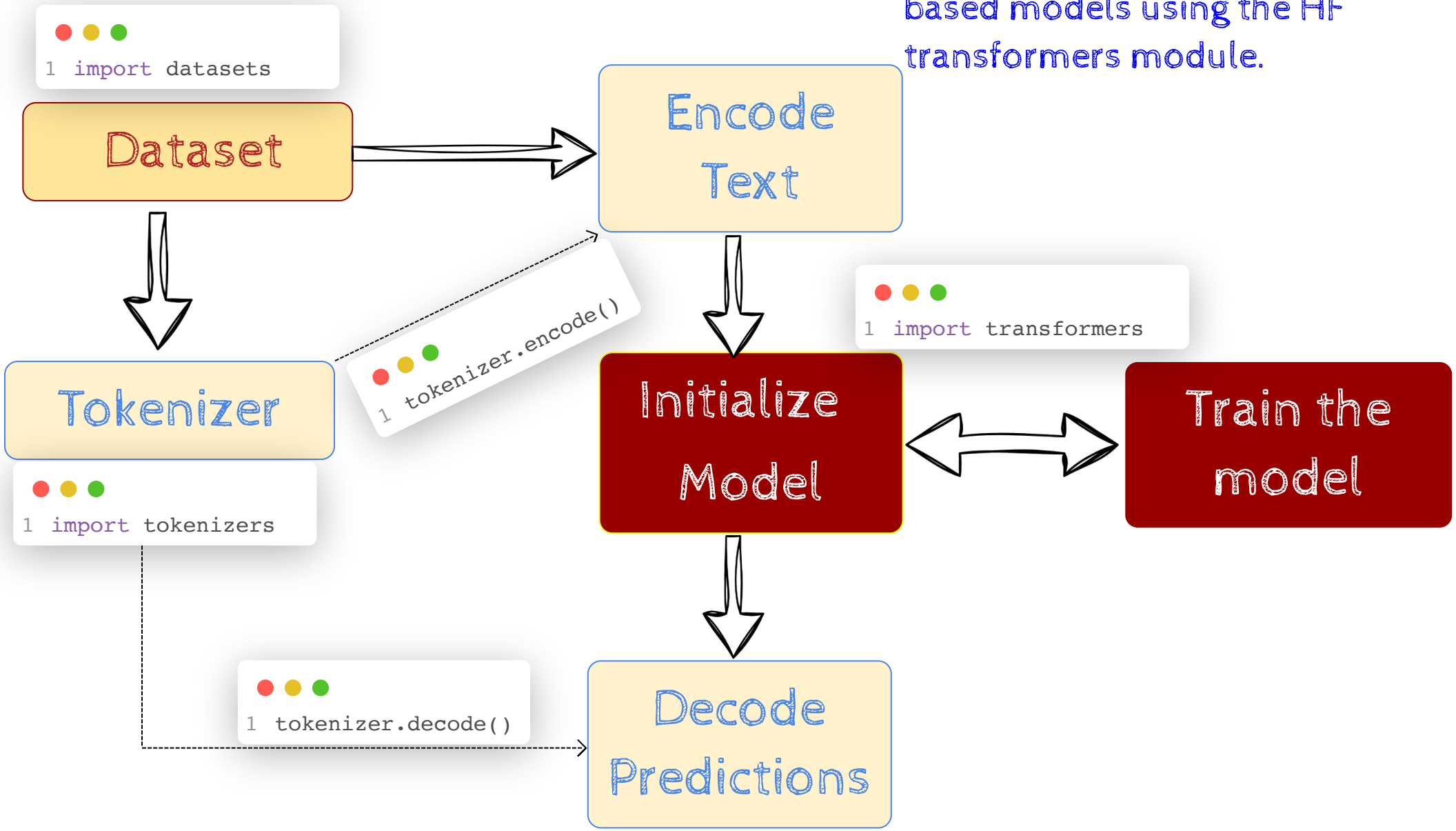

```
1 import evalaute
```

# So far


1 import datasets
Dataset
Encode Text
1 tokenizer.encode()
Tokenizer
1 import tokenizers
Next, we learned how to train a tokenizer that encodes input text according to the model-specific template and decodes the model's predictions
1 tokenizer.decode()
Decode Predictions

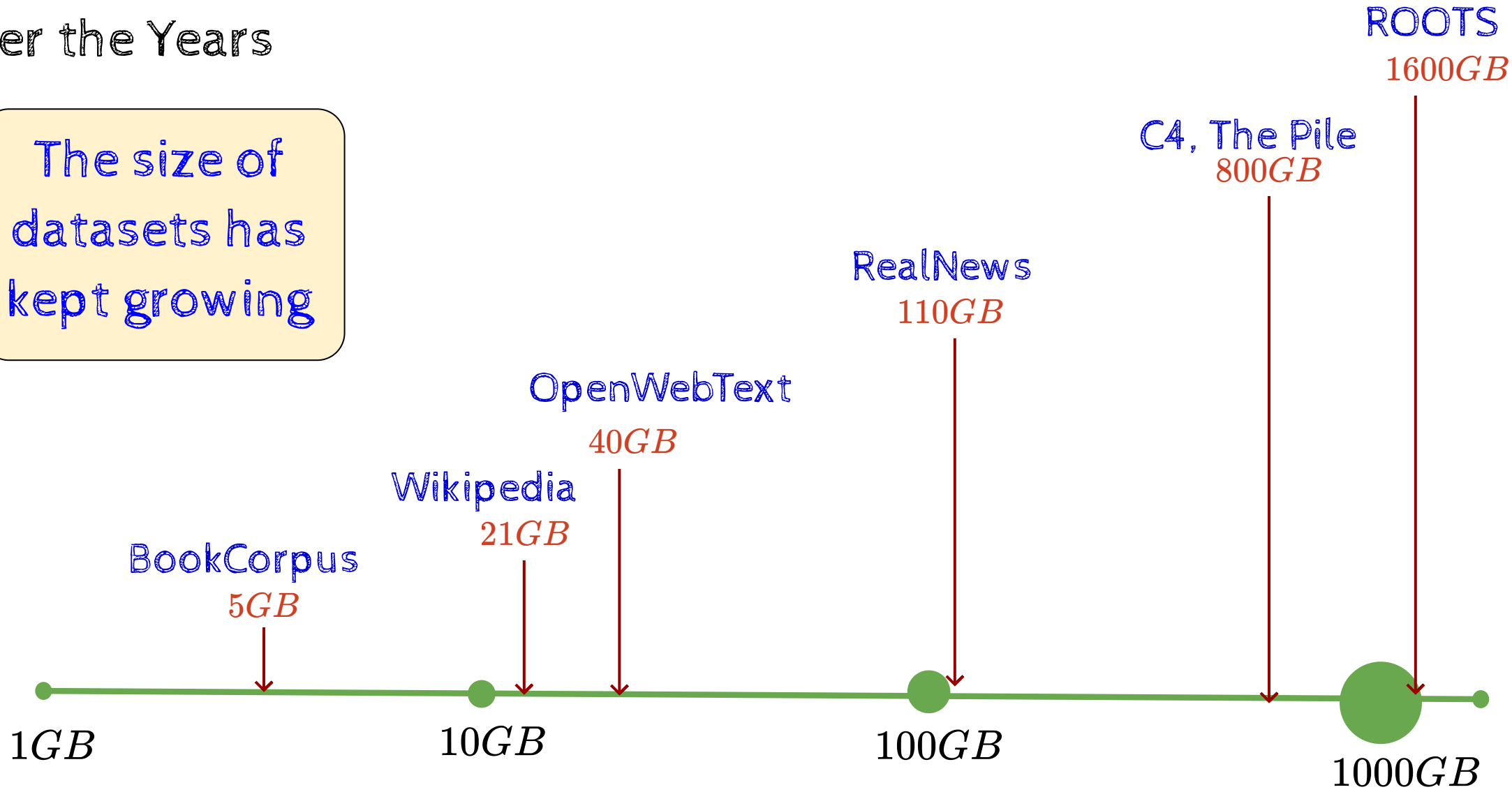
# In this week

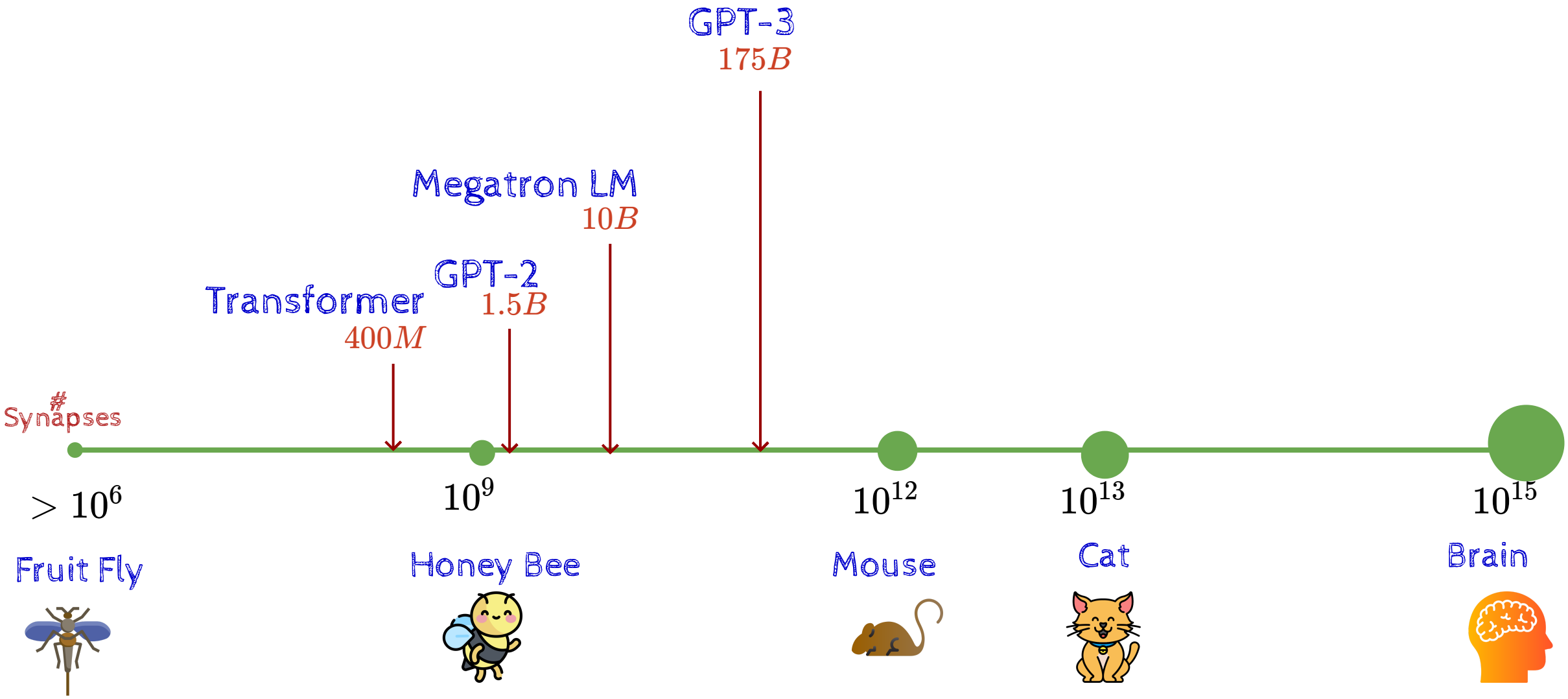
we will learn how to train transformer based models using the HF transformers module.



# Over the Years

The size of datasets has kept growing



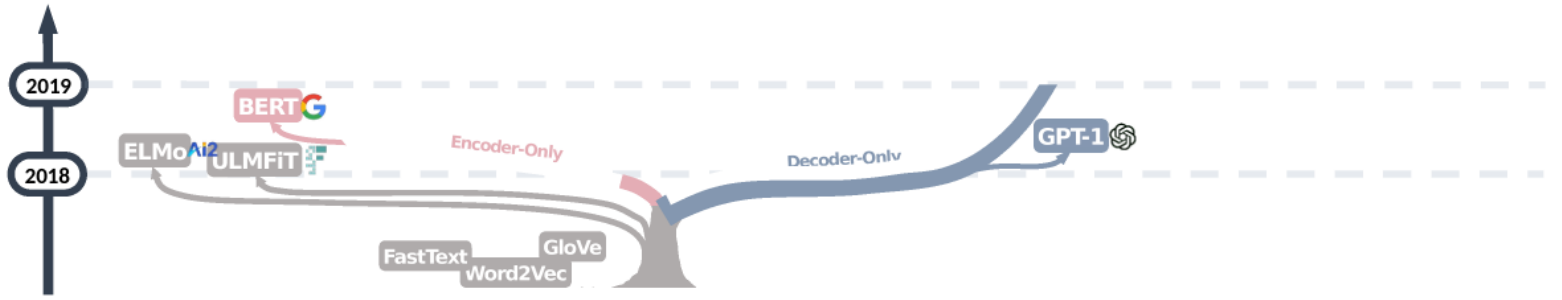

GPT-3
175B
Megatron LM
10B
GPT-2
1.5B
Transformer
400M
# Synapses
> 10^6
10^9
10^12
10^13
10^15
Fruit Fly
Honey Bee
Mouse
Cat
Brain

# Here is the Evolution tree

We can group the transformer based models into three categories

Encoder-only models (BERT and its variants) dominated the field for about two years

Decoder-only models (GPT) soon emerged for the task of Language modeling



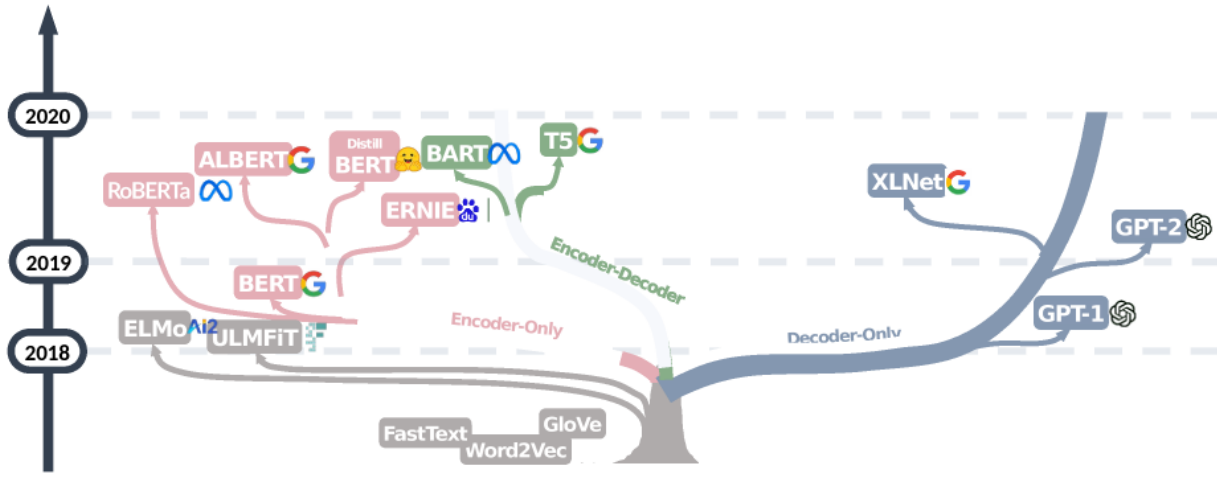
Encoder-only models (BERT and its variants) dominated the field for about two years

GPT-2 and T5 advocated for decoder only models and encoder-decoder models, respectively.

