Deep Learning Practice - NLP

Recap of NN models, Roadmap, Datasets

Mitesh M. Khapra

AI4Bharat, Department of Computer Science and Engineering, IIT Madras

Module 1 : Natural Language Processing

Mitesh M. Khapra

AI4Bharat, Department of Data Science and Artificial Intelligence, IIT Madras

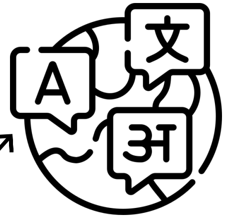
What is it?

enabling computers to understand, interpret, and generate human language

So that, given an input text, we can ask the computer to

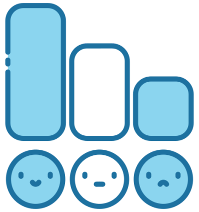
I was watching a movie with my friends at INOX, Chennai. The movie was stunning!

Translate



சென்னை INOXல் நண்பர்களுடன் படம் பார்த்துக்கொண்டிருந்தேன். படம் பிரமிக்க வைத்தது!

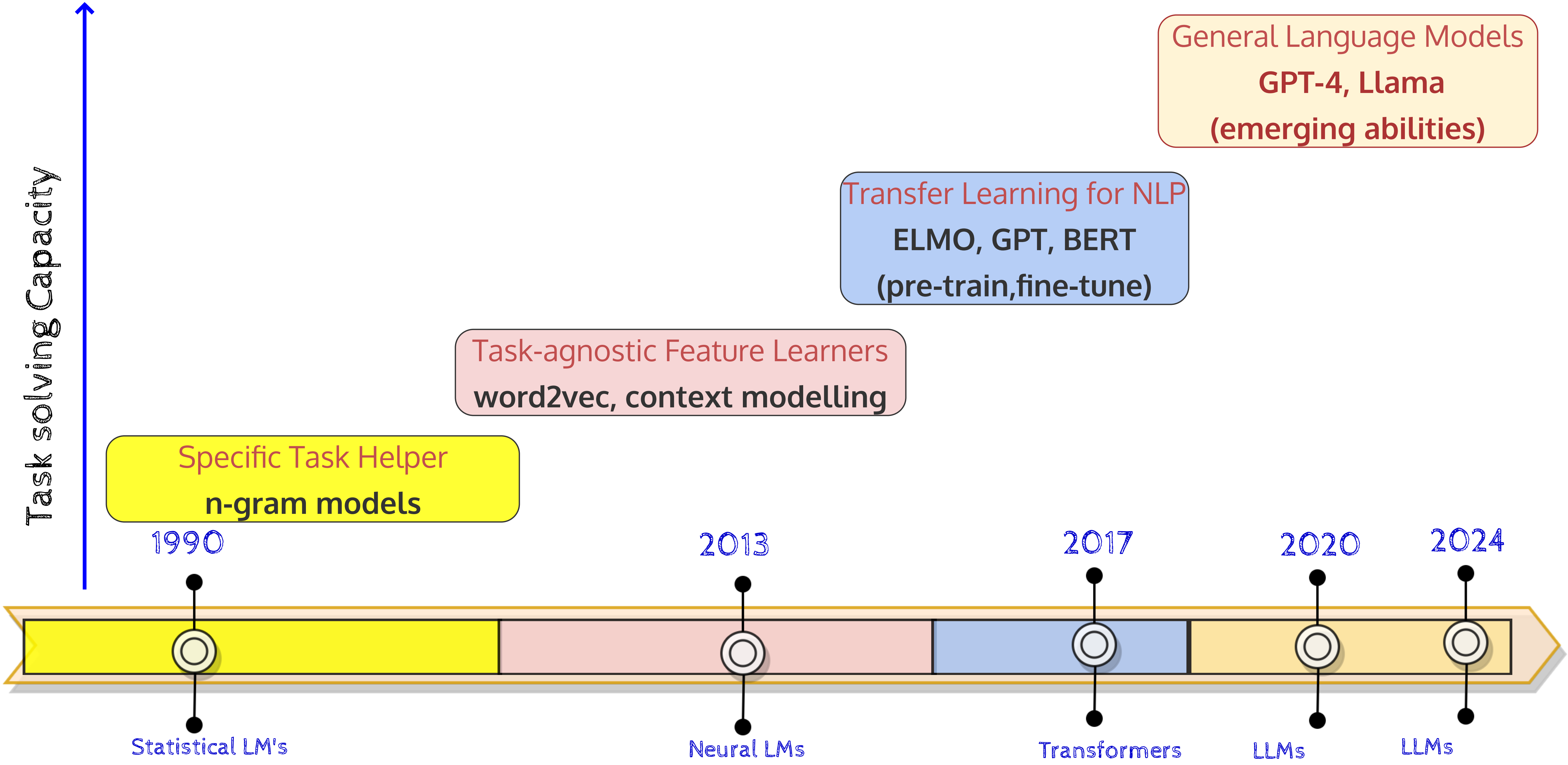
Sentiment



NER

I was watching a movie with my friends at INOX[ORG], Chennai[LOC]. The movie was stunning!

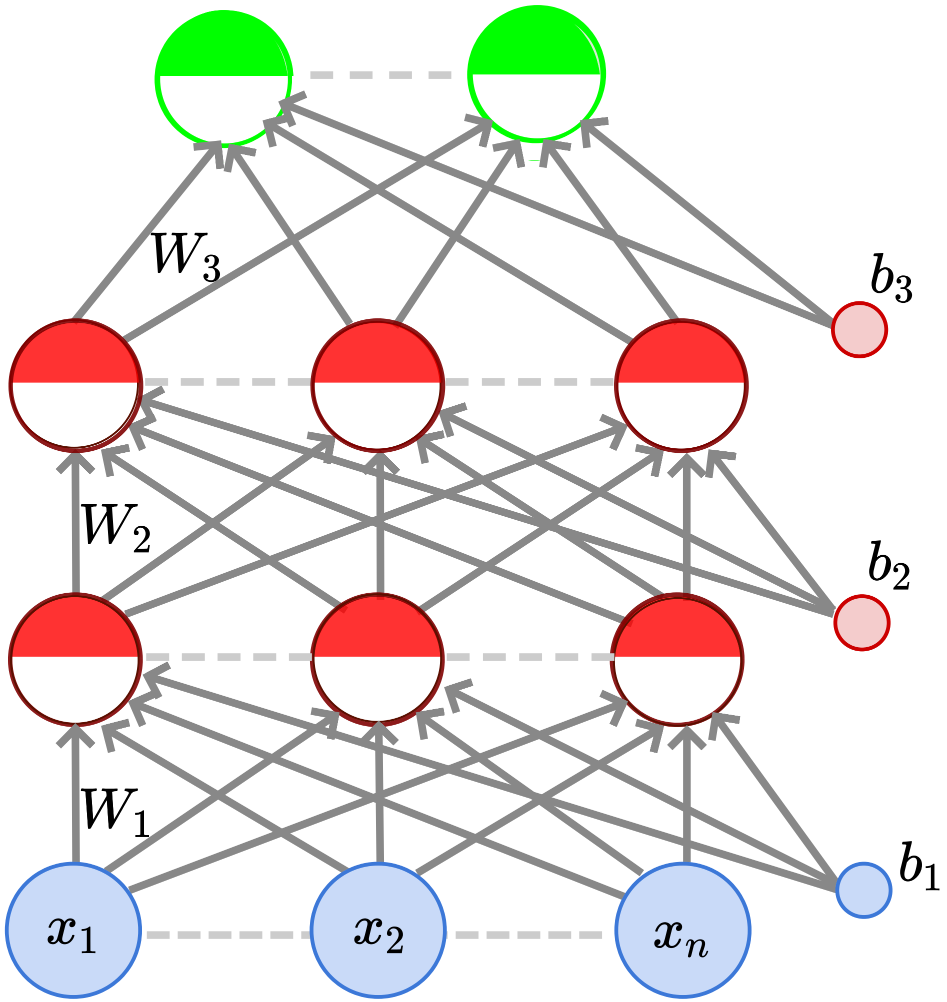
What are the methods?