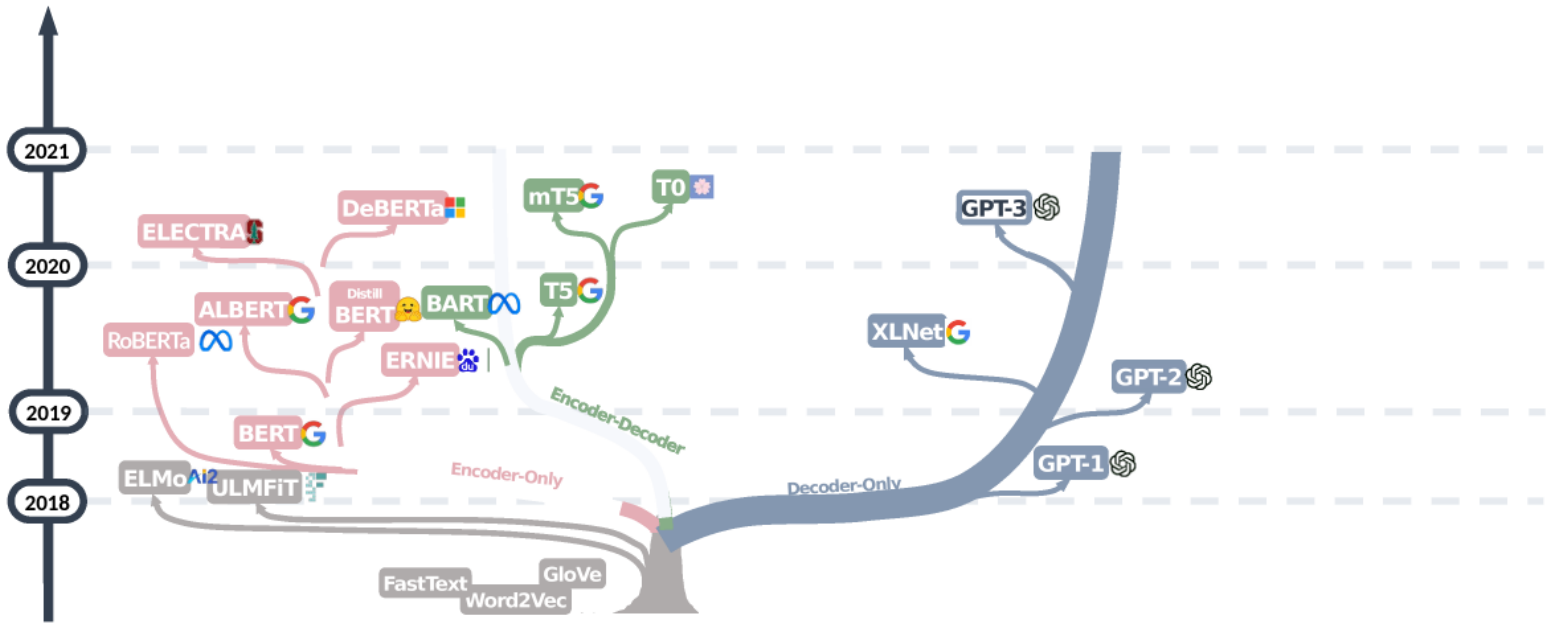
- Scale the model or dataset or both?
- What is the proportion of scaling?

Model

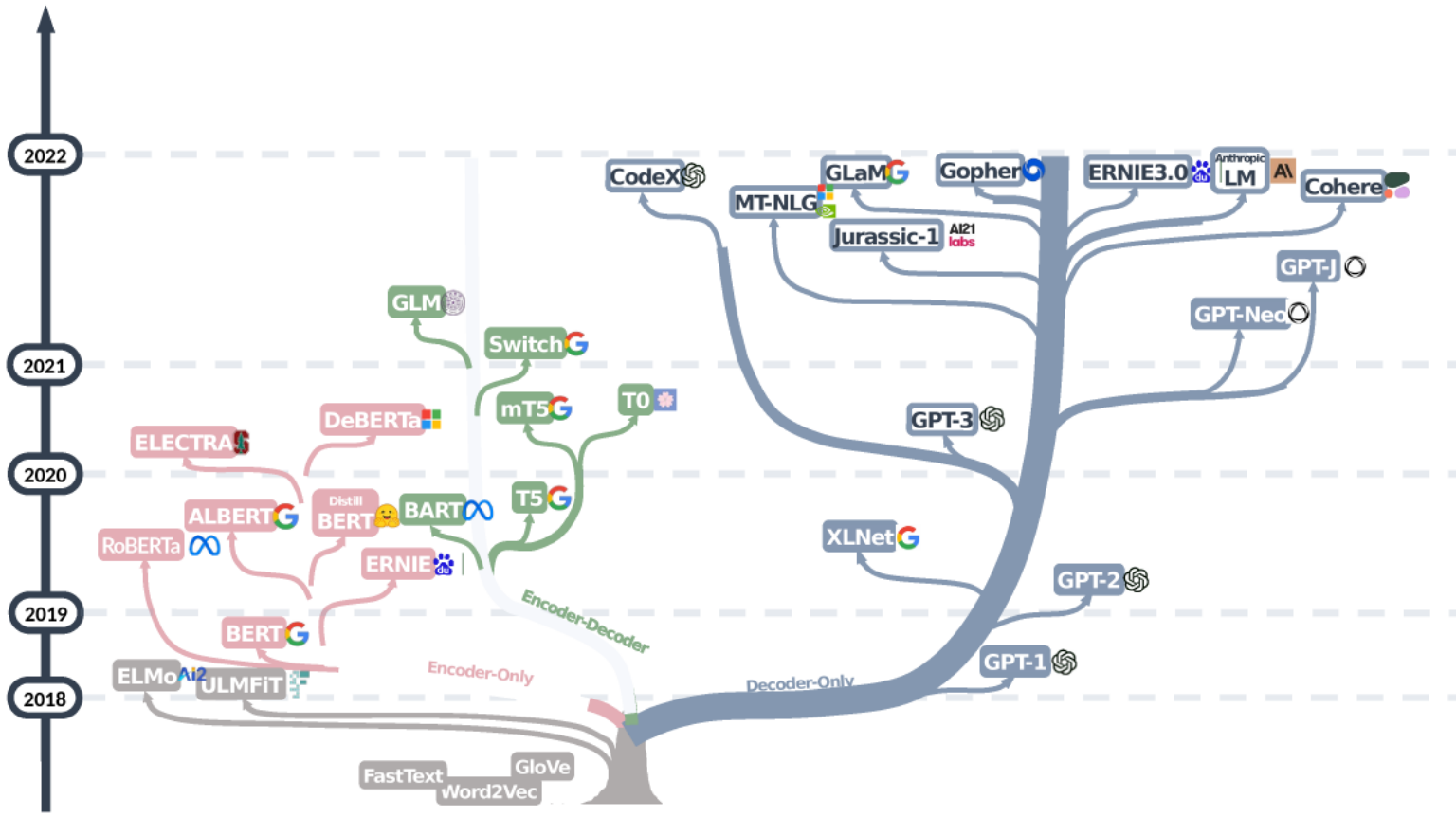
Dataset



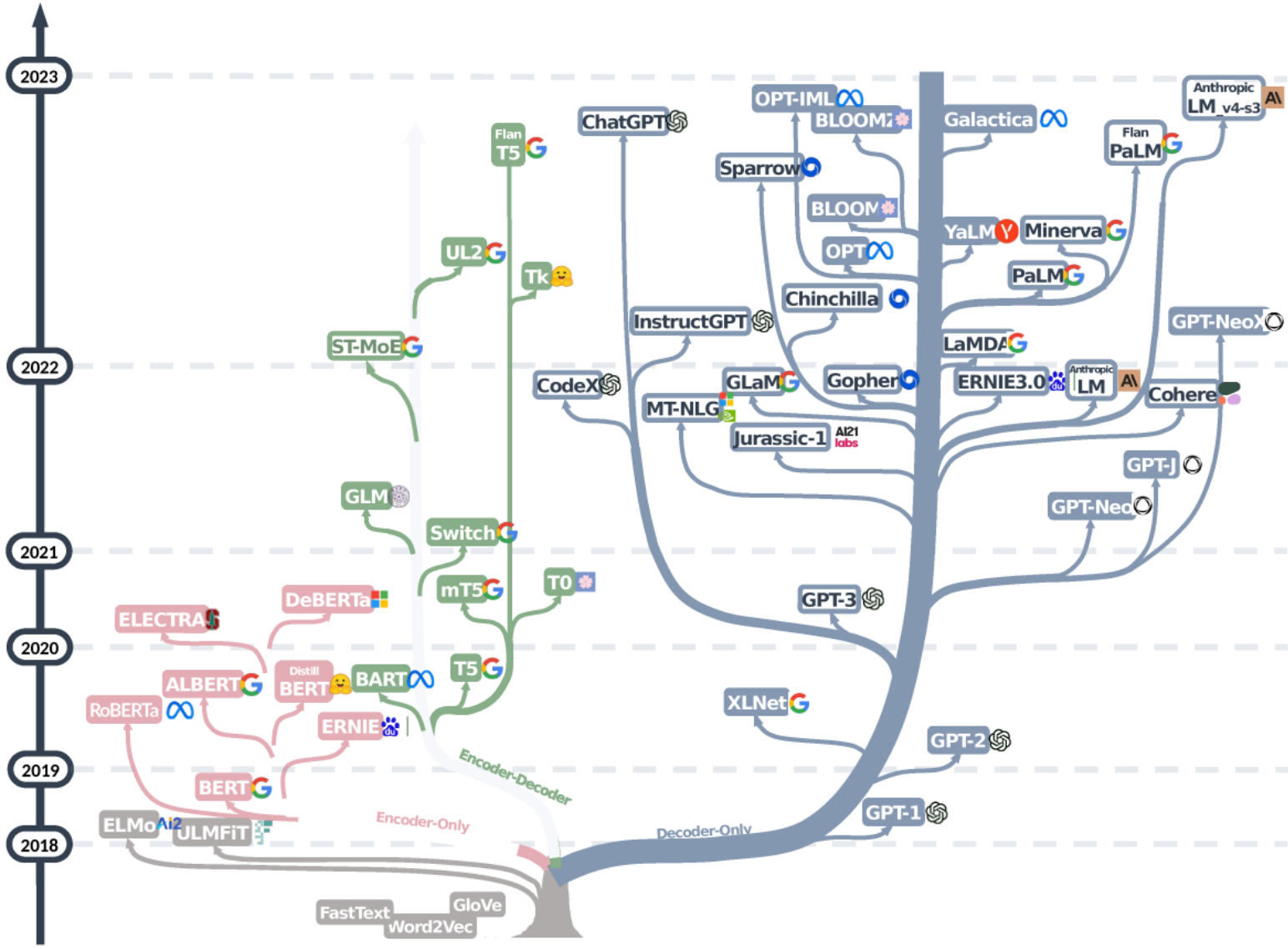
# The tree has grown bigger



The tree has grown bigger and bigger



The tree has grown bigger and bigger in a short span of time

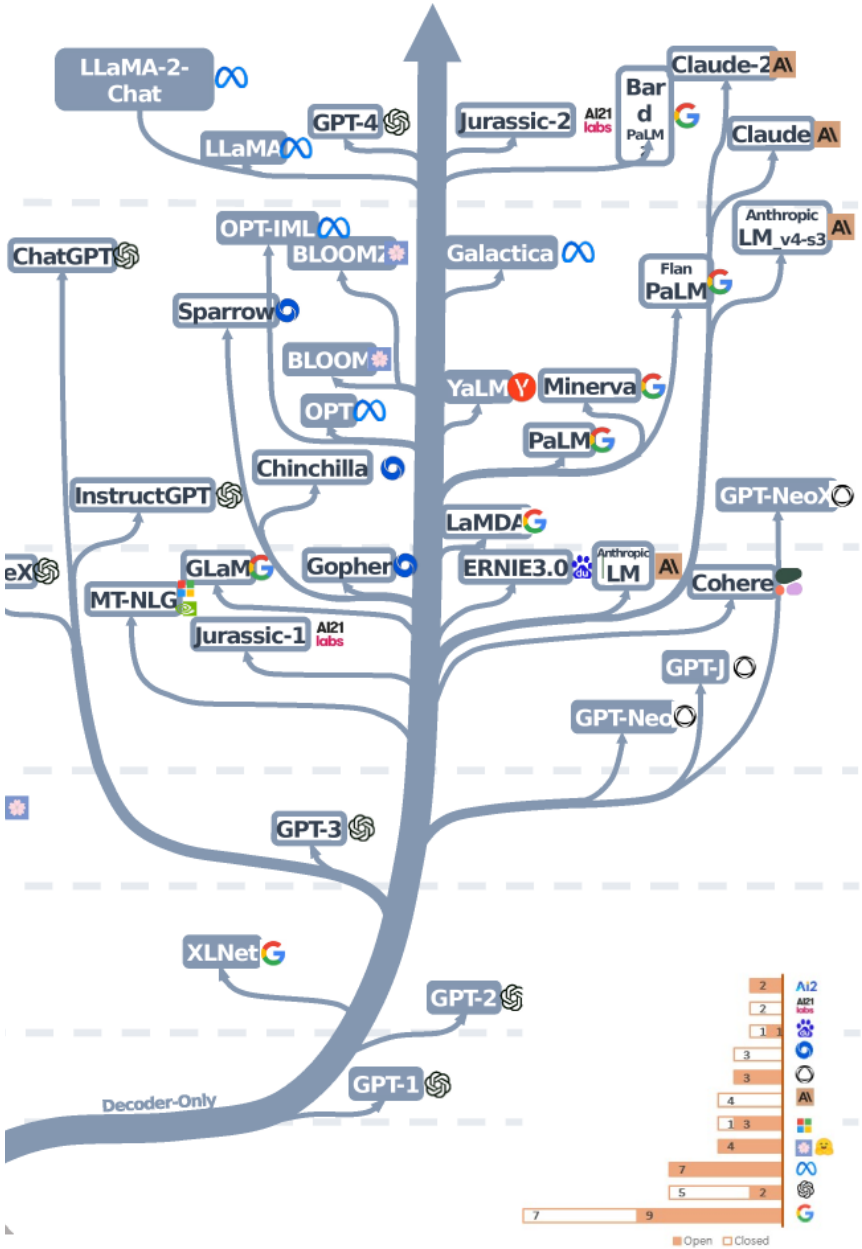


Even if we select only the decoder branch in the evaluation tree

there would still be dozens of models

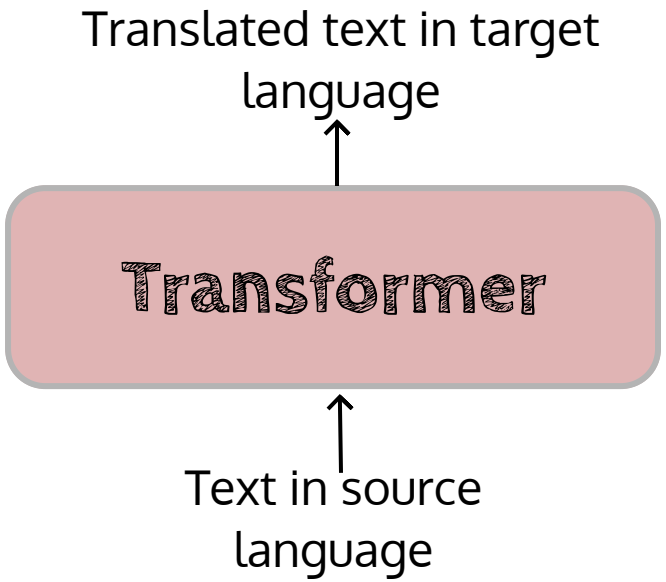
However, all of these LLMs go through a phase called pretraining.

Let's try to understand this using the GPT (Generative Pretrained Transformer) model.



In the DL course, we learned about the components of the transformer architecture in the context of machine translation.

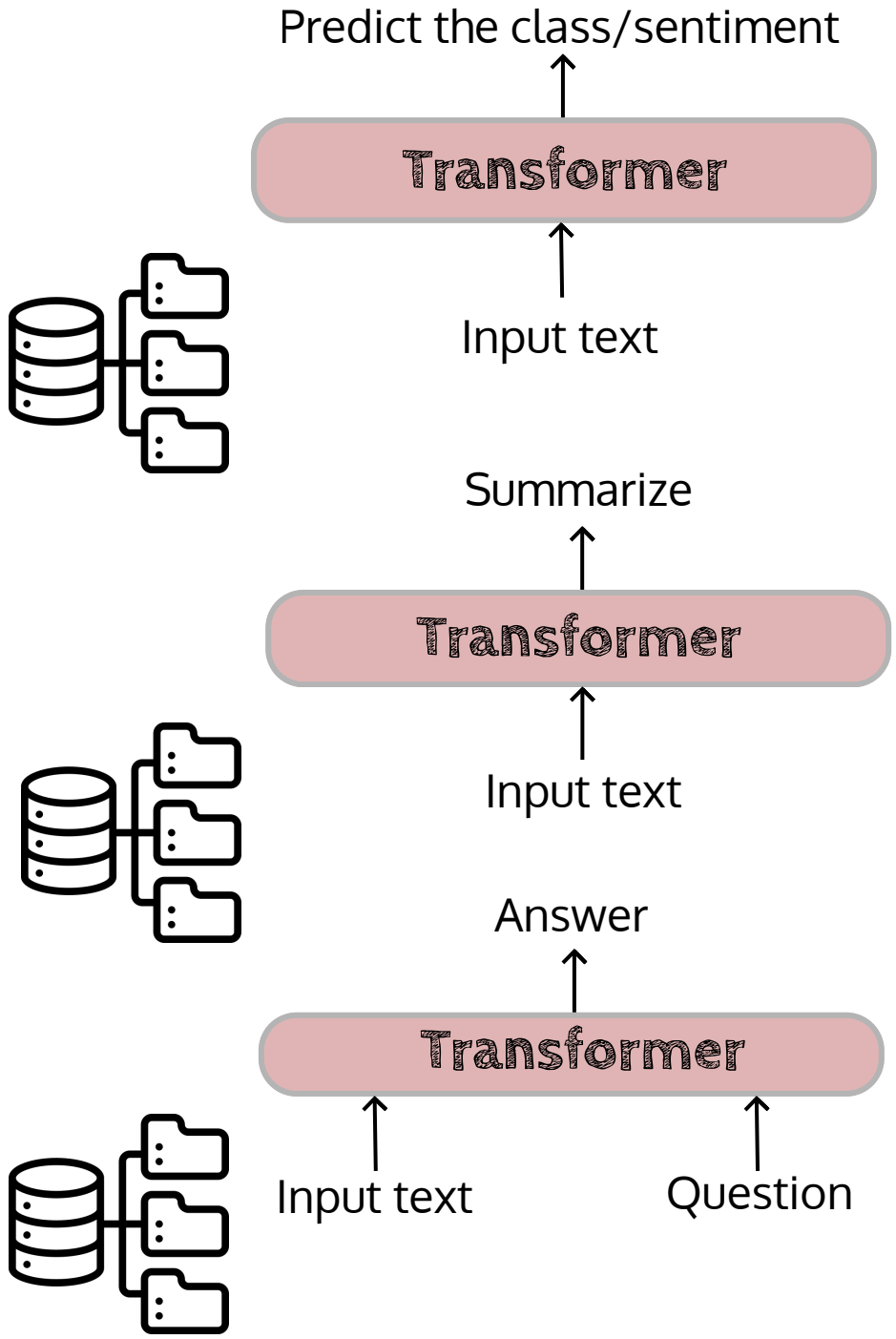
At a higher level of abstraction, we can view it as a black box that takes an input and generates an output.



In the DL course, we learned about the components of the transformer architecture in the context of machine translation.

What if we want to use the transformer architecture for other NLP tasks?

We need to train a separate model for each task using a dataset specific to the task



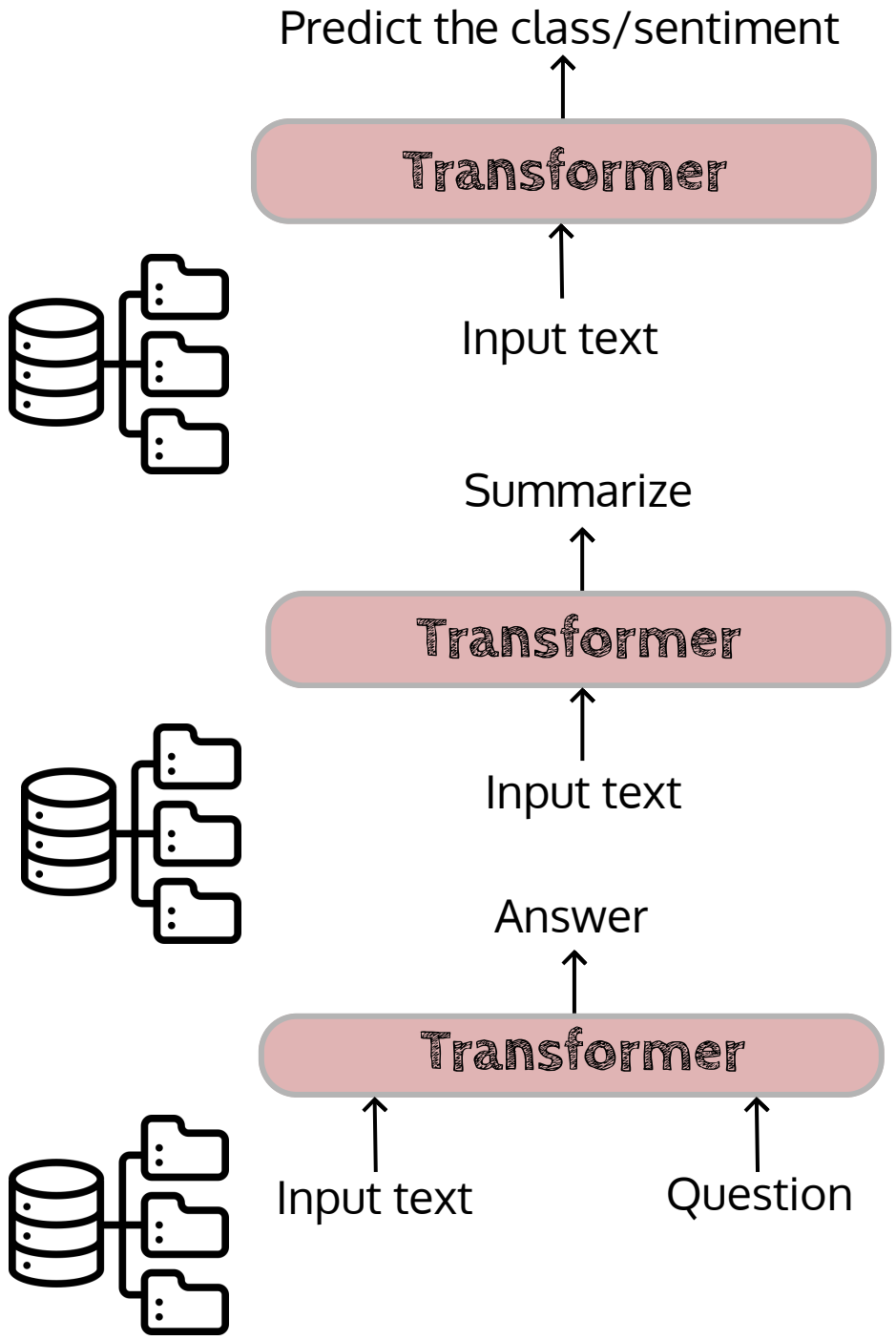
In the DL course, we learned about the components of the transformer architecture in the context of machine translation.

What if we want to use the transformer architecture for other NLP tasks?

We need to train a separate model for each task using dataset specific to the task

If we train the architecture from scratch (that is, by randomly initializing the parameters) for each task, it takes a long time for convergence

Often, we may not have enough labelled samples for many NLP tasks

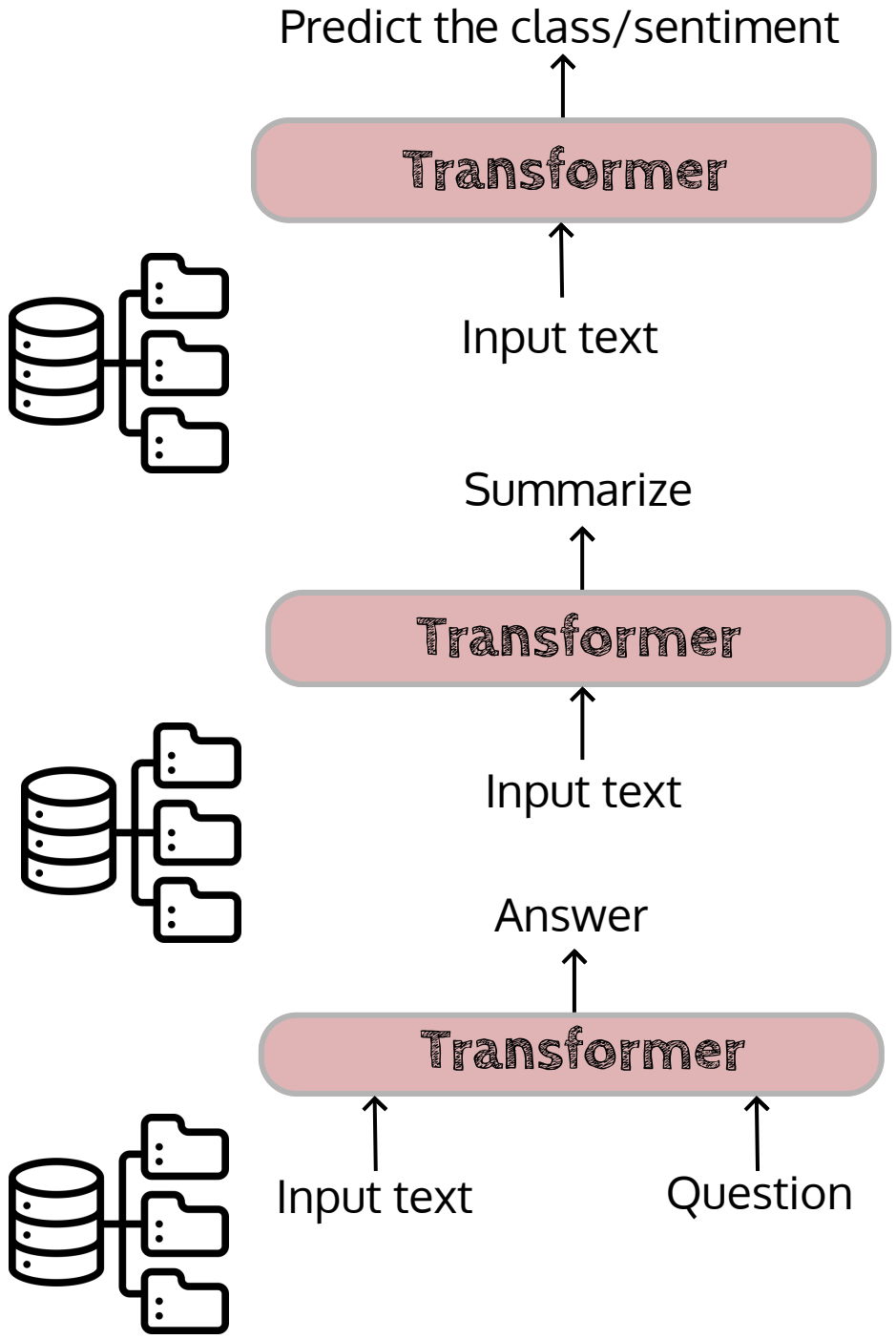


Moreover, preparing labelled data is laborious and costly

On the other hand,

we have a large amount of unlabelled text easily available on the internet


Can we make use of such unlabelled data to train a model?



# Module 3.1 : Language Modelling

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

# Motivation

Assume that we ask questions to a lay person based on a statement or some excerpt

" Wow, India has now reached the moon"

An excerpt from business today "What sets this mission apart is the pivotal role of artificial intelligence (AI) in guiding the spacecraft during its critical descent to the moon's surface."

He likes to stay

He likes to stray

He likes to sway

Is this sentence expressing a positive or a negative sentiment?

Did the lander use AI for soft landing on the moon?

Are these meaningful sentences?

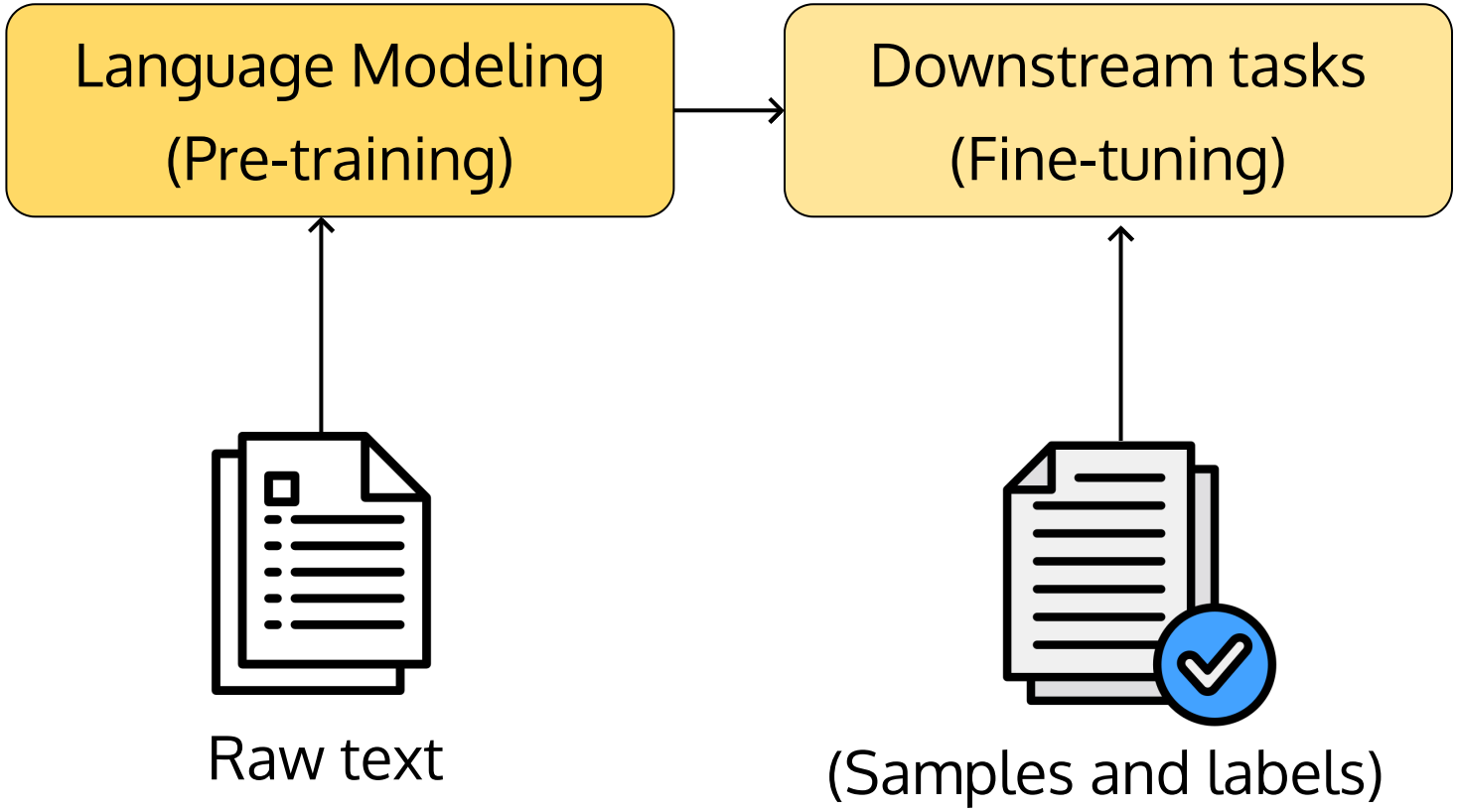
The person will most likely answer all the questions, even though he/she may not be explicitly trained on any of these tasks. How?