Reference: LLM Survey

Neural Network Models

MLP

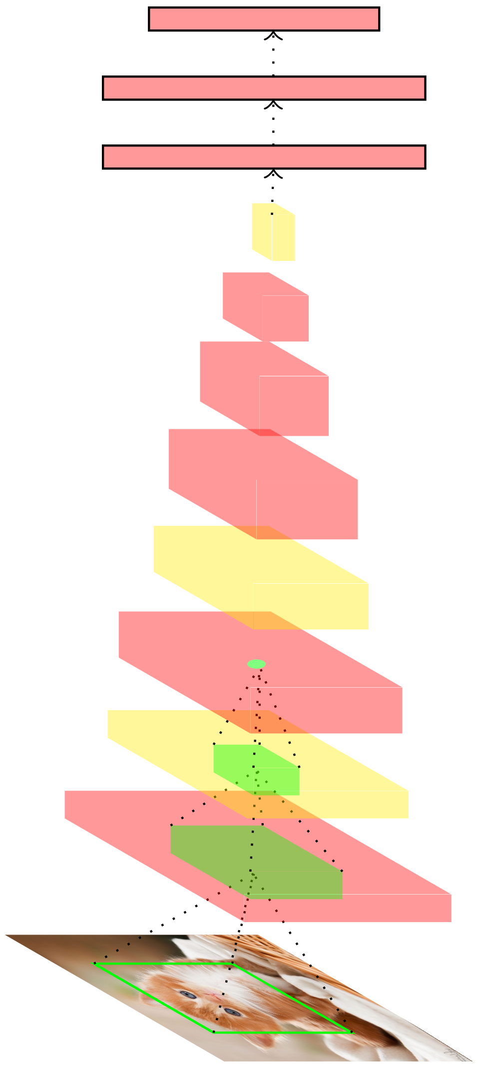


Let's quickly recap the four types of architectures that we learned in the deep learning theory course

We started with a simple Multi-Layer Perceptron (MLP) and the backpropagation algorithm for learning the parameters

It is a feedforward fully connected neural network

CNN

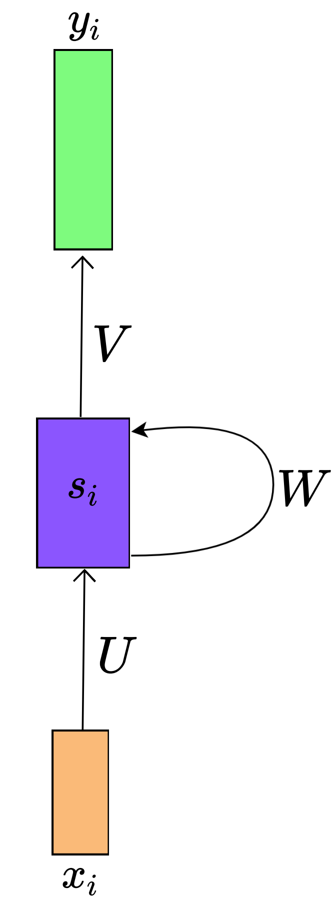


It is also a feedforward neural network commonly used in the domain of computer vision

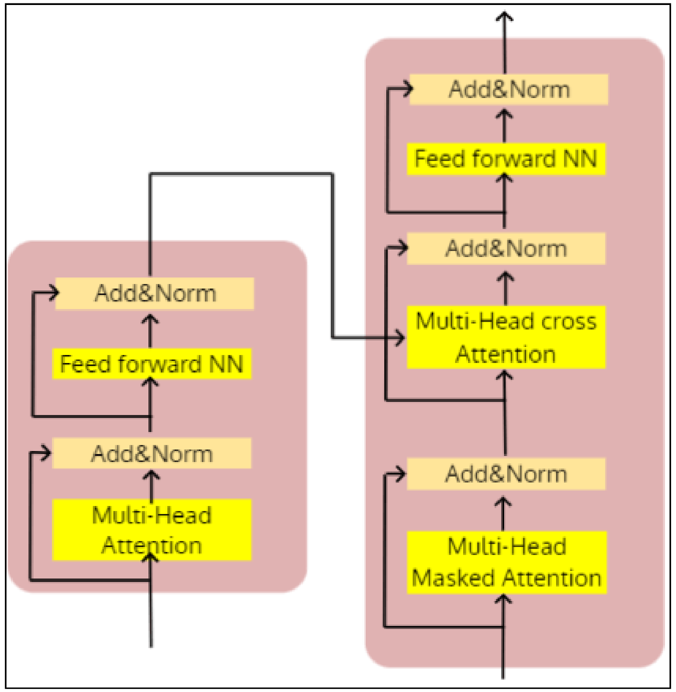
We introduced the weight sharing concept where the weights in the kernels are shared across the input.

This idea of weight sharing is prevalent in almost all modern neural networks

RNN



Transformer

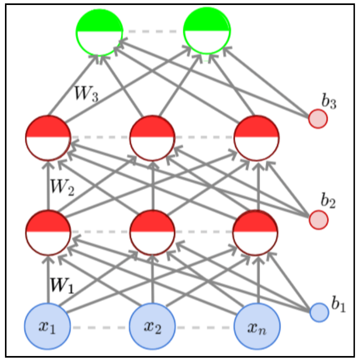


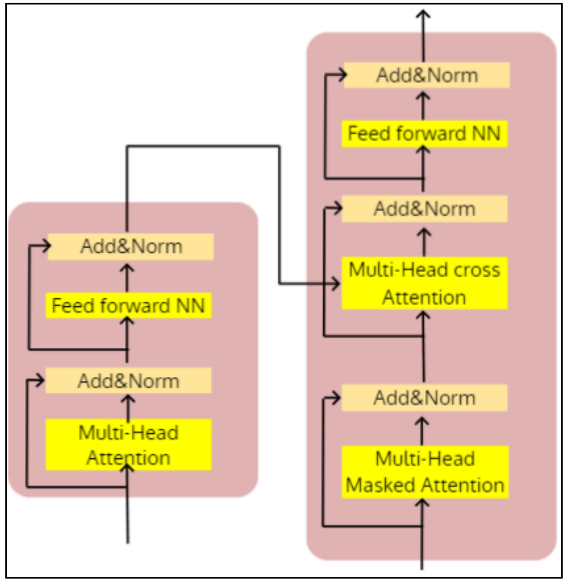
Recurrent Networks such as RNN, LSTM and transformers use the same weights for all elements/timesteps in the input sequence

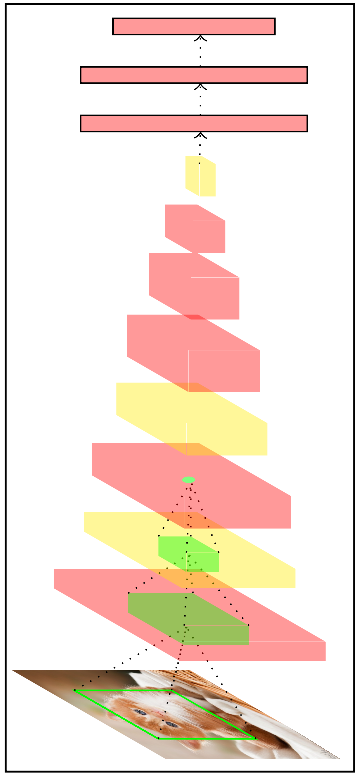
Therefore, they are better suited for Natural Language Processing tasks, as weight sharing enables the models to generalize to sequences of any length

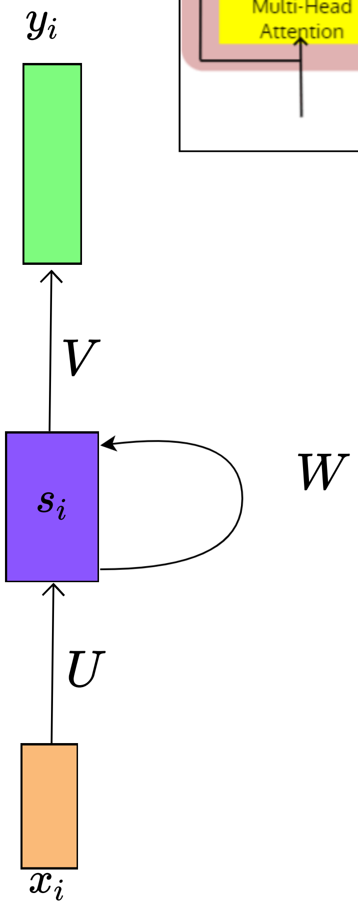
However, training of transformers can be parallelized as opposed to RNNs and LSTMs

This gives the transformer an edge over recurrent neural networks









Today, we can quickly build any of these architectures using modern deep learning frameworks such as Pytorch and TensorFlow