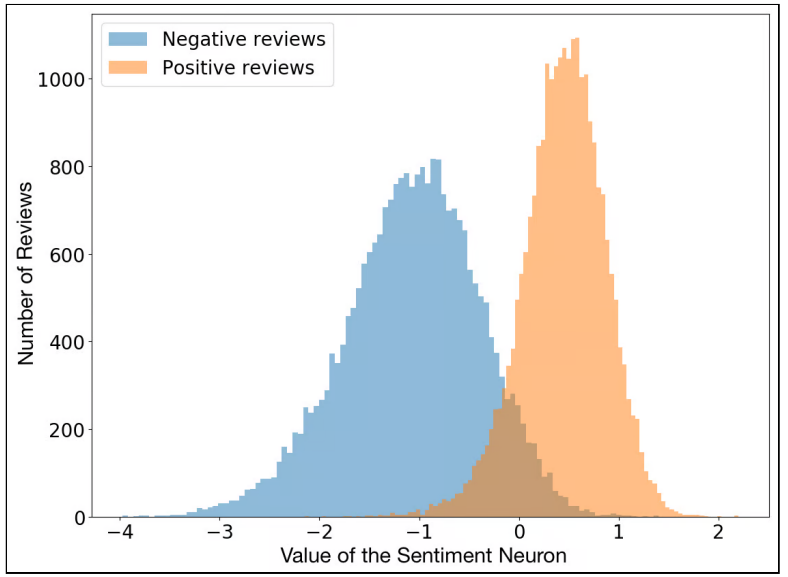
We develop a strong understanding of language through various language based interactions( listening/reading) over our life time without any explicit supervision

# Idea

Can a model develop basic understanding of language by getting exposure to a large amount of raw text? [pre-training]

More importantly, after getting exposed to such raw data can it learn to perform well on downstream tasks with minimum supervision?
[Supervised Fine-tuning]




With this representation a **linear model** classifies reviews with 91.8% accuracy beating the SOTA (Ref)

...matches the performance of previous supervised systems using **30-100x fewer labeled** examples (Ref)

# Language modelling

Let $\mathcal{V}$ be a vocabulary of language (i.e., collection of all unique words in the language)

We can think of a sentence as a sequence $X_1, X_2, \cdots, X_n$, where $X_i \in \mathcal{V}$

For example, if $\mathcal{V} = \{an, apple, ate, I\}$, some possible sentences (not necessarily grammatically correct) are

a. An apple ate I

b. I ate an apple

c. I ate apple

d. an apple

e. ....

Intuitively, some of these sentences are more probable than others.

What do we mean by that?

Intuitively, we mean that give a very very large corpus, we expect some of these sentences to appear more frequently than others (hence, more probable)

We are now looking for a function which takes a sequence as input and assigns a probability to each sequence

$$f : (X_1, X_2, \cdots X_n) \rightarrow [0, 1]$$

Such a function is called a language model.

# Language modelling

$$P(x_1, x_2, \cdots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\cdots P(x_T|x_{T-1}, \cdots, x_1)$$

$$= \prod_{i=1}^{T} P(x_i|x_1, \cdots, x_{i-1})$$

If we naively assume that the words in a sequence are independent of each other then

$$\boxed{P(x_1, x_2, \cdots, x_T) = \prod_{i=1}^{T} P(x_i)}$$

How do we enable a model to understand language?

Simple Idea: Teach it the task of predicting the next token in a sequence..

You have tons of sequences available on the web which you can use as training data

## Chandrayaan-3

Article Talk

From Wikipedia, the free encyclopedia

**Chandrayaan-3** (/ ˌtʃʌndrəˈjɑːn/ *CHUN-drə-YAHN*) is the third mission in the Chandrayaan programme, a series of lunar-exploration missions [redacted] by the Indian Space Research Organisation (ISRO).[7] Launched on 14 July 2023, the mission consists of a l[redacted] named *Vikram* and a lunar rover named *Pragyan*, similar to those launched aboard Chandrayaan-2 in 2019.

Chandrayaan-3 was [redacted] from Satish Dhawan Space Centre on 14 July 2023. The spacecraft [redacted] lunar orbit on 5 August, and the lander [redacted] down near the Lunar south pole[8] on 23 August at 18:03 IST (12:33 UTC), making India the fourth country to [redacted] land on the Moon, and the first to do so near the lunar south [redacted] On 3 September the lander hopped and repositioned itself 30–40 cm (12–16 in) from its landing site.[13] [redacted] the completion of its mission objectives, it was hoped that the lander and rover would [redacted] for extra tasks, on 22 September 2023,

**BECAUSE** he is the [redacted] in conceiving and implementing India's digital building blocks such as UPI, Aadhaar, eKYC and FASTag. [redacted] also helped the government develop the IT infrastructure to [redacted] GST and the Ayushman Bharat Yojana. The cerebral [redacted] is the government's go-to person for tech intervention, irrespective of the party in power

**BECAUSE** he was [redacted] in the launch of the Beckn protocol, now being used in sectors such as e-commerce, mobility and health. He was [redacted] the government-backed Open Network for Digital [redacted] aimed at helping small Indian retailers fight back against e-com giants such as [redacted] and Flipkart. He is also supporting IIT Madras's AI4Bharat, an [redacted] AI initiative, in 22 Indian languages

?

Roughly speaking, this task of predicting the next token in a sequence is called language modelling

However, we know that the words in a sentence are not independent but depend on the previous words

a. I enjoyed reading a **book**

b. I enjoyed reading a **thermometer**

The presence of "enjoyed" makes the word "book" more likely than "thermometer"

Hence, the naive assumption does not make sense

$\prod_{i=1}^{T} P(x_i|x_1, \cdots, x_{i-1})$ : Current word $\underbrace{x_i}$ depends on previous words $\underbrace{x_1, \cdots, x_{i-1}}$

$\prod_{i=1}^{T} P(x_i|x_1,\cdots,x_{i-1})$ : Current word $\underbrace{x_i}$ depends on previous words $\underbrace{x_1,\cdots,x_{i-1}}$

How do we estimate these conditional probabilities?

**One solution:** use **autoregressive models** where the conditional probabilities are given by parameterized functions with a fixed number of parameters (like transformers).

# Causal Language Modelling (CLM)

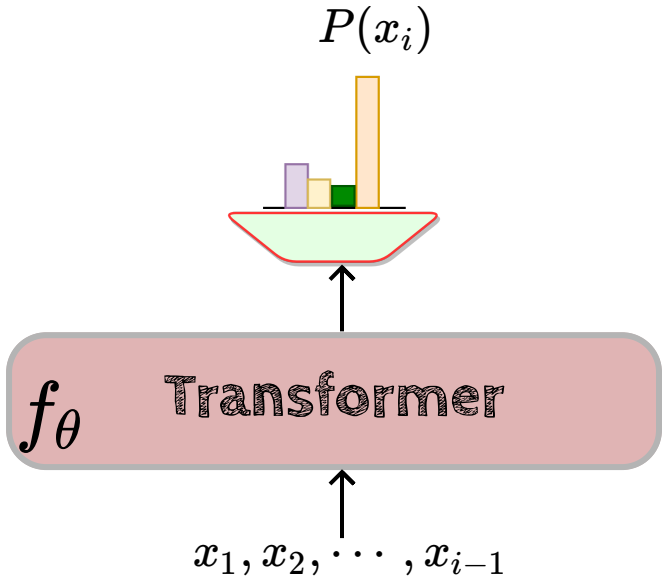
$$P(x_1, x_2, \cdots, x_T) = \prod_{i=1}^{T} P(x_i | x_1, \cdots, x_{i-1})$$

$$= P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\cdots P(x_T|x_{T-1}, \cdots, x_1)$$

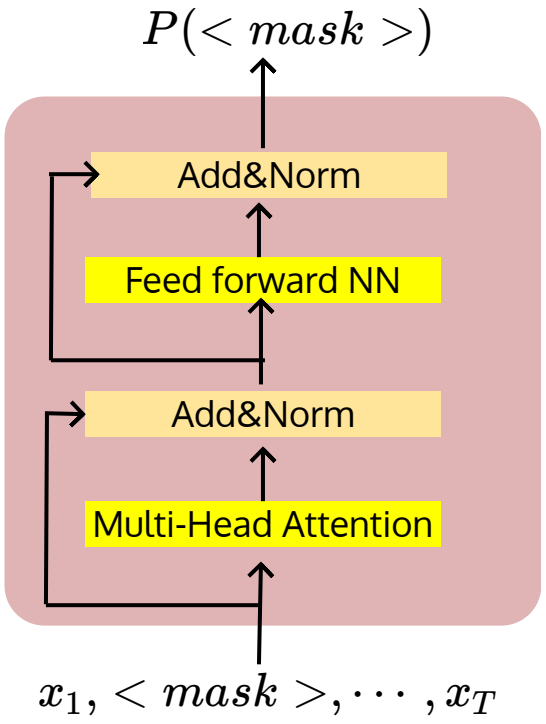
We are looking for $f_\theta$ such that

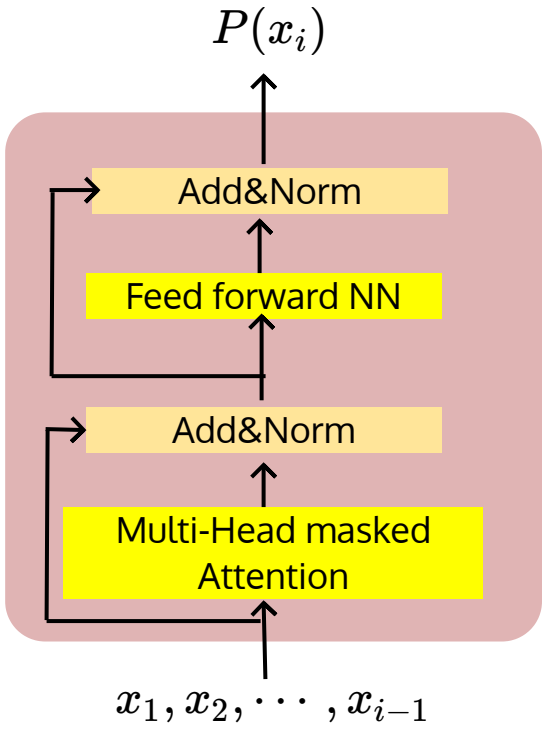
$$P(x_i | x_1, \cdots, x_{i-1}) = f_\theta(x_i | x_1, \cdots, x_{i-1})$$

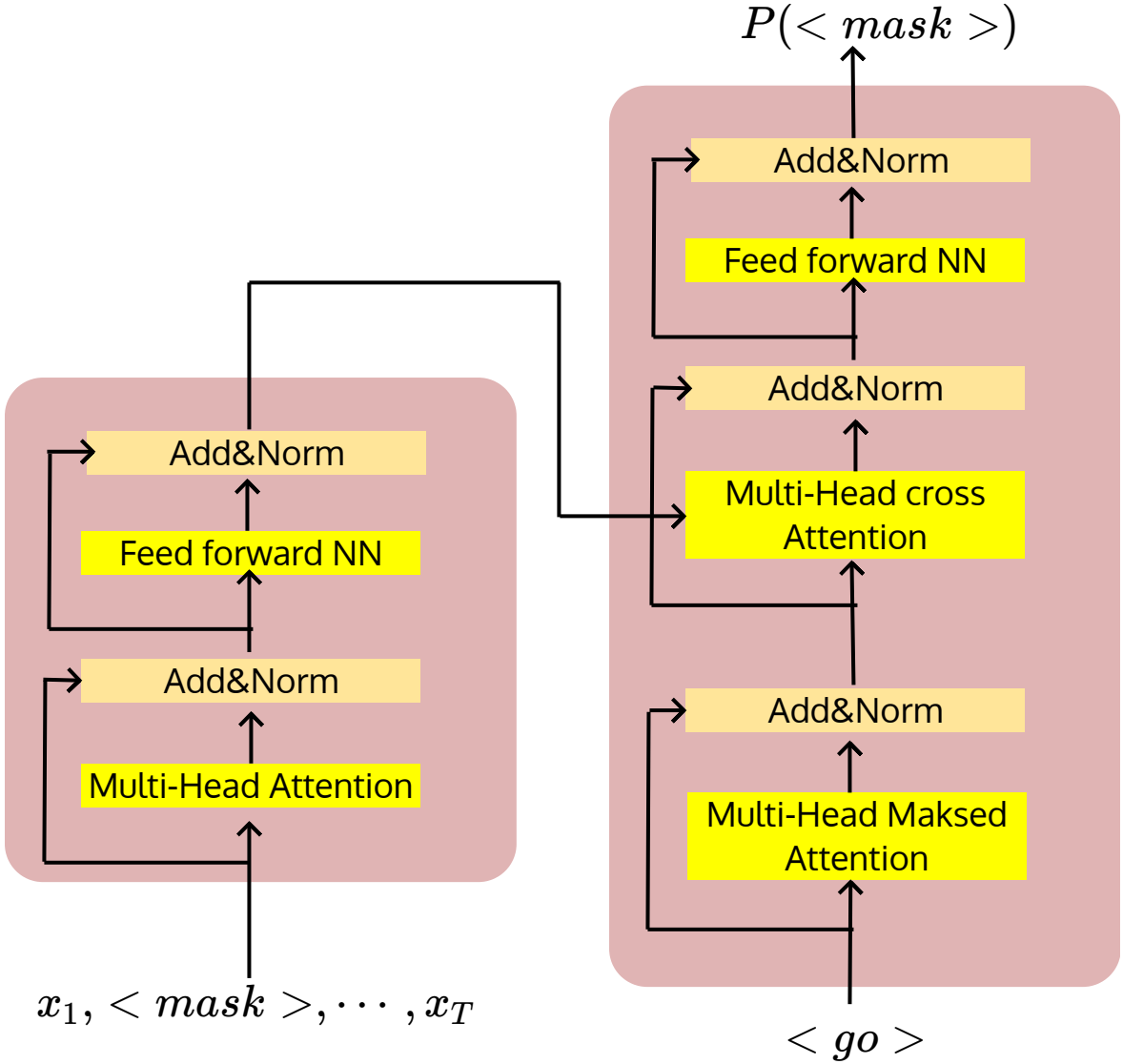
Can $f_\theta$ be a transformer?



# Some Possibilities



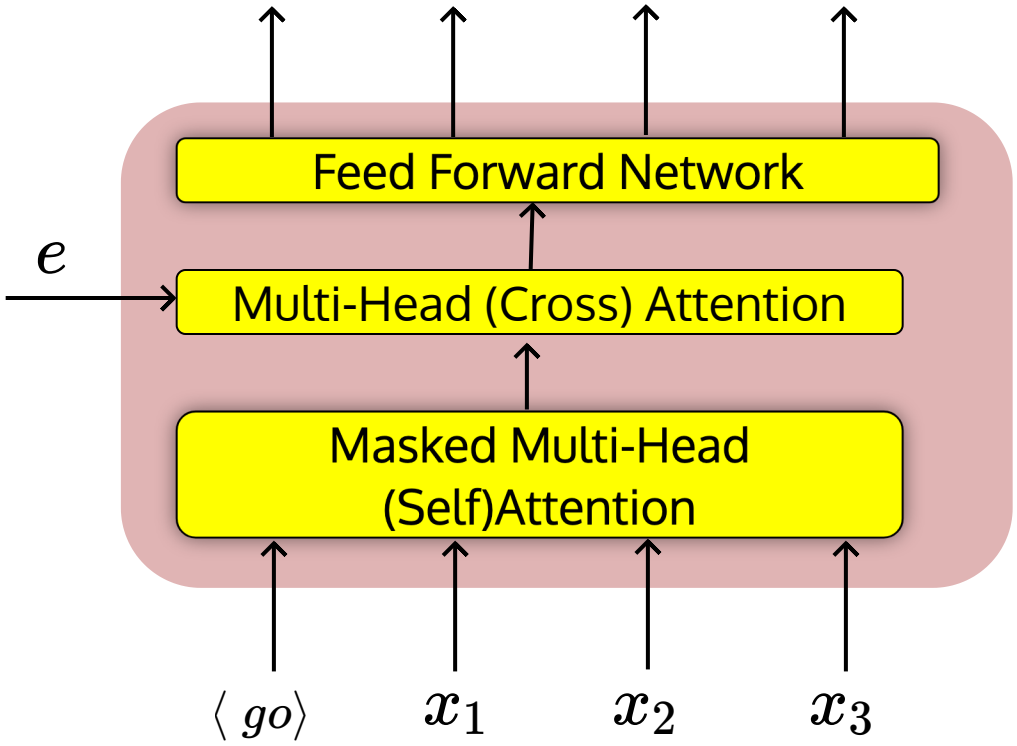




Using only the encoder of the transformer (encoder only models)

Using only the decoder of the transformer (decoder only models)

Using both the encoder and decoder of the transformer (encoder decoder models)