In this course, we will use Pytorch or any framework built on top of that

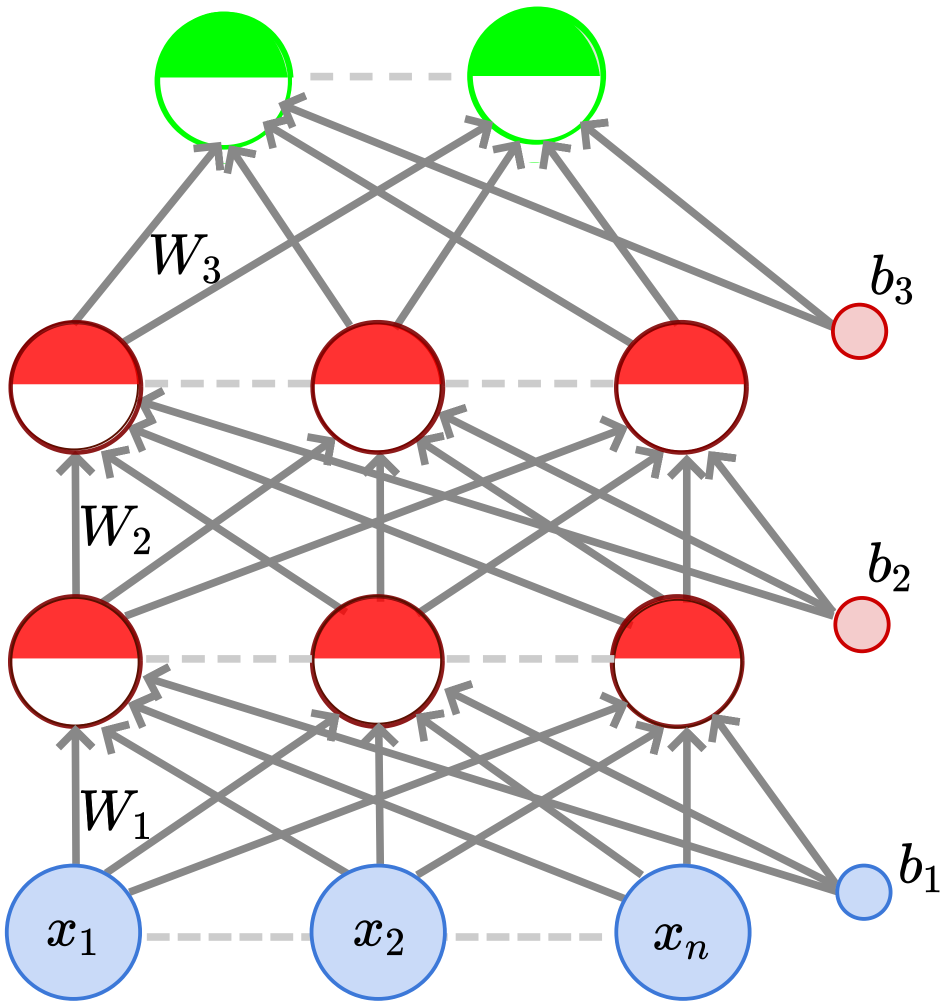
All architectures contain a sequence of learnable layers (linear, convolution, rnn, embedding ...)

Every learnable layer can be derived from the `torch.nn.Module` in PyTorch

Therefore, any architecture can be composed with the `torch.nn.Module`

let's build all four architectures with a few lines of code

MLP



Implementation in Pytorch

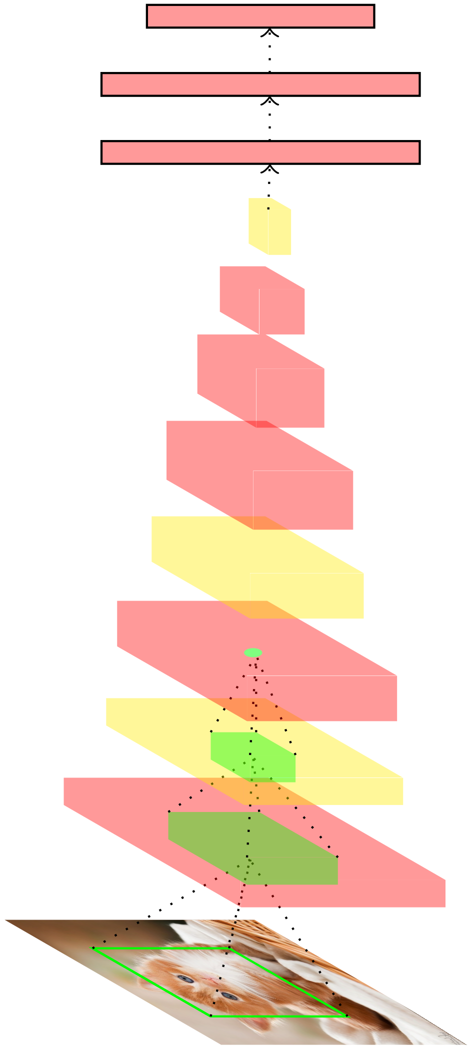
The model is composed of linear layers and non-linear activations (assuming Relu). Therefore, the code contains only nn.linear and torch.relu

```
class MLP(nn.Module):

  def __init__(self,x_dim,h1_dim,h2_dim,out_dim):
    super().__init__()
    self.w1 = nn.Linear(x_dim,h1_dim,bias=True)
    self.w2 = nn.Linear(h1_dim,h2_dim,bias=True)
    self.w3 = nn.Linear(h2_dim,out_dim,bias=True)

  def forward(self,x):
    out = torch.relu(self.w1(x))
    out = torch.relu(self.w2(out))
    out = torch.relu(self.w3(out))
    return out
```

CNN



Implementation in Pytorch

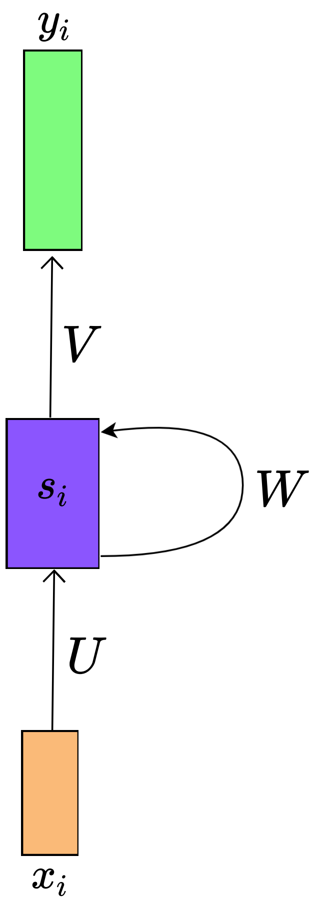
The model is composed of nn.linear,nn.Conv2d, MaxPool2D and torch.relu

```
class CNN(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool  = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 10)
        self.fc2 = nn.Linear(10, 4)
        self.fc3 = nn.Linear(4, 2)

    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

RNN



Implementation in Pytorch

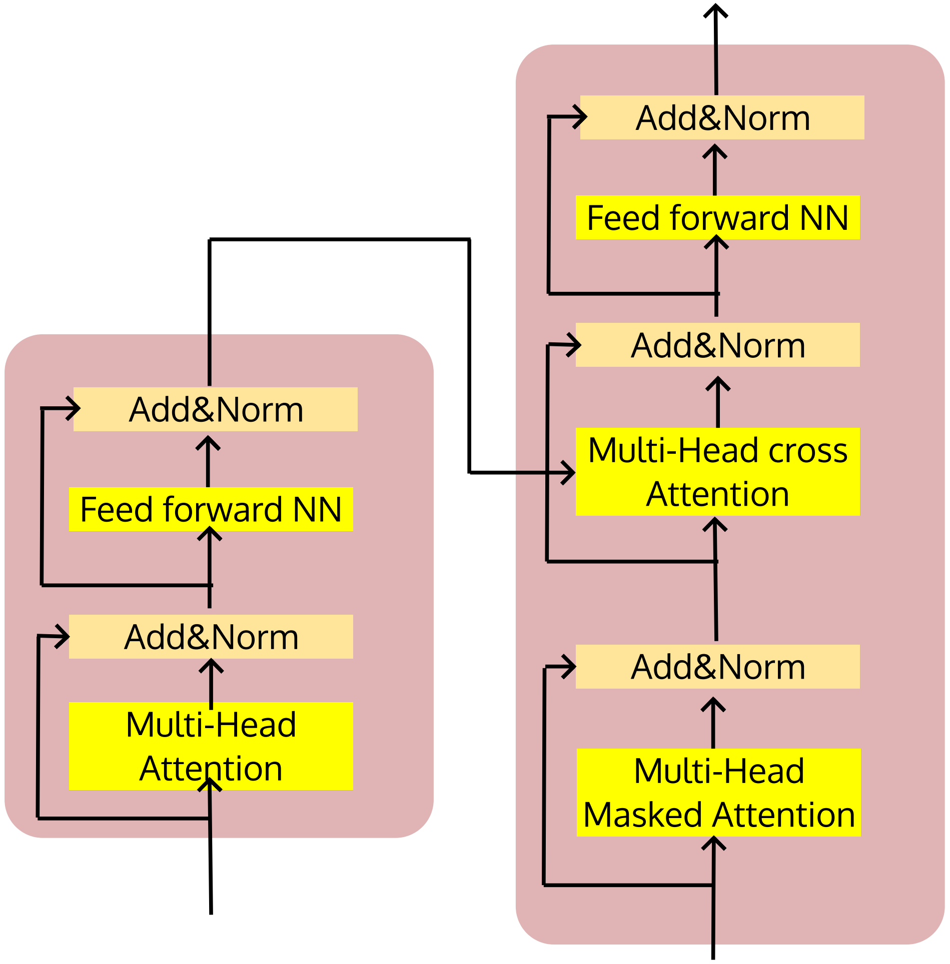
The model is composed of nn.linear,nn.Embedding, nn.RNN

nn.RNN is composed of nn.linear, torch.relu and other required modules