The input is a sequence of words

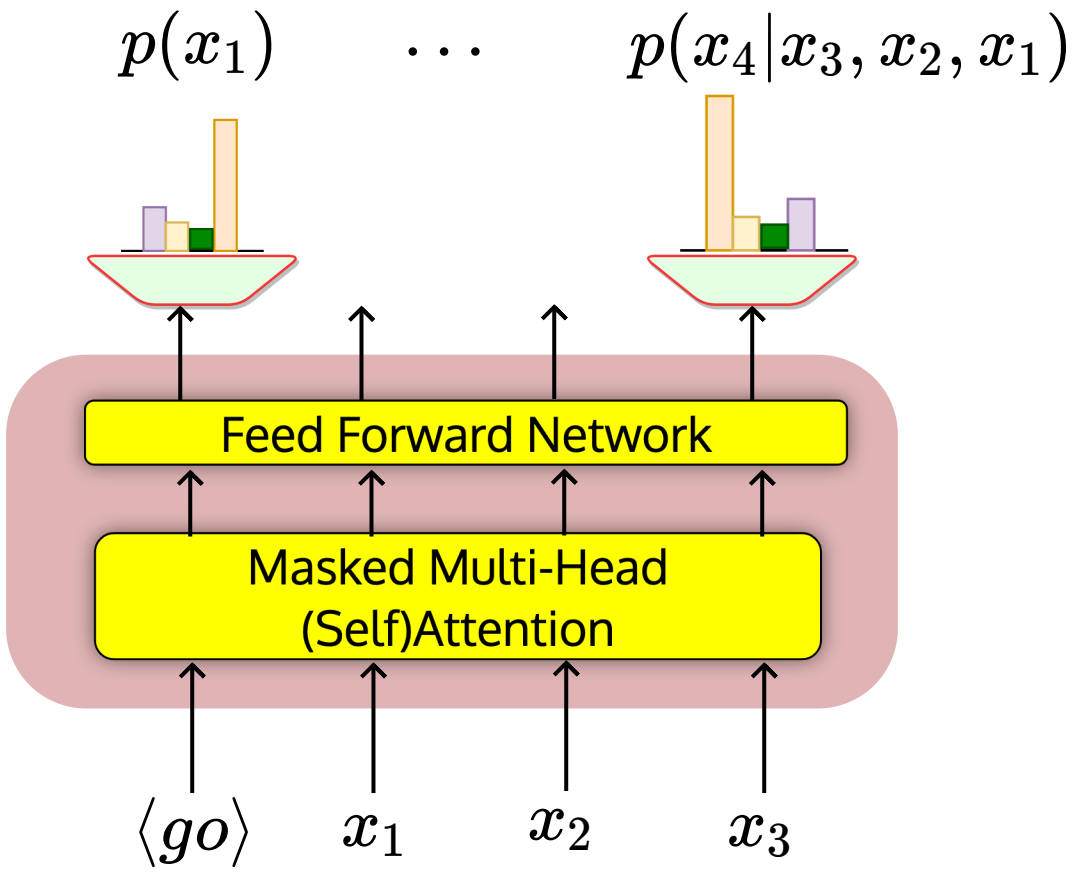
We want the model to see only the present and past inputs.

We can achieve this by applying the mask.

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The masked multi-head attention layer is required.

However, we do not need multi-head cross attention layer (as there is no encoder).




The outputs represent each term in the chain rule

$$= P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)P(x_4|x_3, x_2, x_1)$$

However, this time the probabilities are determined by the parameters of the model,

$$= P(x_1; \theta)P(x_2|x_1; \theta)P(x_3|x_2, x_1; \theta)P(x_4|x_3, x_2, x_1; \theta)$$

Therefore, the objective is to maximize the likelihood $L(\theta)$

# Module 3.2 : Generative Pretrained Transformer (GPT)

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

# Generative Pretrained Transformer (GPT)

Now we can create a stack $(n)$ of modified decoder layers (called transformer block in the paper)

Let, $X$ denote the input sequence

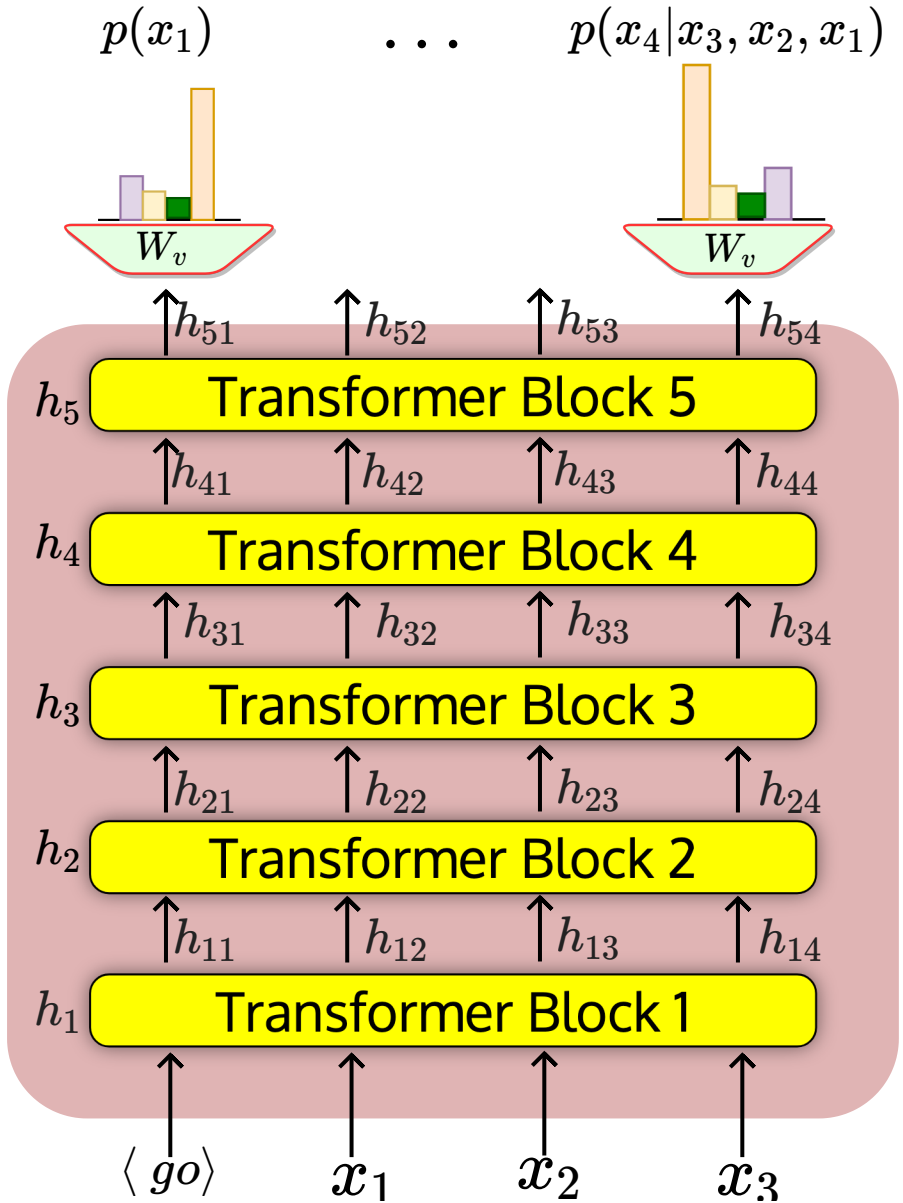
$$h_0 = X \in \mathbb{R}^{T \times d_{model}}$$

$$h_l = transformer\_block(h_{l-1}), \forall l \in [1, n]$$

Where $h_n[i]$ is the $i-$the output vector in $h_n$ block.

$$P(x_i) = softmax(h_n[i]W_v)$$

$$\mathscr{L} = \sum_{i=1}^{T} \log(P(x_i|x_1, \cdots, x_{i-1}))$$



# Input data

BookCorpus

The corpus contains 7000 unique books, 74 Million sentences and approximately 1 Billion words across 16 genres

Also, uses long-range contiguous text
(i.e., no shuffling of sentences or paragraphs)

Tokenizer: Byte Pair Encoding

Vocab size $|\mathcal{V}|$: 40478

Embedding dim: 768

Side Note: The other benchmark dataset called 1B words could also be used. However, the sentences are not contiguous (no entailment)

# MODEL

Contains 12 decoder layers (transformer blocks)

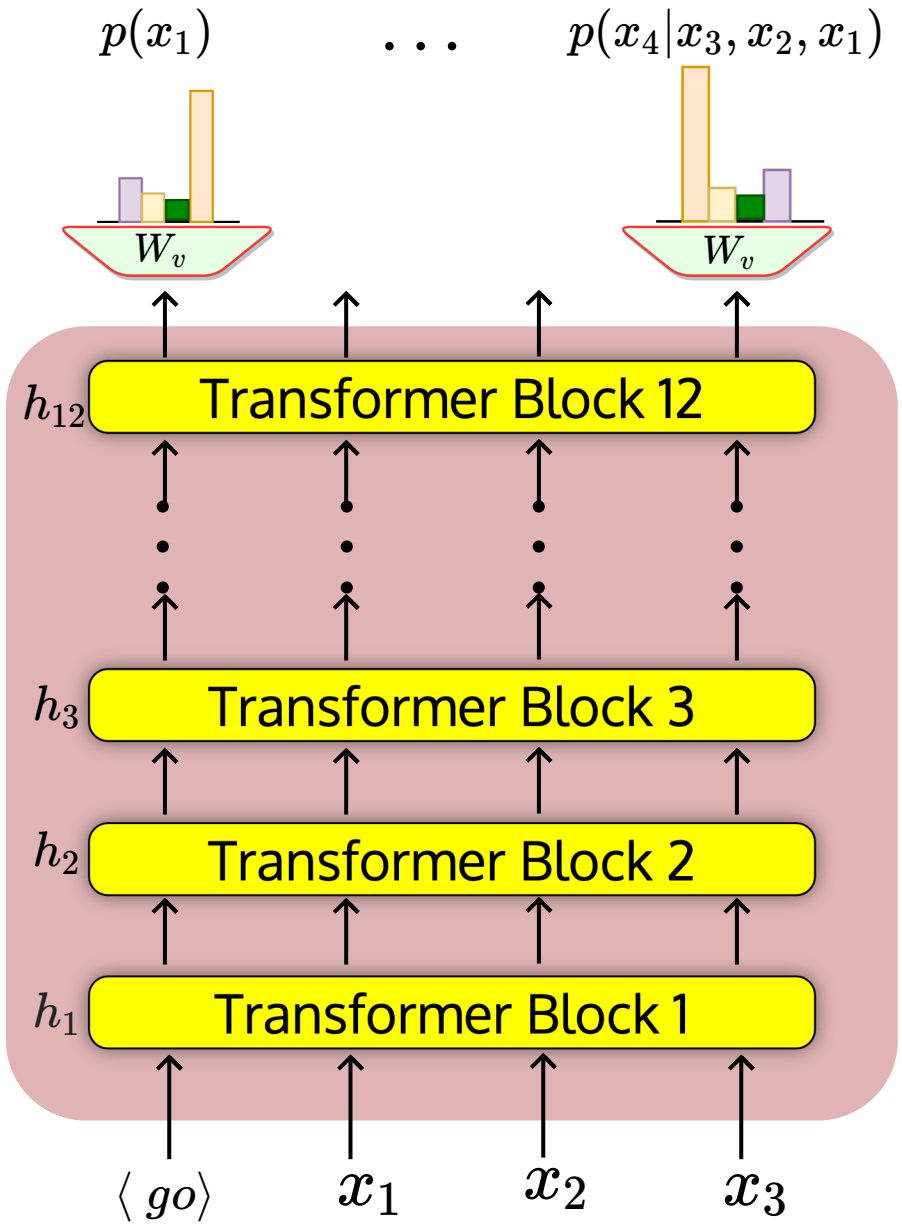
Context size: 512

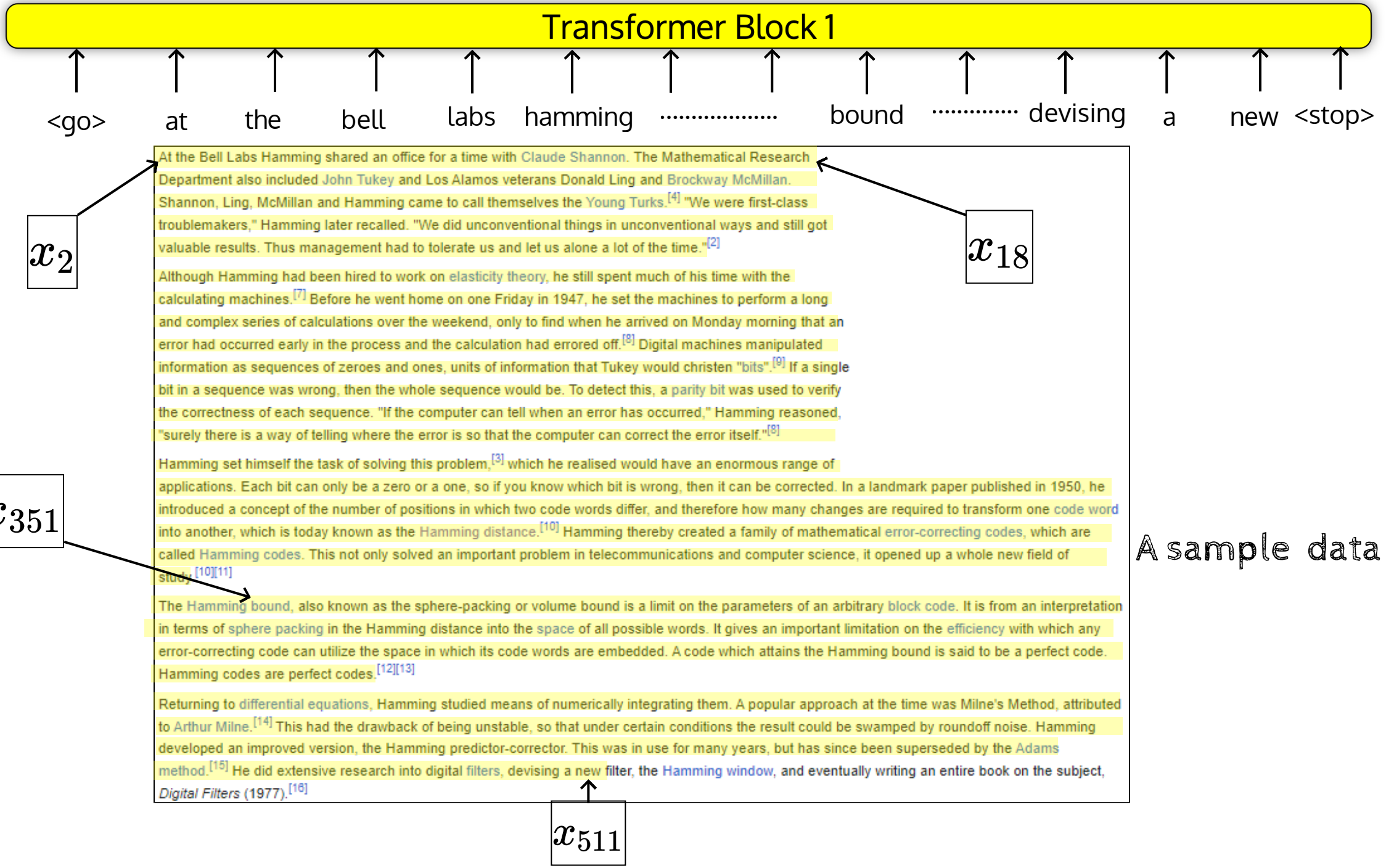
Attention heads: 12

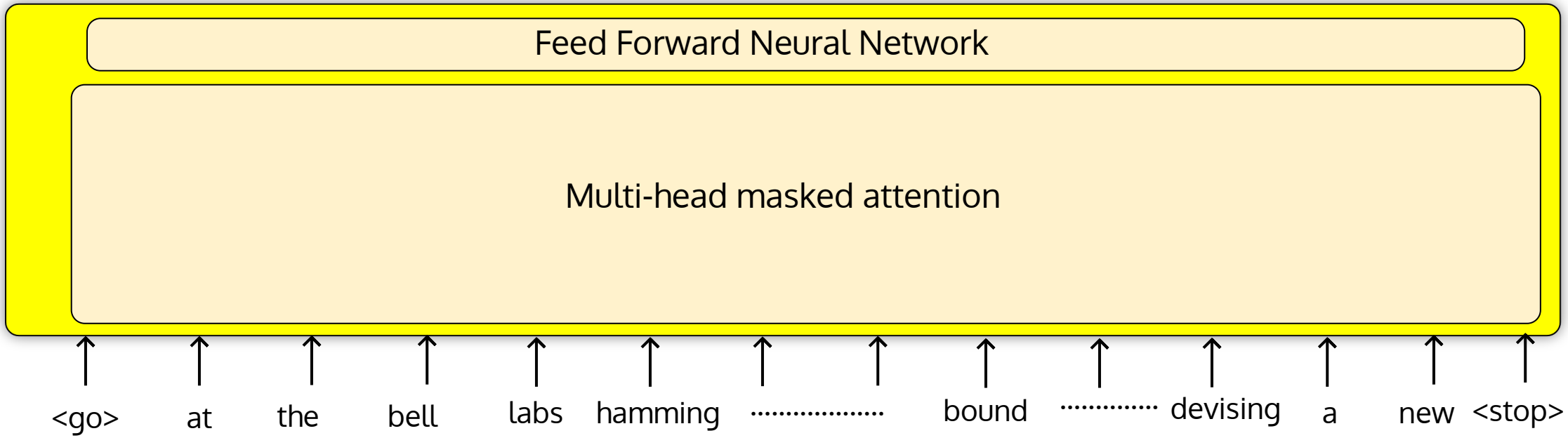
FFN hidden layer size: $768 \times 4 = 3072$