```
class RNN(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.rnn = nn.RNN(embed_dim,
                          hidden_dim,
                          batch_first=True)
        self.fc = nn.Linear(hidden_dim, num_class)

    def forward(self, x,length)
        x = self.embedding(x)
        x = pack_padded_sequence(x,
                                 lengths=length,
                                 enforce_sorted=False,
                                 batch_first=True)
        x = self.rnn(x)
        x = self.fc(x[-1])
        return x
```

Transformer



Implementation in Pytorch

The model is composed of nn.Embedding, nn.Transformer, nn.Linear

nn.Transformer is composed of nn.MultiHeadAttention and other required modules

```
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
```

```python
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
```

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


        self.rı


        self.fc

    def forwarc
        x = sel
        x = pac



        x = sel
        x = sel
        return
```

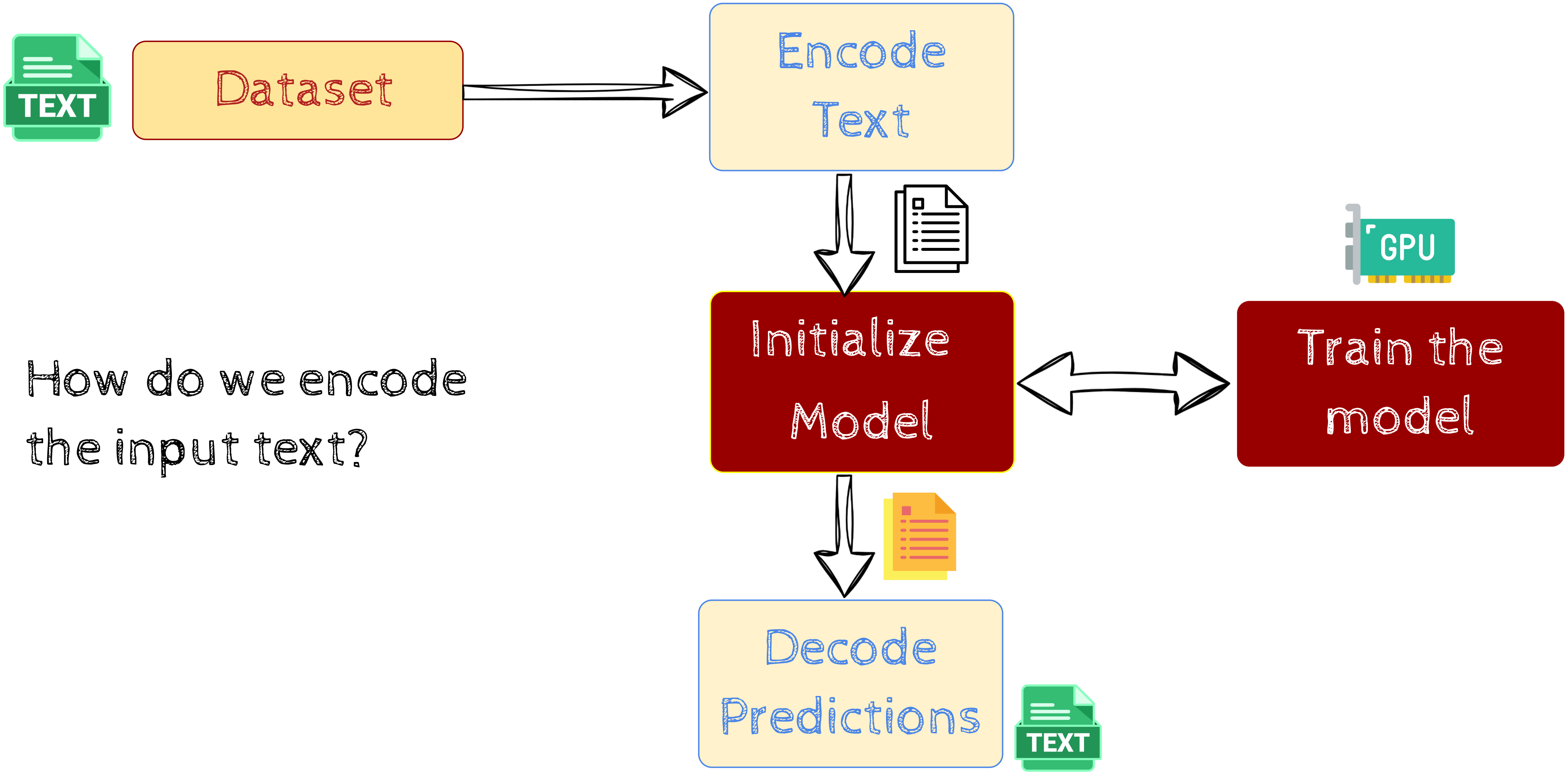
```python
class TRANSFORMER(torch.nn.Module):
    def __init__(self,vocab_size,embed_dim,hidden_dim,num_class):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
                                      embed_dim,
                                      padding_idx=0)
        self.transformer = 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 create any architecture with a few lines of code. However, this is just a part of the entire training setup.

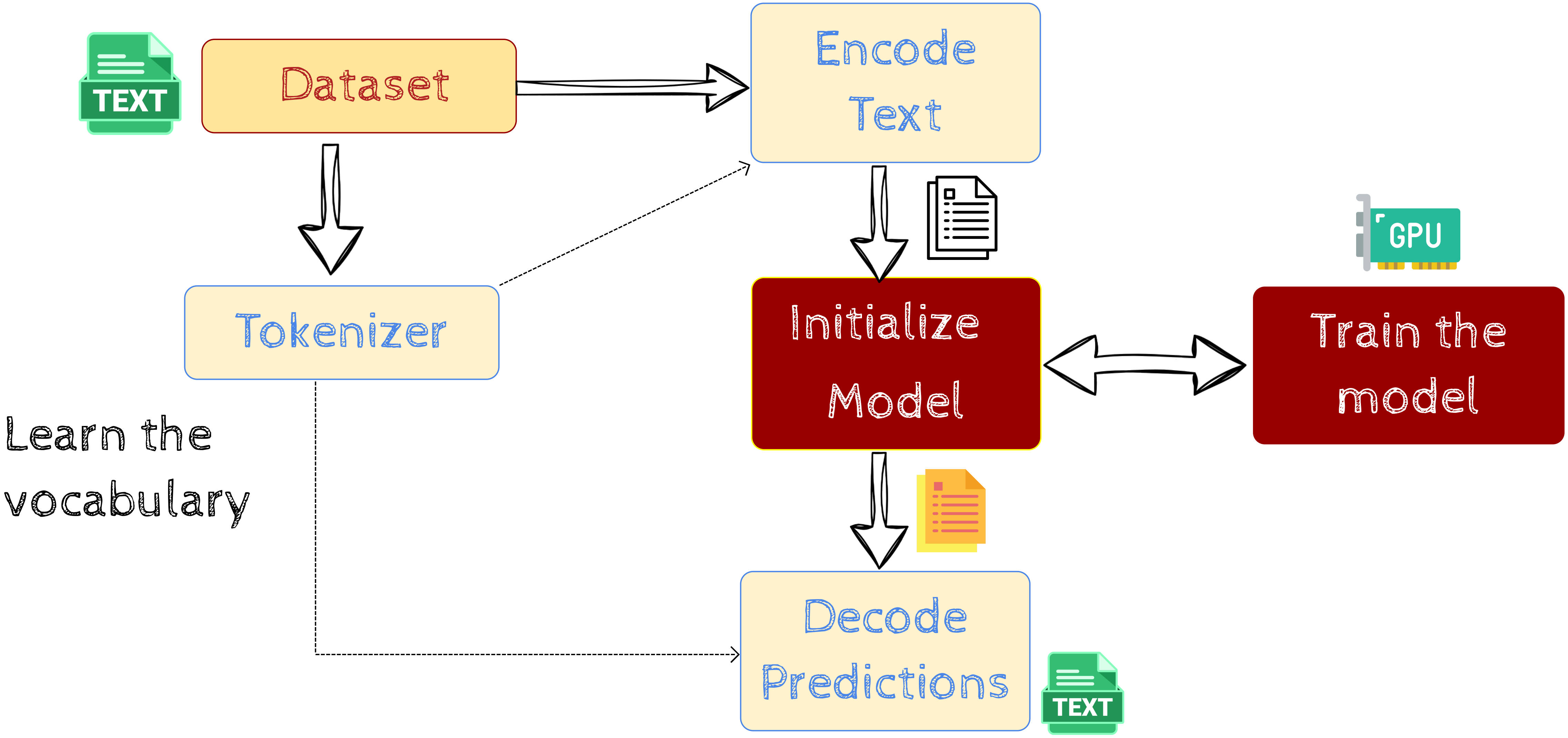
The training setup contains many other components. Let's see

Experimental Setup

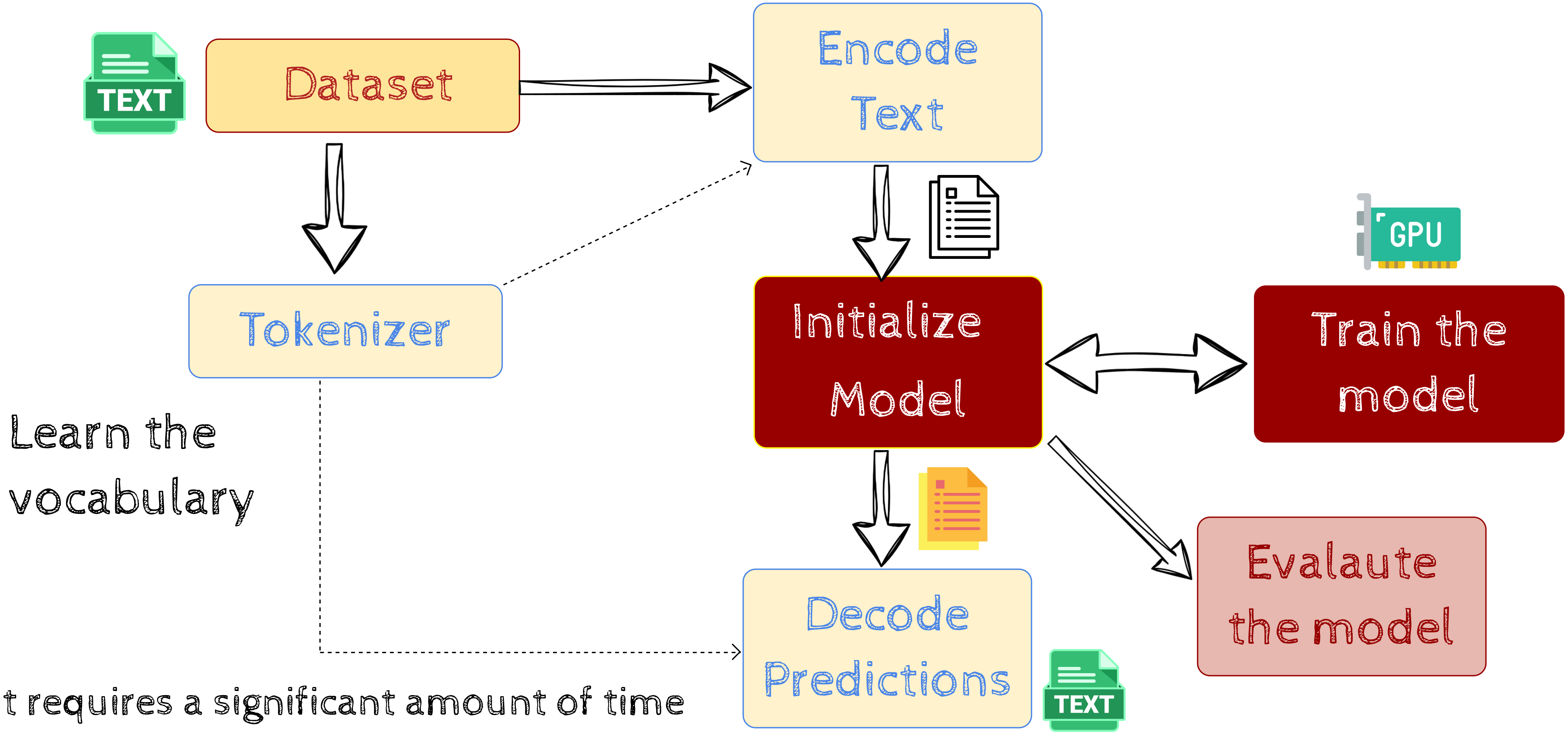


How do we encode the input text?

Experimental Setup

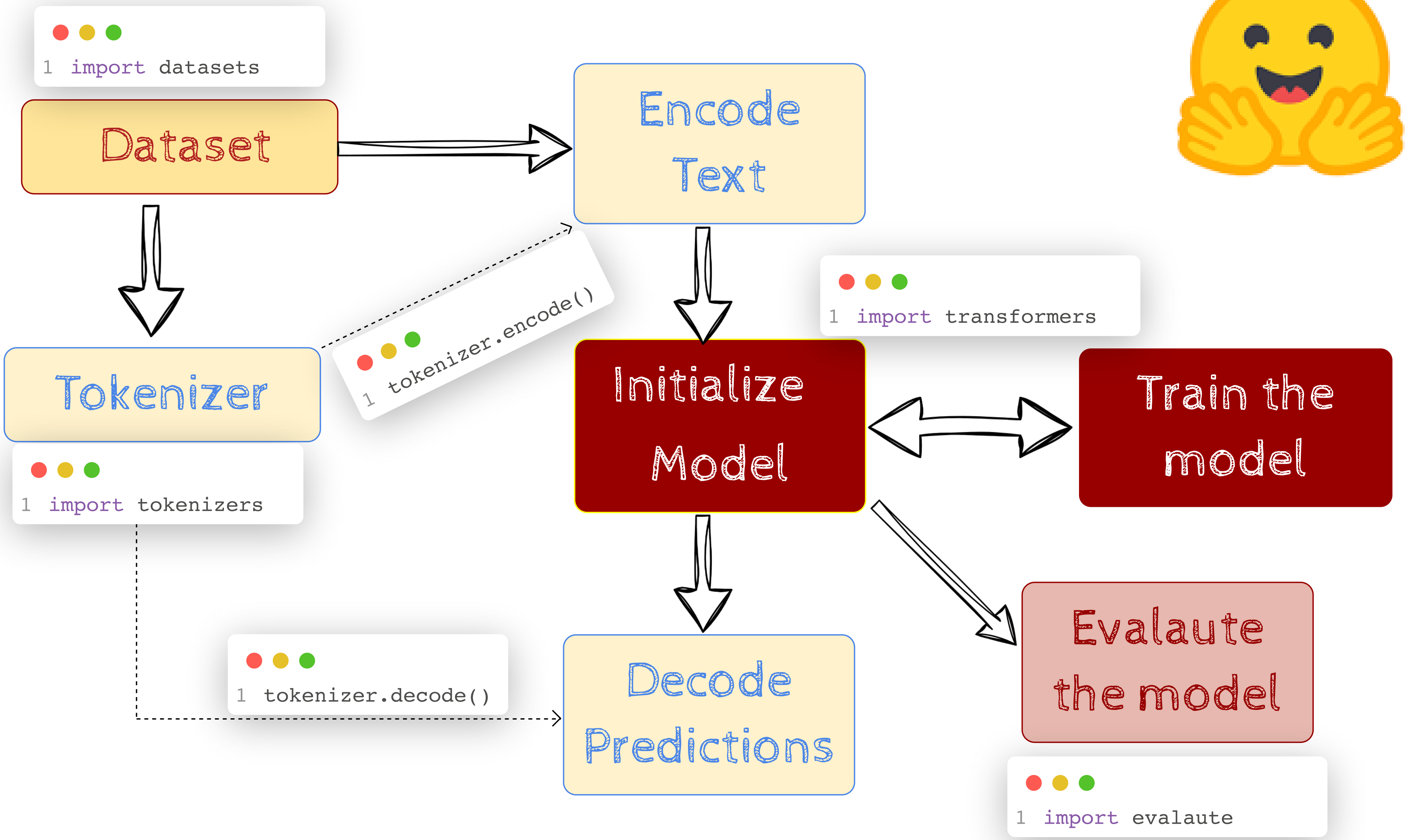


Experimental Setup

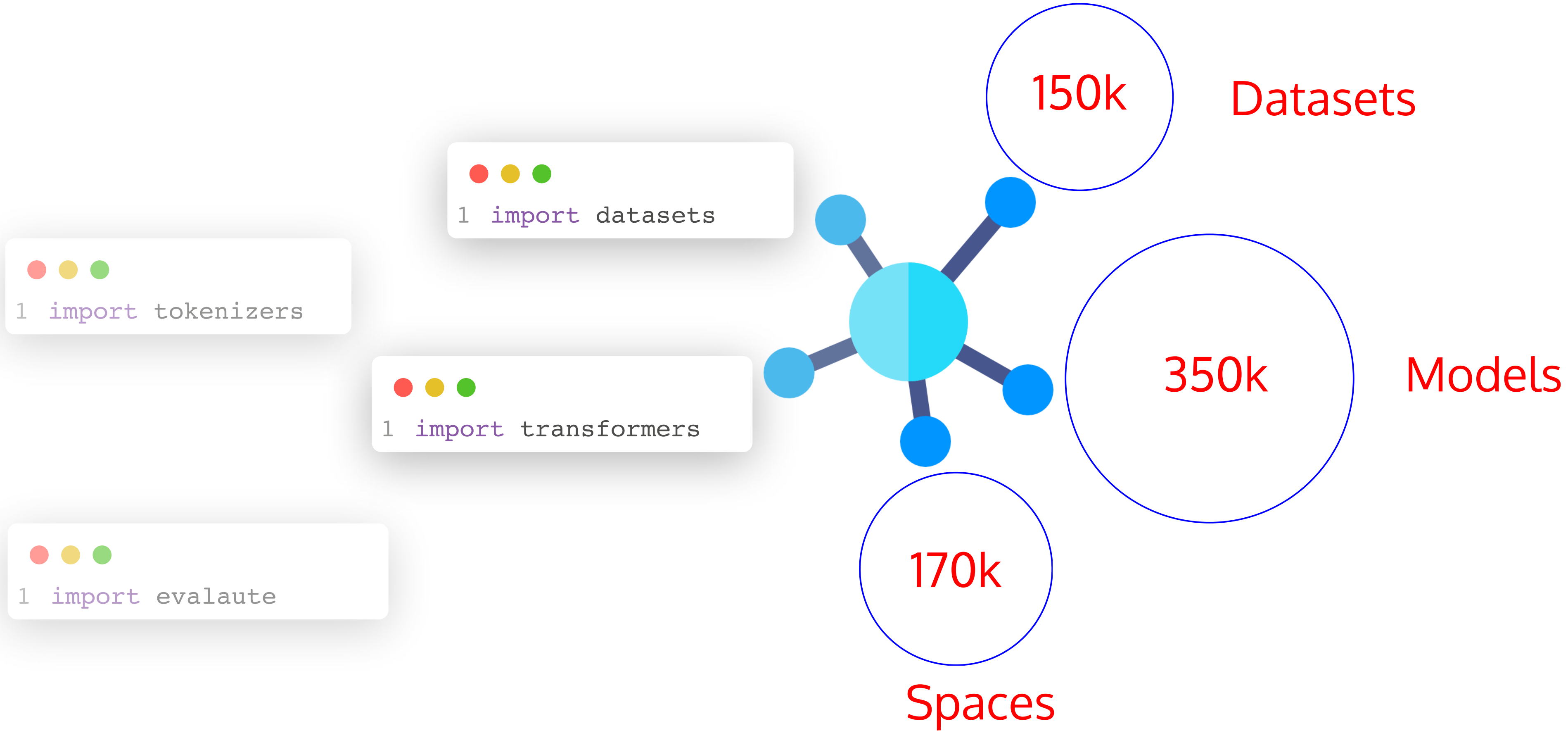


It requires a significant amount of time and effort to set this up properly

Hugging Face Does the Heavy Lifting



Hugging Face Hub



We have access to 150k datasets and 350k models via hugging face hub

Rich Ecosystem