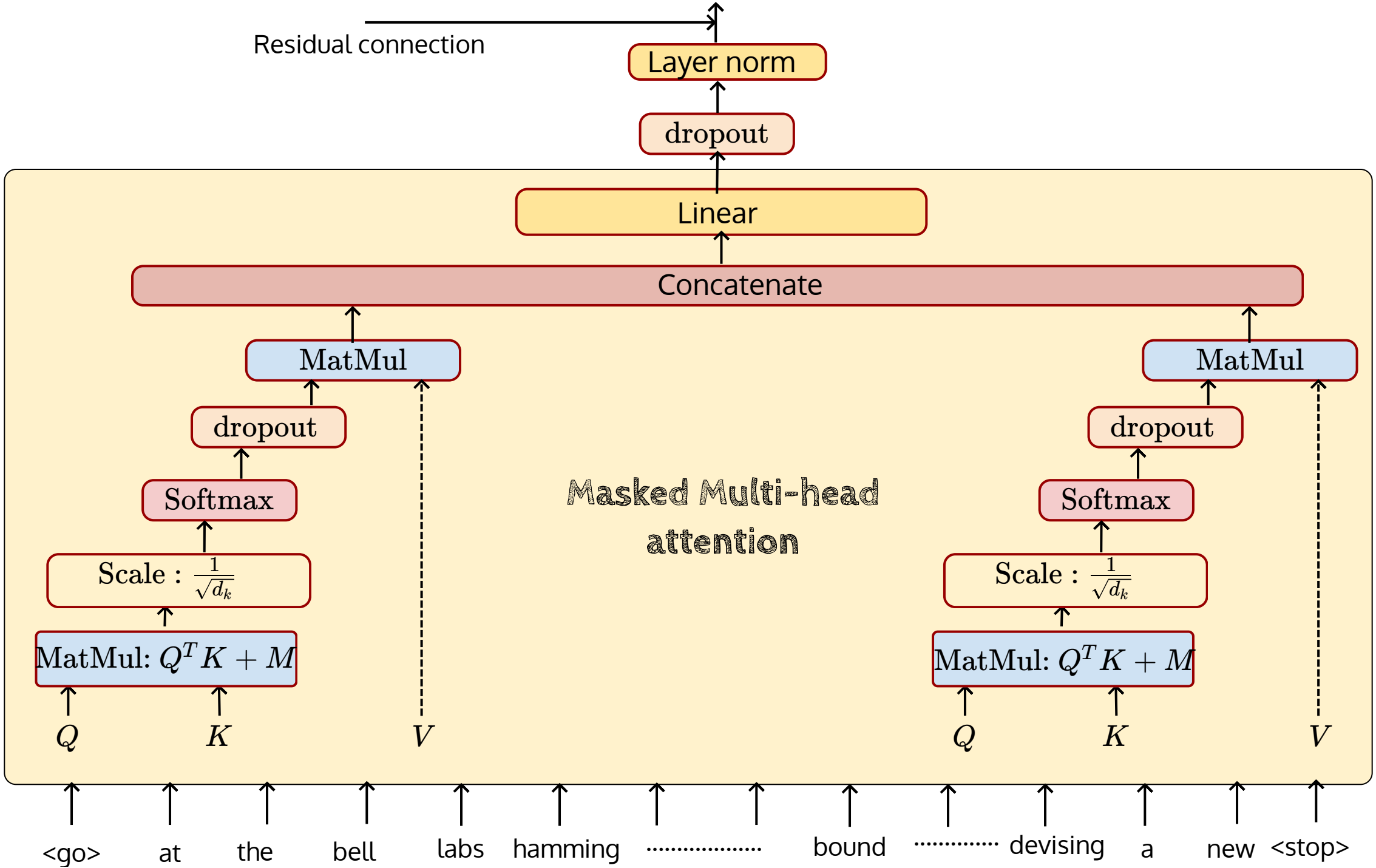
**Activation:** Gaussian Error Linear Unit (GELU)

Dropout, layer normalization, and residual connections were implemented to enhance convergence during training.




Transformer Block 1
<go>
at
the
bell
labs
hamming
bound
devising
a
new
<stop>
$x_2$
$x_{18}$
$x_{351}$
$x_{511}$
At the Bell Labs Hamming shared an office for a time with Claude Shannon. The Mathematical Research Department also included John Tukey and Los Alamos veterans Donald Ling and Brockway McMillan. Shannon, Ling, McMillan and Hamming came to call themselves the Young Turks.[4] "We were first-class troublemakers," Hamming later recalled. "We did unconventional things in unconventional ways and still got valuable results. Thus management had to tolerate us and let us alone a lot of the time."[2]
Although Hamming had been hired to work on elasticity theory, he still spent much of his time with the calculating machines.[7] Before he went home on one Friday in 1947, he set the machines to perform a long and complex series of calculations over the weekend, only to find when he arrived on Monday morning that an error had occurred early in the process and the calculation had errored off.[8] Digital machines manipulated information as sequences of zeroes and ones, units of information that Tukey would christen "bits".[9] If a single bit in a sequence was wrong, then the whole sequence would be. To detect this, a parity bit was used to verify the correctness of each sequence. "If the computer can tell when an error has occurred," Hamming reasoned, "surely there is a way of telling where the error is so that the computer can correct the error itself."[8]
Hamming set himself the task of solving this problem,[3] which he realised would have an enormous range of applications. Each bit can only be a zero or a one, so if you know which bit is wrong, then it can be corrected. In a landmark paper published in 1950, he introduced a concept of the number of positions in which two code words differ, and therefore how many changes are required to transform one code word into another, which is today known as the Hamming distance.[10] Hamming thereby created a family of mathematical error-correcting codes, which are called Hamming codes. This not only solved an important problem in telecommunications and computer science, it opened up a whole new field of study.[10][11]
The Hamming bound, also known as the sphere-packing or volume bound is a limit on the parameters of an arbitrary block code. It is from an interpretation in terms of sphere packing in the Hamming distance into the space of all possible words. It gives an important limitation on the efficiency with which any error-correcting code can utilize the space in which its code words are embedded. A code which attains the Hamming bound is said to be a perfect code. Hamming codes are perfect codes.[12][13]
Returning to differential equations, Hamming studied means of numerically integrating them. A popular approach at the time was Milne's Method, attributed to Arthur Milne.[14] This had the drawback of being unstable, so that under certain conditions the result could be swamped by roundoff noise. Hamming developed an improved version, the Hamming predictor-corrector. This was in use for many years, but has since been superseded by the Adams method.[15] He did extensive research into digital filters, devising a new filter, the Hamming window, and eventually writing an entire book on the subject, Digital Filters (1977).[16]
A sample data


Feed Forward Neural Network
Multi-head masked attention
<go>
at
the
bell
labs
hamming
.....
bound
.....
devising
a
new
<stop>


Residual connection
Layer norm
dropout
Linear
Concatenate
MatMul
dropout
Softmax
Scale : $\frac{1}{\sqrt{d_k}}$
MatMul: $Q^T K + M$
$Q$
$K$
$V$
Masked Multi-head attention
MatMul
dropout
Softmax
Scale : $\frac{1}{\sqrt{d_k}}$
MatMul: $Q^T K + M$
$Q$
$K$
$V$
<go>
at
the
bell
labs
hamming
.....
bound
.....
devising
a
new
<stop>

Residual connection

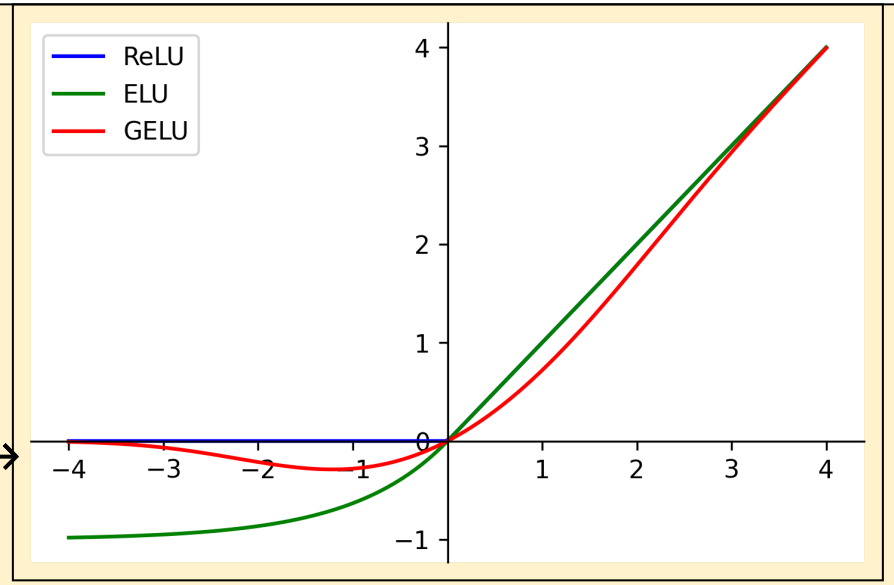
Layer norm

dropout

$o_1 = \mathbb{R}^{768}$

$l = 3042$

$z_1 = \mathbb{R}^{768}$




Feed Forward Neural Network

Masked Multi-head attention

<go> at the bell labs hamming ..... bound ..... devising a new <stop>

# Number of parameters

token Embeddings: $|\mathcal{V}| \times$ embedding_dim

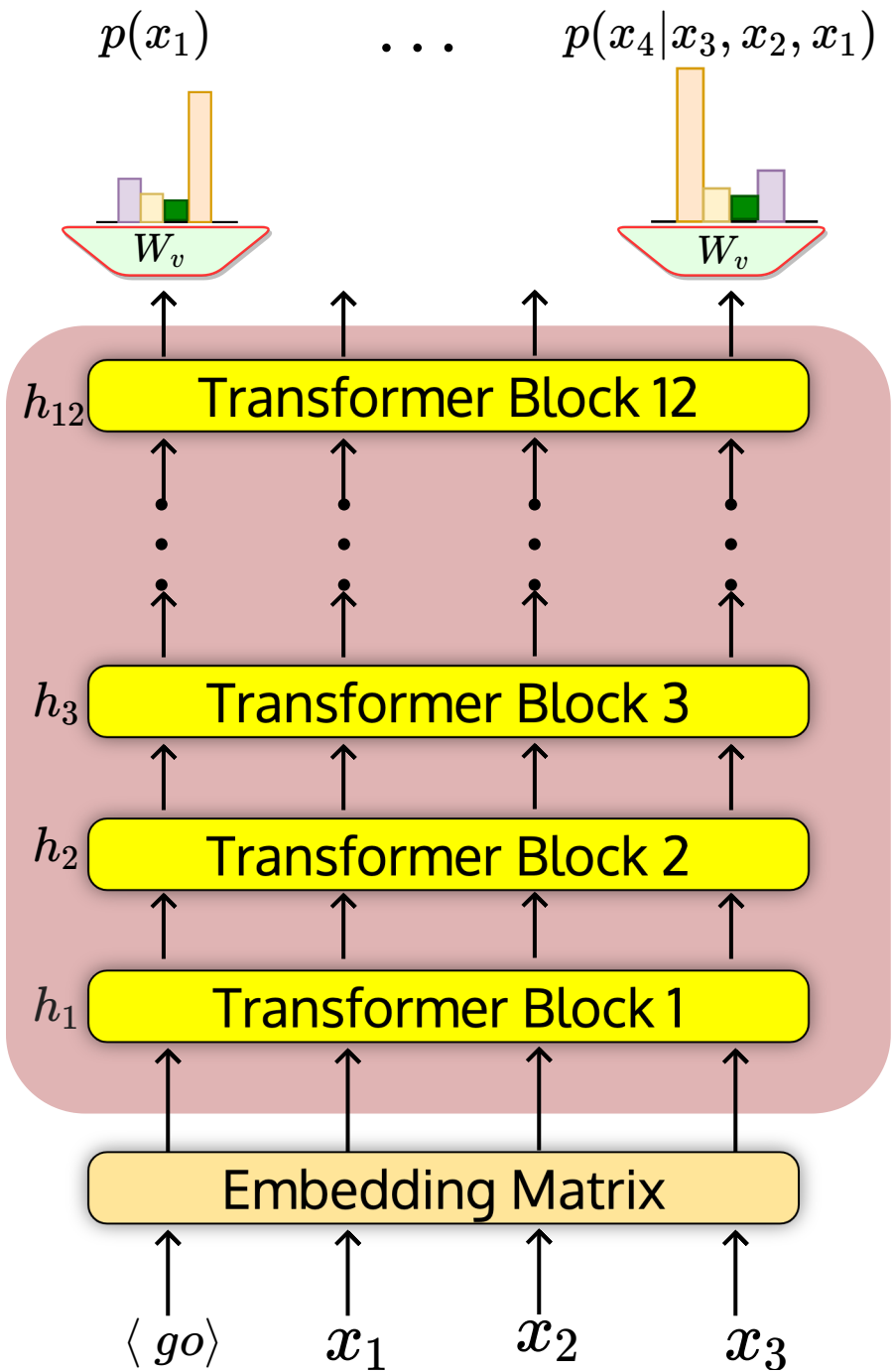
$$40478 \times 768 \quad = 31 \times 10^6 \quad = 31M$$

Position Embeddings : context length $\times$ embedding_dim

$$512 \times 768 \quad = 0.3 \times 10^6 \quad = 0.3M$$

Total: $31.3M$

The positional embeddings are also learned, unlike the original transformer which uses fixed sinusoidal embeddings to encode the positions.



# Number of parameters

Attention parameters per block

$$W_Q = W_K = W_v = (768 \times 64)$$

Per attention head

$$3 \times (768 \times 64) \approx 147 \times 10^3$$

For 12 heads

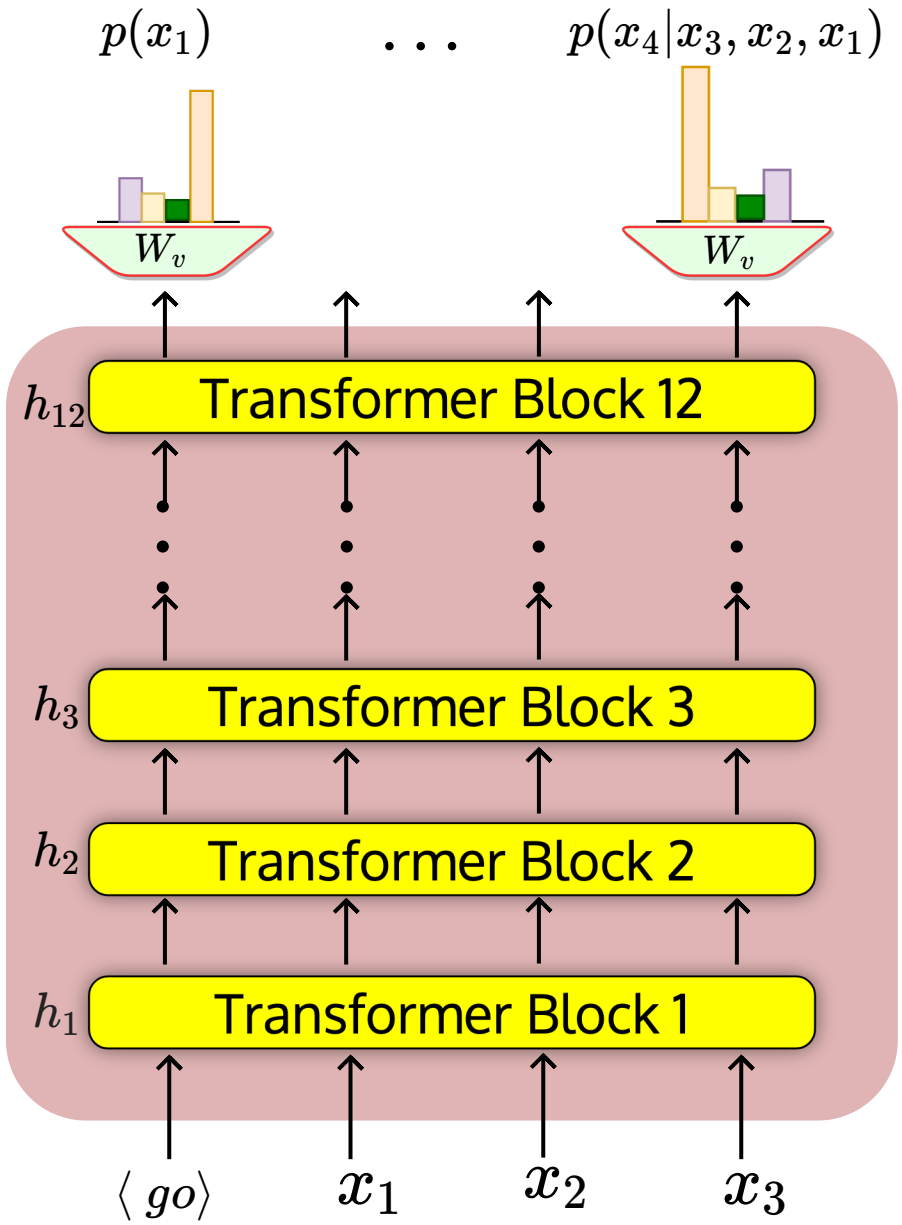
$$12 \times 147 \times 10^3 \approx 1.7M$$

For a Linear layer:

$$768 \times 768 \approx 0.6M$$

For all 12 blocks

$$12 \times 2.3 = 27.6M$$



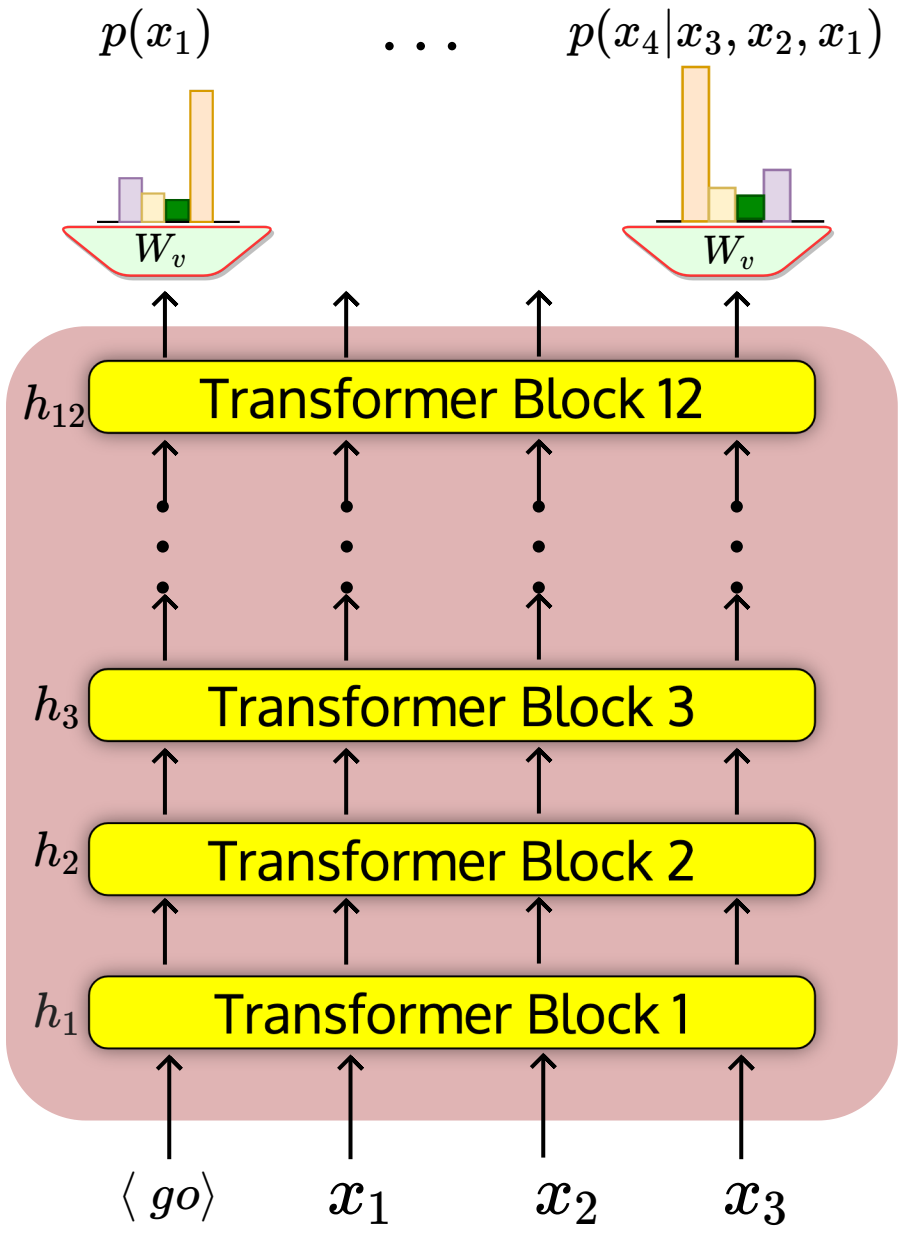
# Number of parameters

FFN parameters per block

$$2 \times (768 \times 3072) + 3072 + 768$$
$$= 4.7 \times 10^6 = 4.7M$$

For all 12 blocks

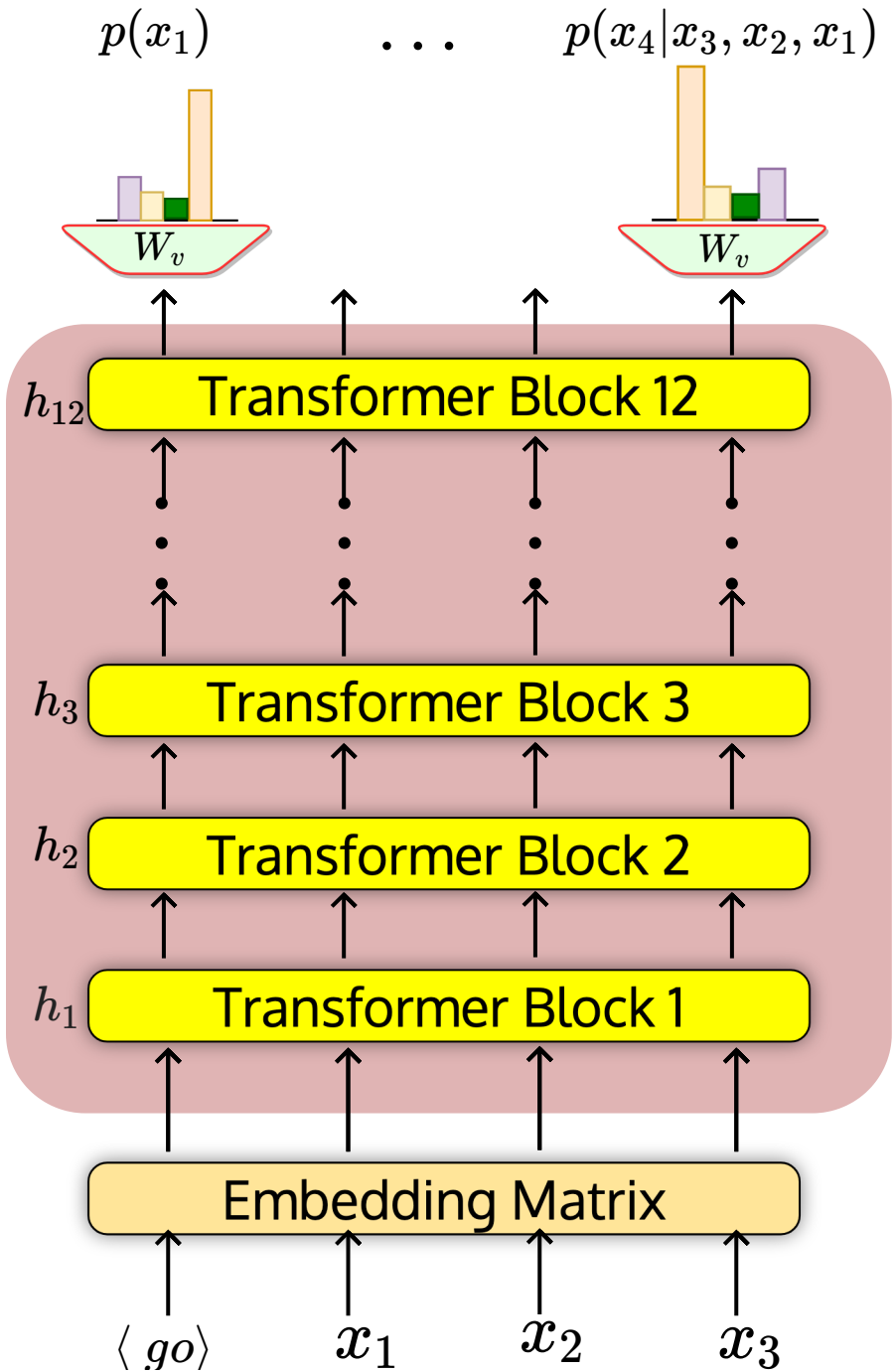
$$12 \times 4.7 = 56.4M$$



# Number of parameters

| Layer | Parameters (in Millions) |
|---|---|
| Embedding Layer | $31.3$ |
| Attention layers | $27.6$ |
| FFN Layers | $56.4$ |
| Total | $116.461056^*$ |

Thus, GPT-1 has around 117 million parameters.




*Without rounding the number of parameters in each layer

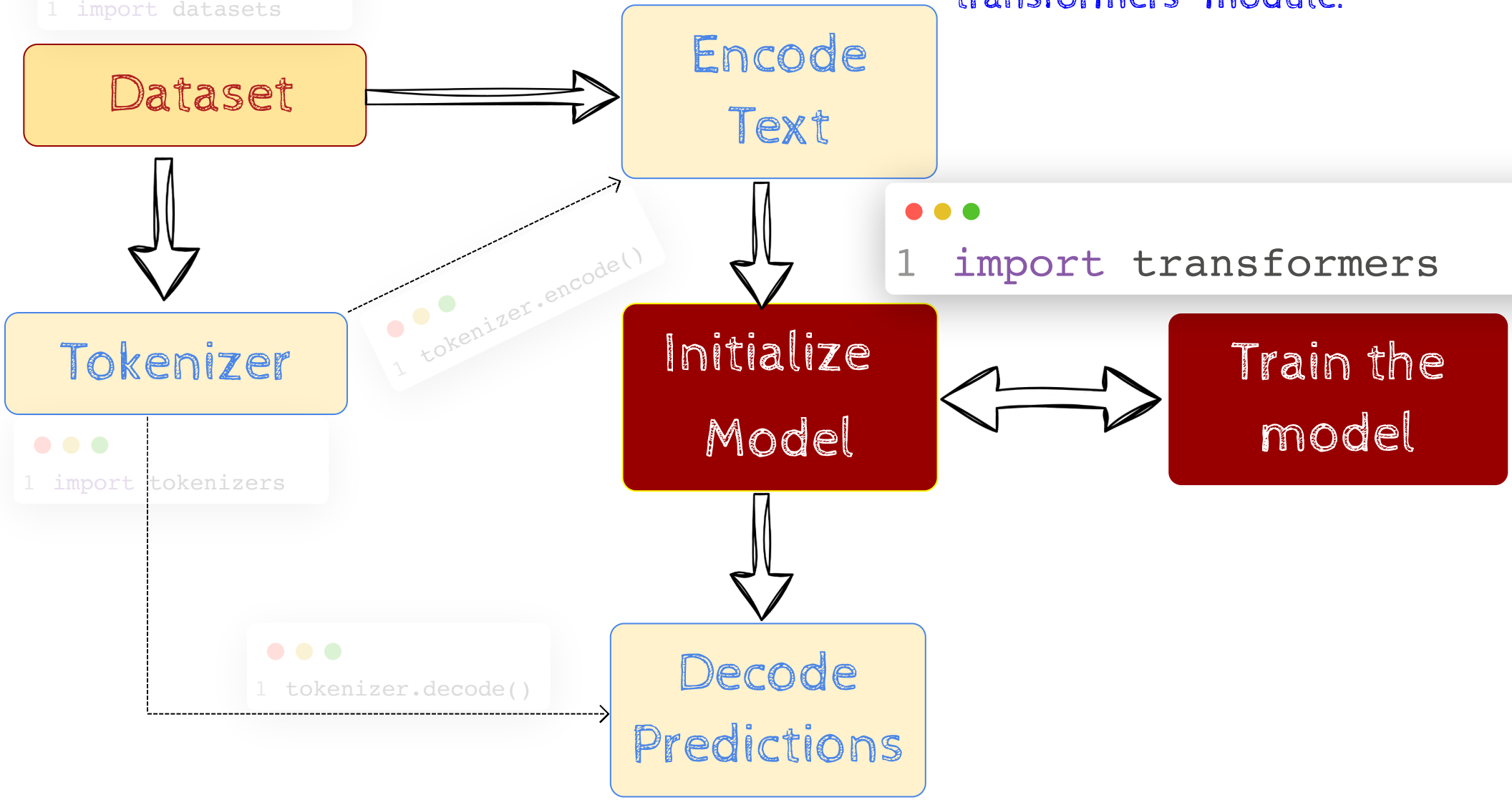
# Module 3.3 : HF Transformers

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

# In this week

we will learn how to train transformer based models using the HF "transformers" module.