```python
import datasets
```

```python
import tokenizers
```

```python
import transformers
```

```python
import evalaute
```

```python
from  accelarate ..
```

```python
from  optimum ..
```

```python
from  peft ..
```

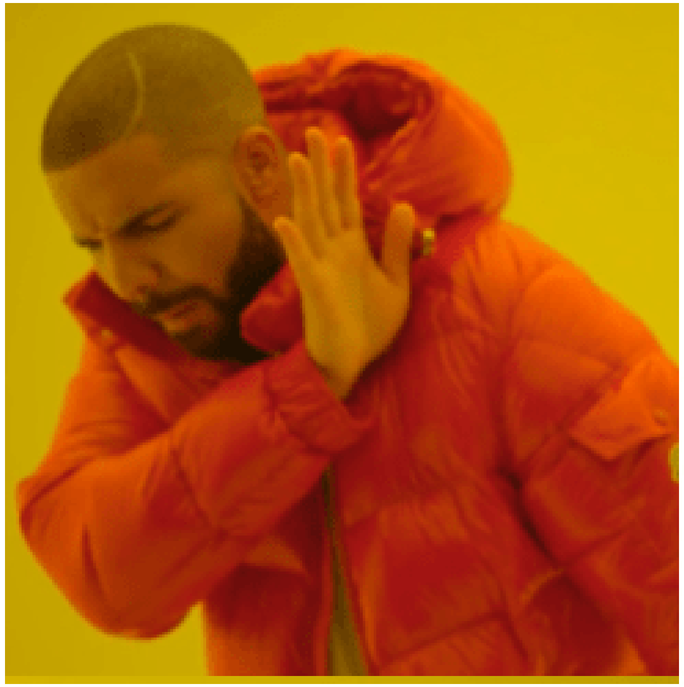
```python
import bitsandbytes
```

⋮

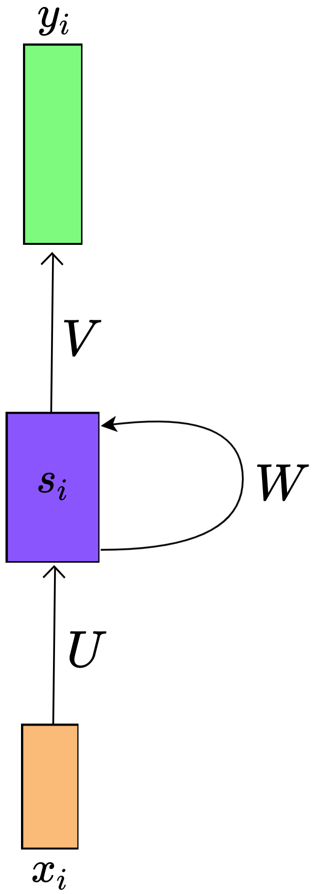
Module 2 : Roadmap

Mitesh M. Khapra

AI4Bharat, Department of Data Science and Artificial Intelligence, IIT Madras

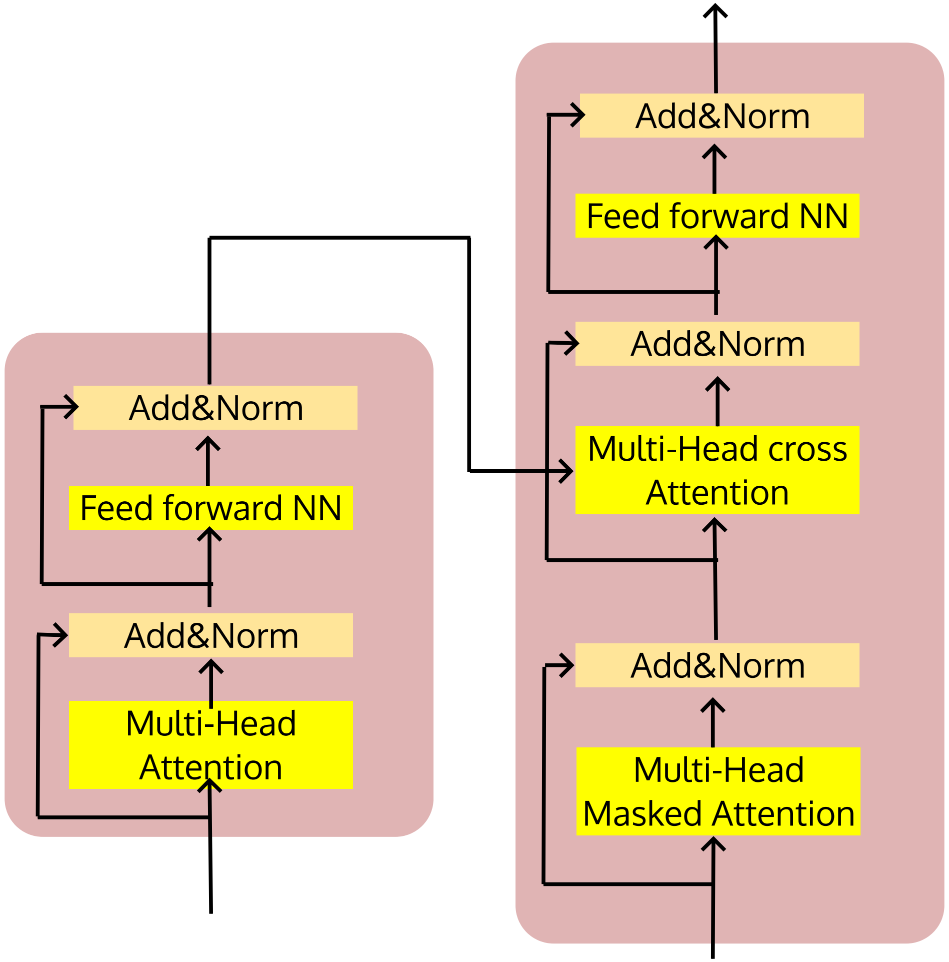


RNN based models for NLP

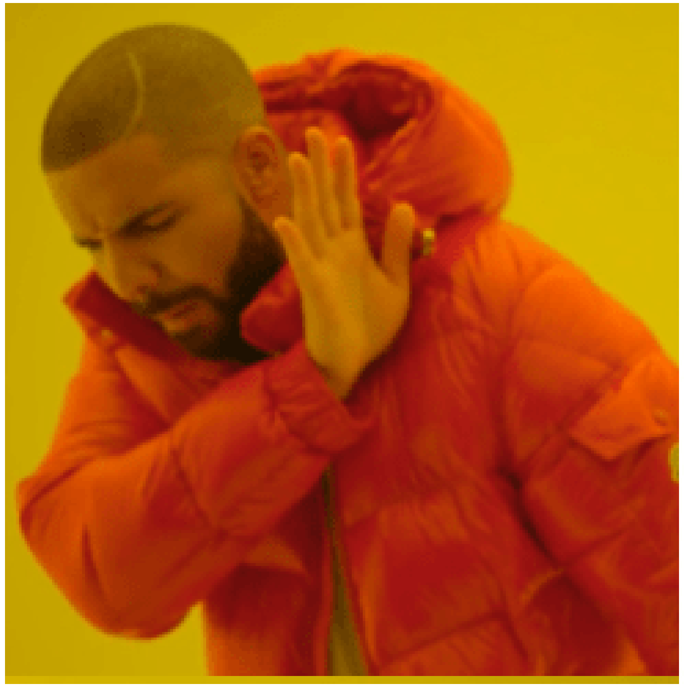


Paradigm shift

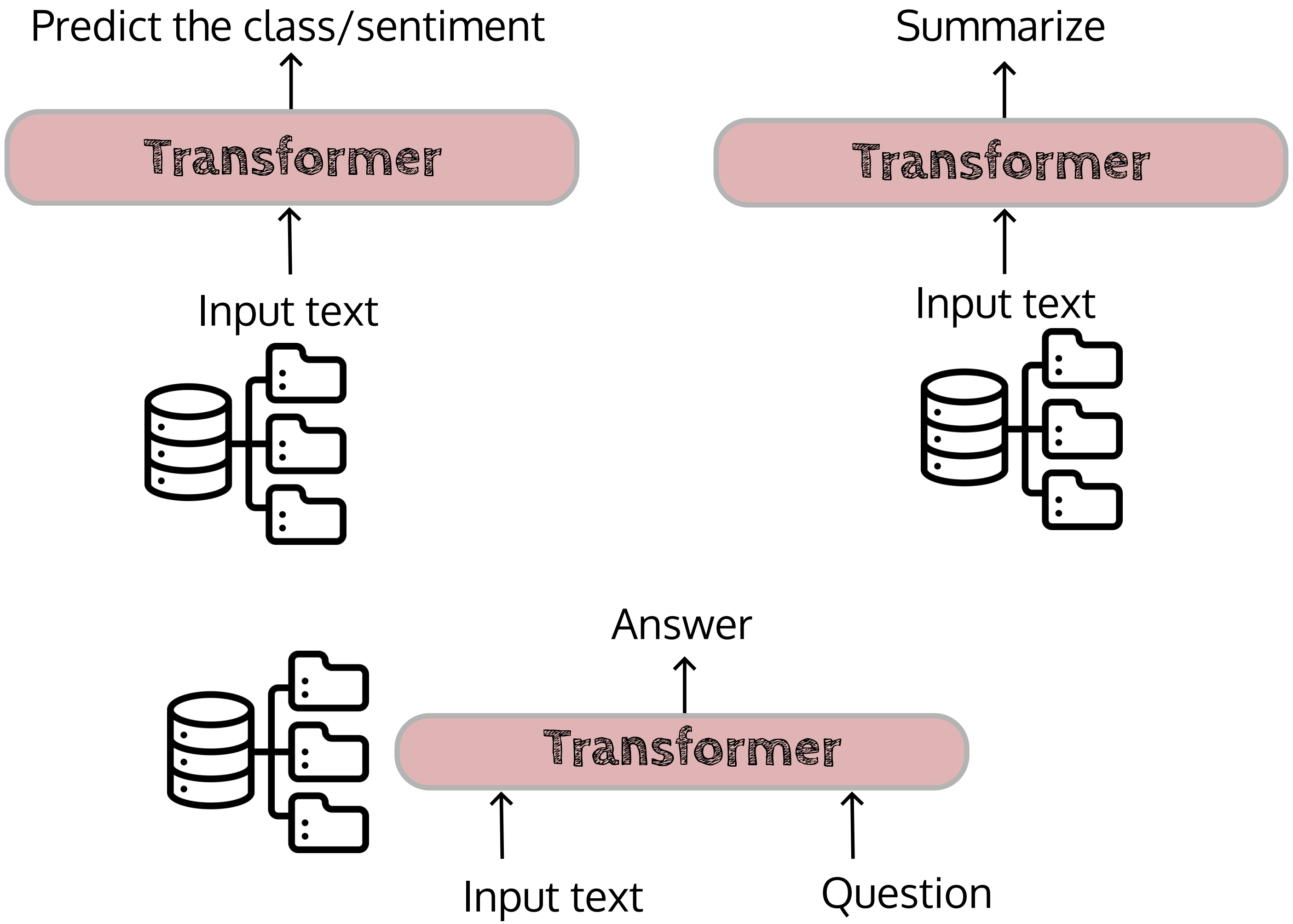
Transformer based models for NLP



Traditional NLP



Task-specific Supervised Training



Modern NLP

Pre-training, Fine Tuning, Instruction tuning,

Raw text data (cleaned)