Recall that we can create any architecture with a few lines of code in PyTorch

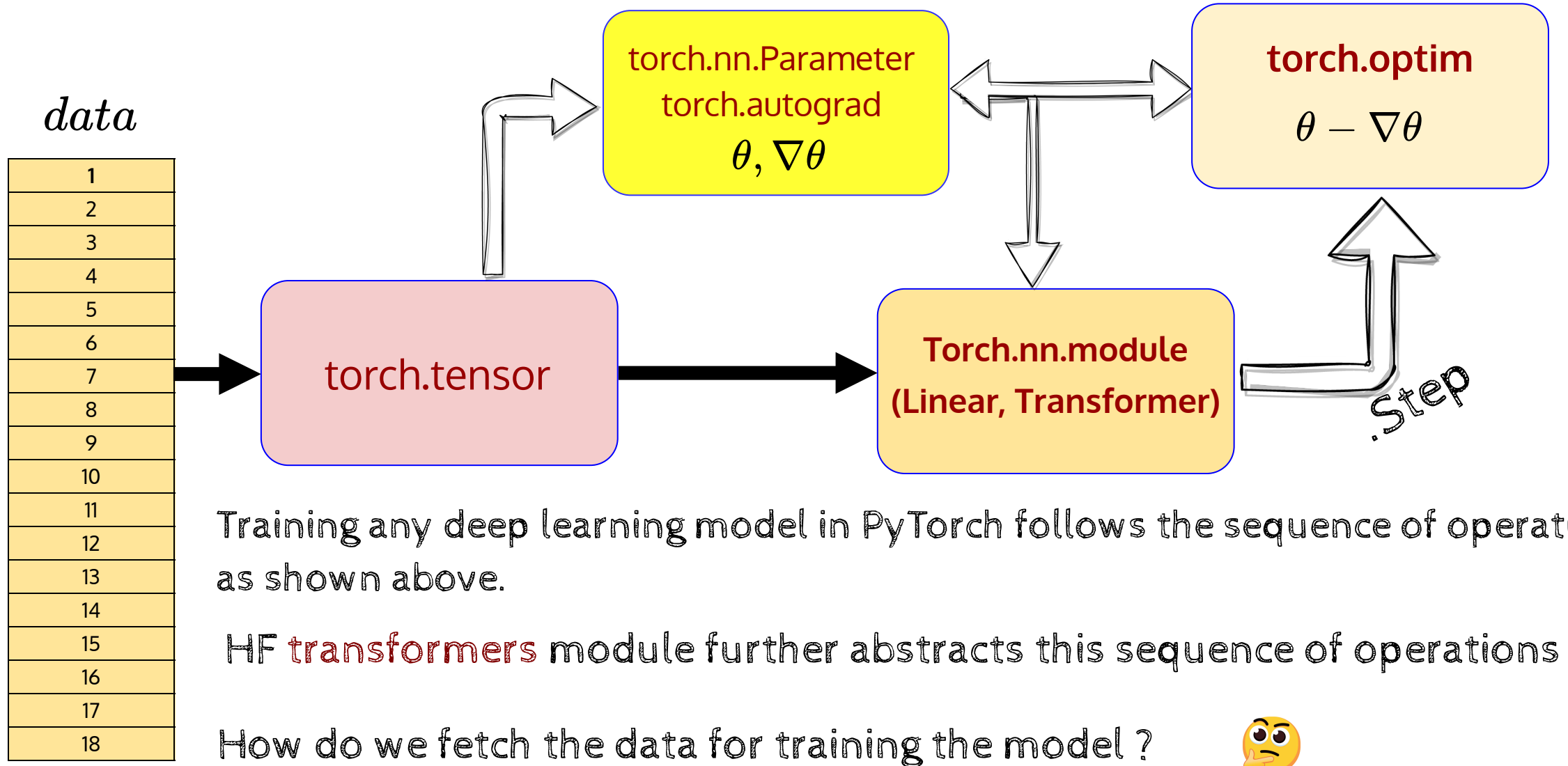
```
class CNN(nn.Module):
    def __init__(self):
        super(Ne
        self.con
        self.poo
        self.con
        self.fc1
        self.fc2
        self.fc3

    def forward(

        x = self
        x = self
        x = x.vi
        x = F.re
        x = F.re
        x = self
        return x
```

```
class RNN(torch.nn.Module):
    def __init__(self,vocab_size,embed_dim,hidden_dim,num_class):
        super()
        self.e



        self.r



        self.f

    def forwar
        x = se
        x = pa



        x = se
        x = se
        return
```

```
class TRANSFORMER(torch.nn.Module):
    def __init__(self,vocab_size,embed_dim,hidden_dim,num_class):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      embed_dim,
                                      padding_idx=0)
        self.transformer = nn.Transformer(dmodel,
                          nhead,
                          num_encoder_layers,
                          num_decoder_layers,
                          dim_feedforward)
        self.fc = Linear(dmodel, vocab_size)

    def forward(self, x,length)
        x = self.embedding(x)
        x = self.transformer(x)
        x = self.fc(x[1])
        return x
```

We can also set up the entire training pipeline in PyTorch

# HF with PyTorch Backend

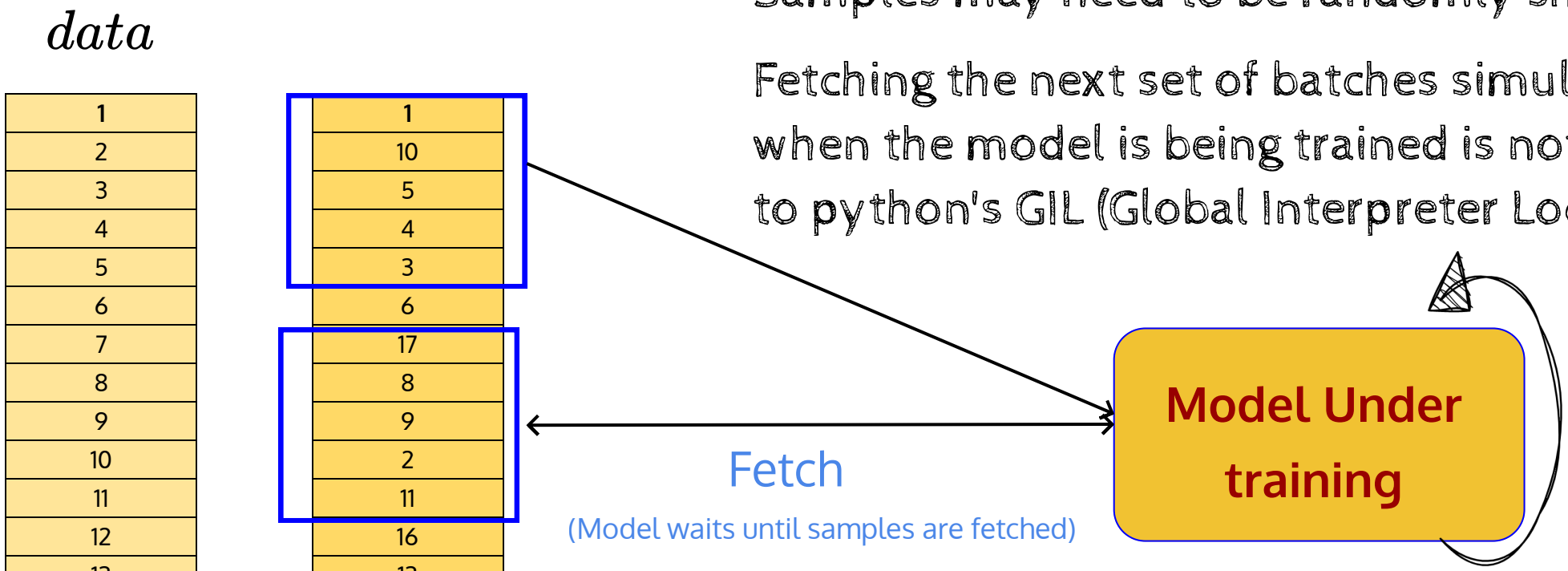



Training any deep learning model in PyTorch follows the sequence of operations as shown above.

HF transformers module further abstracts this sequence of operations

How do we fetch the data for training the model ? 🤔

We can slice the batch and iterate, not a big deal! wait..




Samples may need to be randomly shuffled

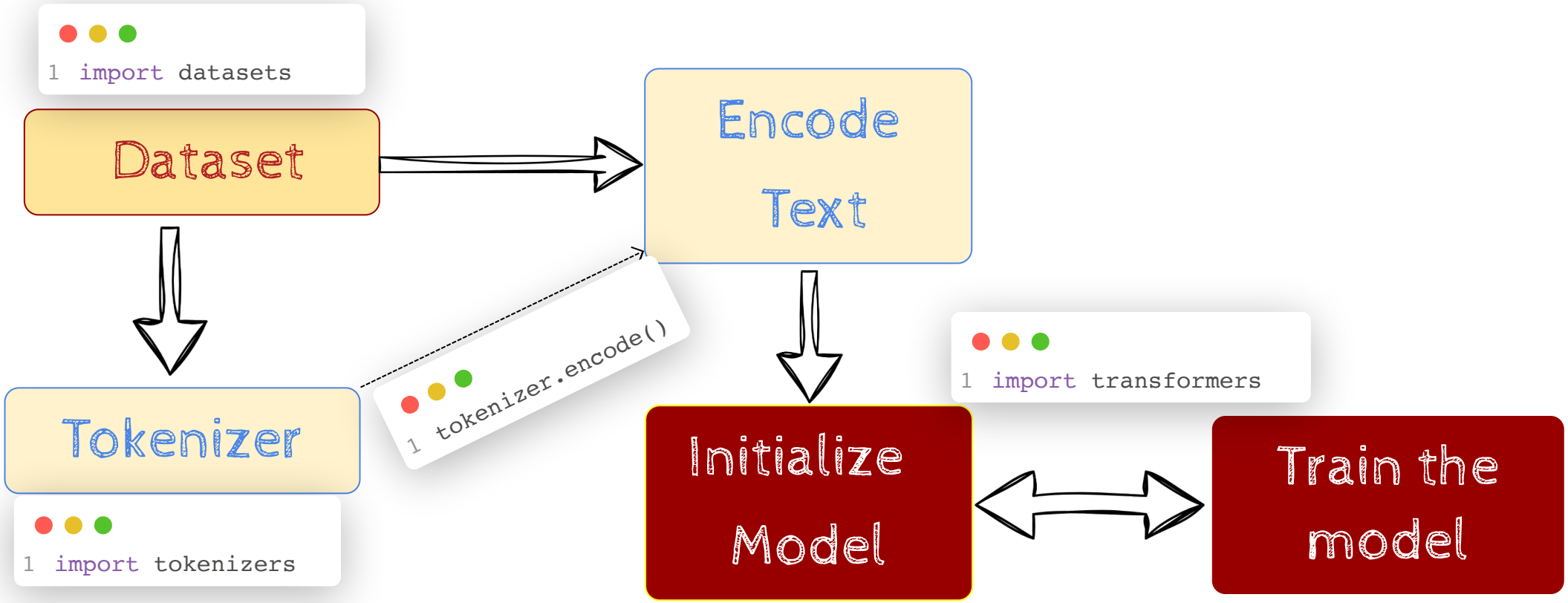
Fetching the next set of batches simultaneously when the model is being trained is not possible due to python's GIL (Global Interpreter Lock)

The complexity of distributing data to multiple GPUs in distributed training is high.

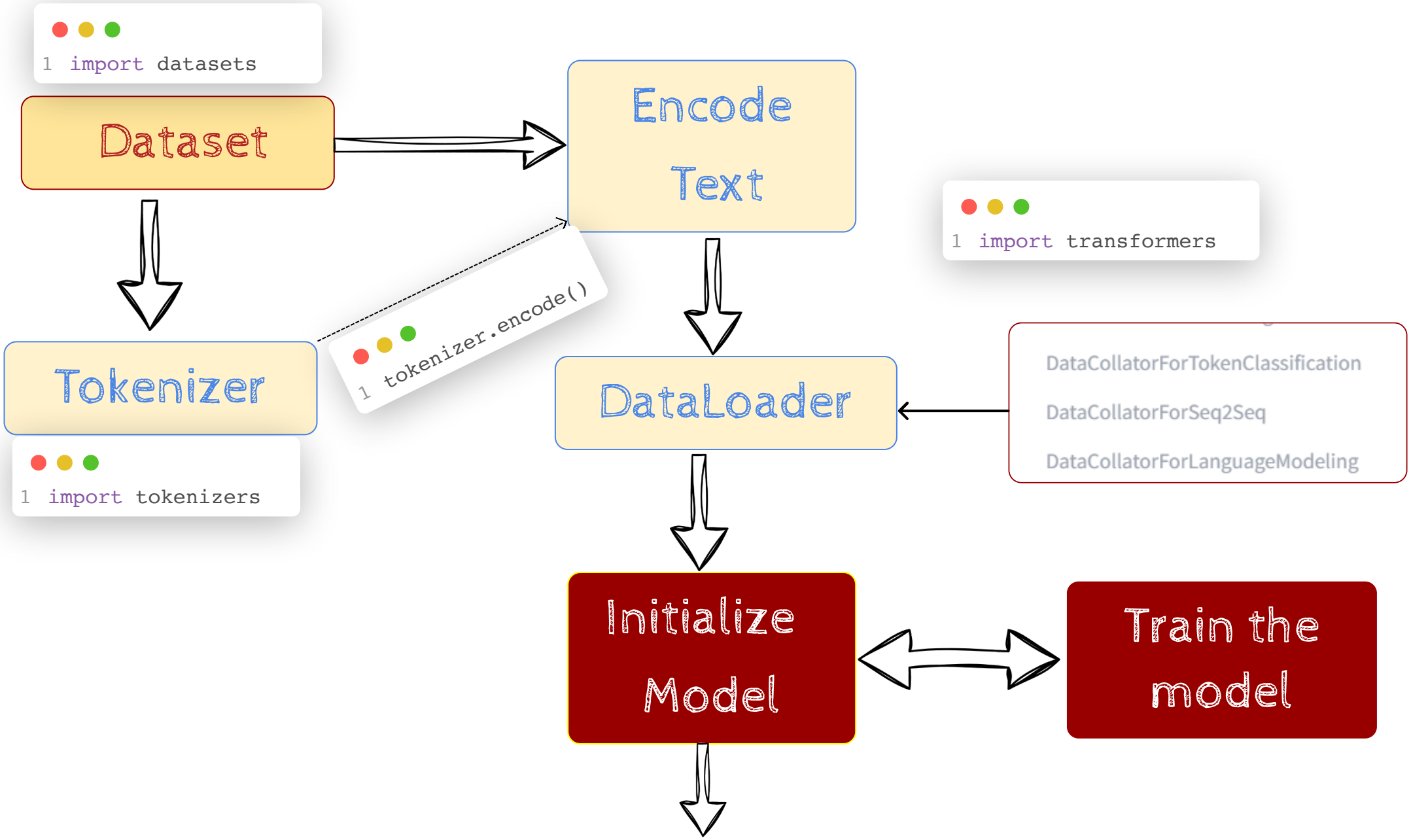
The complexity becomes even greater in a multinode distributed training setup

Therefore we need a mechanism that efficiently handles all these requirements

# Current Training Setup

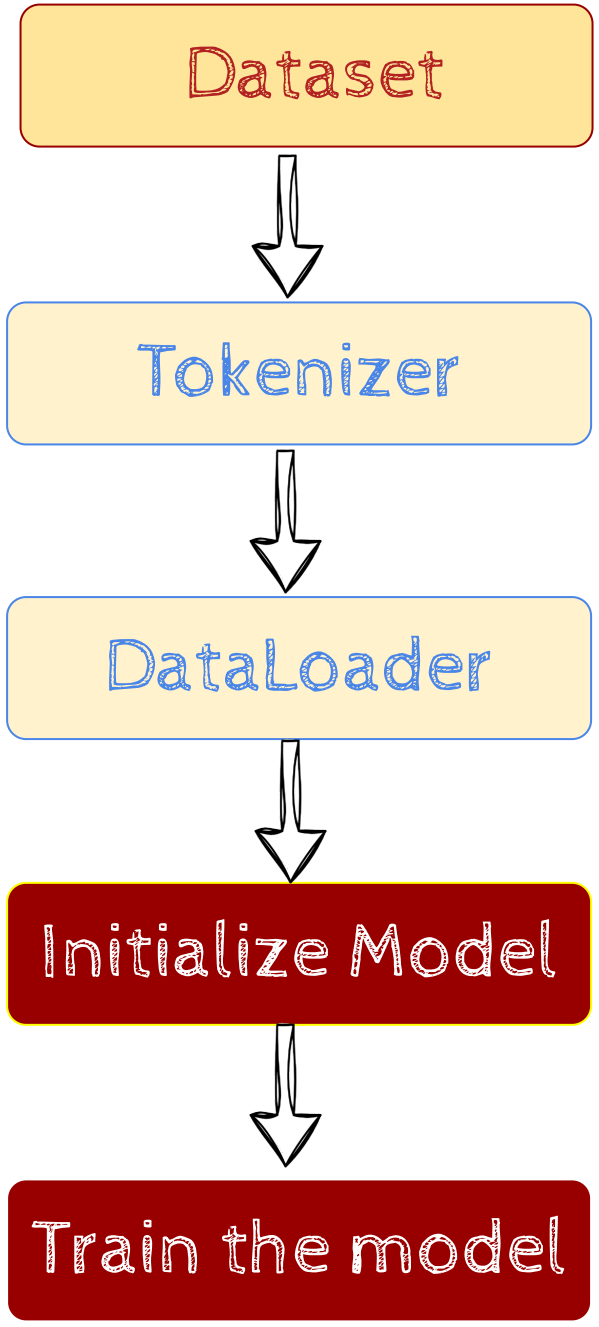


# Training with DataLoader



# The Main Sections of the Hands-on Notebook

Let's dive in




```python
from datasets import load_dataset
```

```python
from transformers import AutoTokenizer
```

```python
from transformers import DataCollatorForLanguageModeling
```

```python
from transformers import GPT2Config, GPT2LMHeadModel
```

```python
from transformers import TrainingArguments, Trainer
```