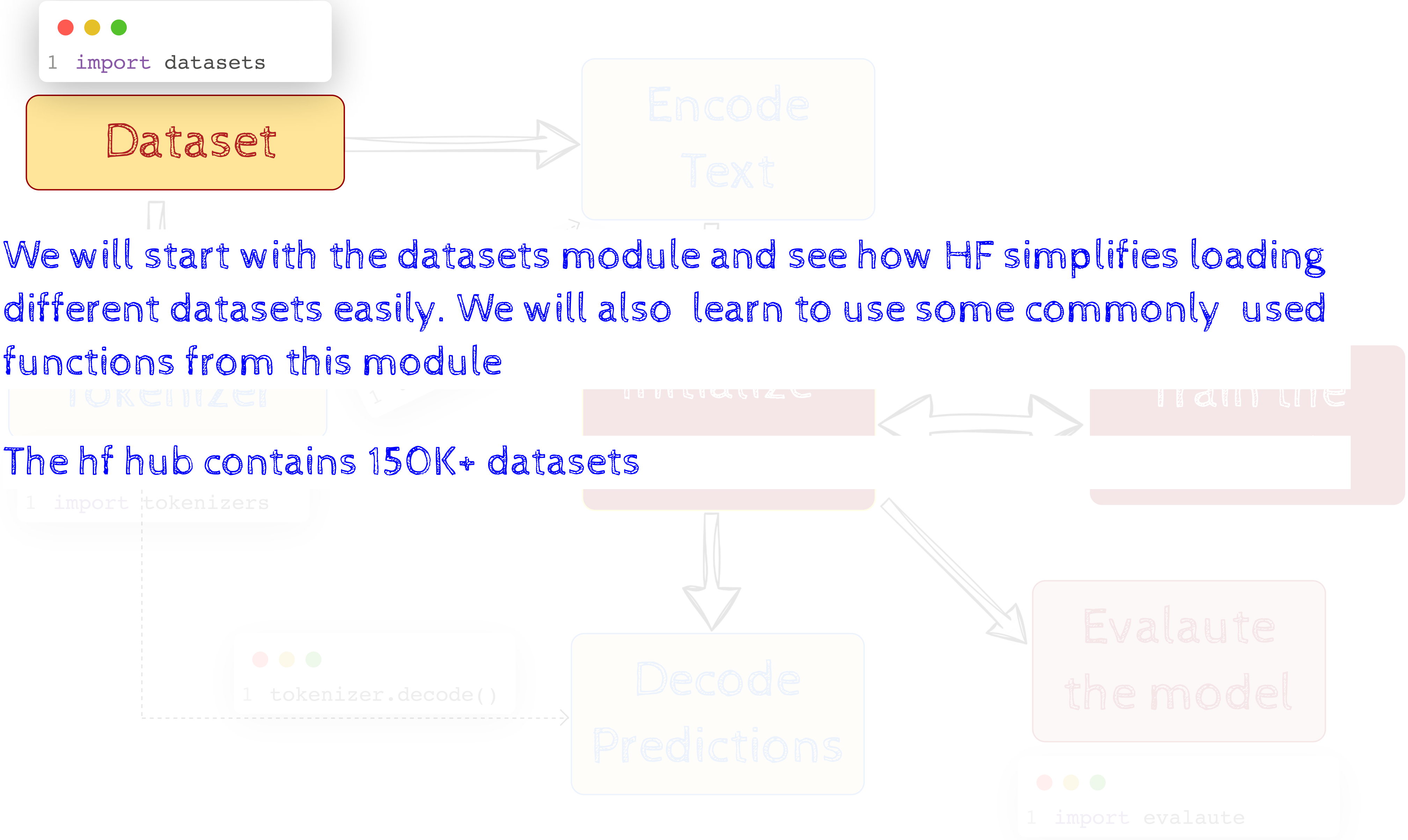
Output response **conditioned** on prompt

Language Modelling

Prompt: Input text

Prompt: Predict sentiment, summarize, fill in the blank, generate story

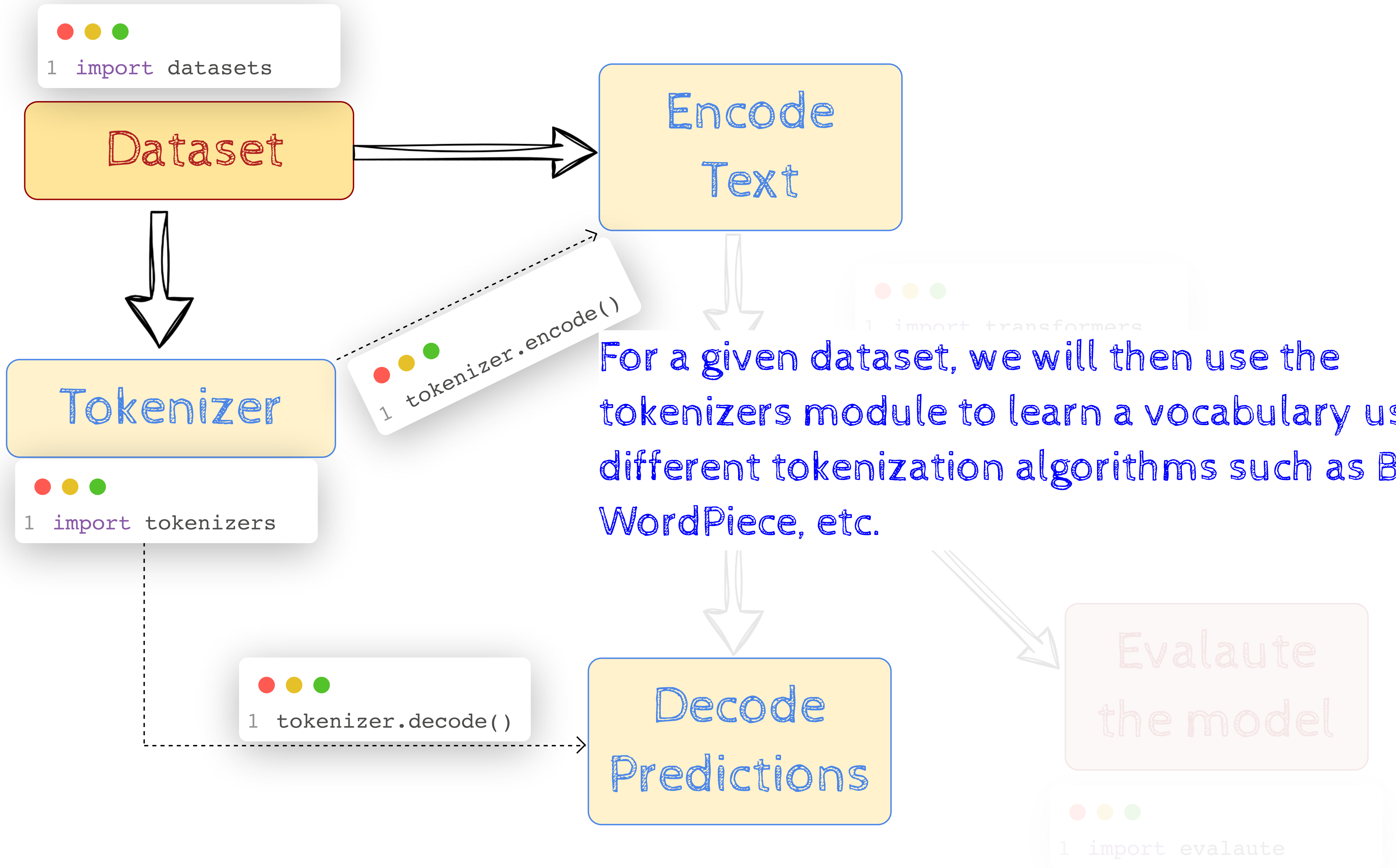
Syllabus Outline



We will start with the datasets module and see how HF simplifies loading different datasets easily. We will also learn to use some commonly used functions from this module

The hf hub contains 150K+ datasets

Syllabus Outline

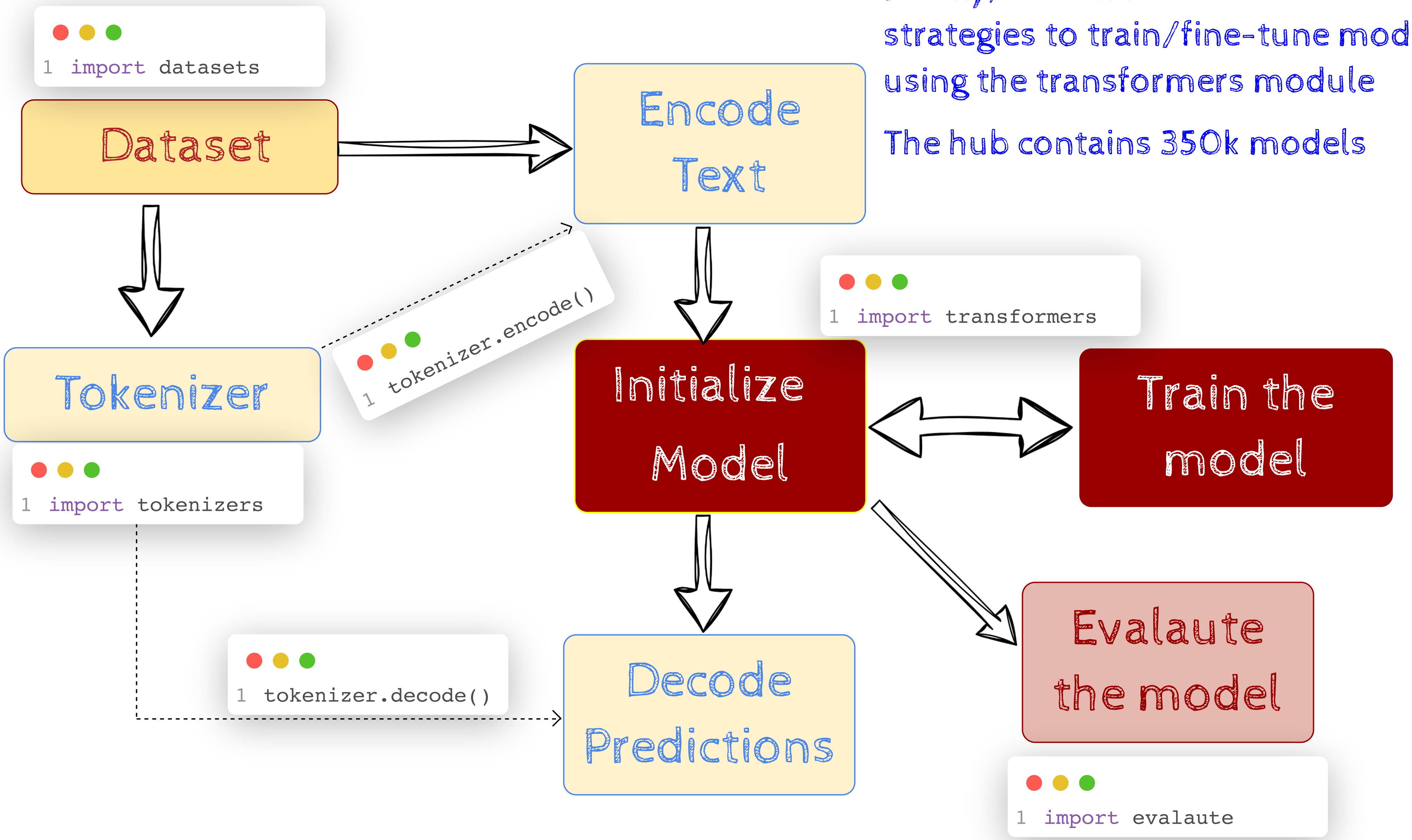


For a given dataset, we will then use the tokenizers module to learn a vocabulary using different tokenization algorithms such as BPE, WordPiece, etc.

Syllabus Outline

Finally, we will learn various strategies to train/fine-tune models using the transformers module

The hub contains 350k models



In each week, we cover the required concepts to understand various terminolgies that are helpful while developing a solution for NLP problems

We expect you to revisit the transformer architecture in detail as covered in the deep learning theory course

This week, we will delve deeper into the dataset part of the development cycle.

Module 3 : Datasets

Mitesh M. Khapra

AI4Bharat, Department of Data Science and Artificial Intelligence, IIT Madras

Task Specific Datasets

Learning starts with a dataset

We have a wide range of NLP tasks such as

- Sentiment classification
- Machine Translation
- Named Entity Recognition
- Question Answering
- Textual entailment
- Summarization
- Generation

For each of these tasks, we may have hundreds of datasets with thousands of samples

For example, for sentiment classification we have

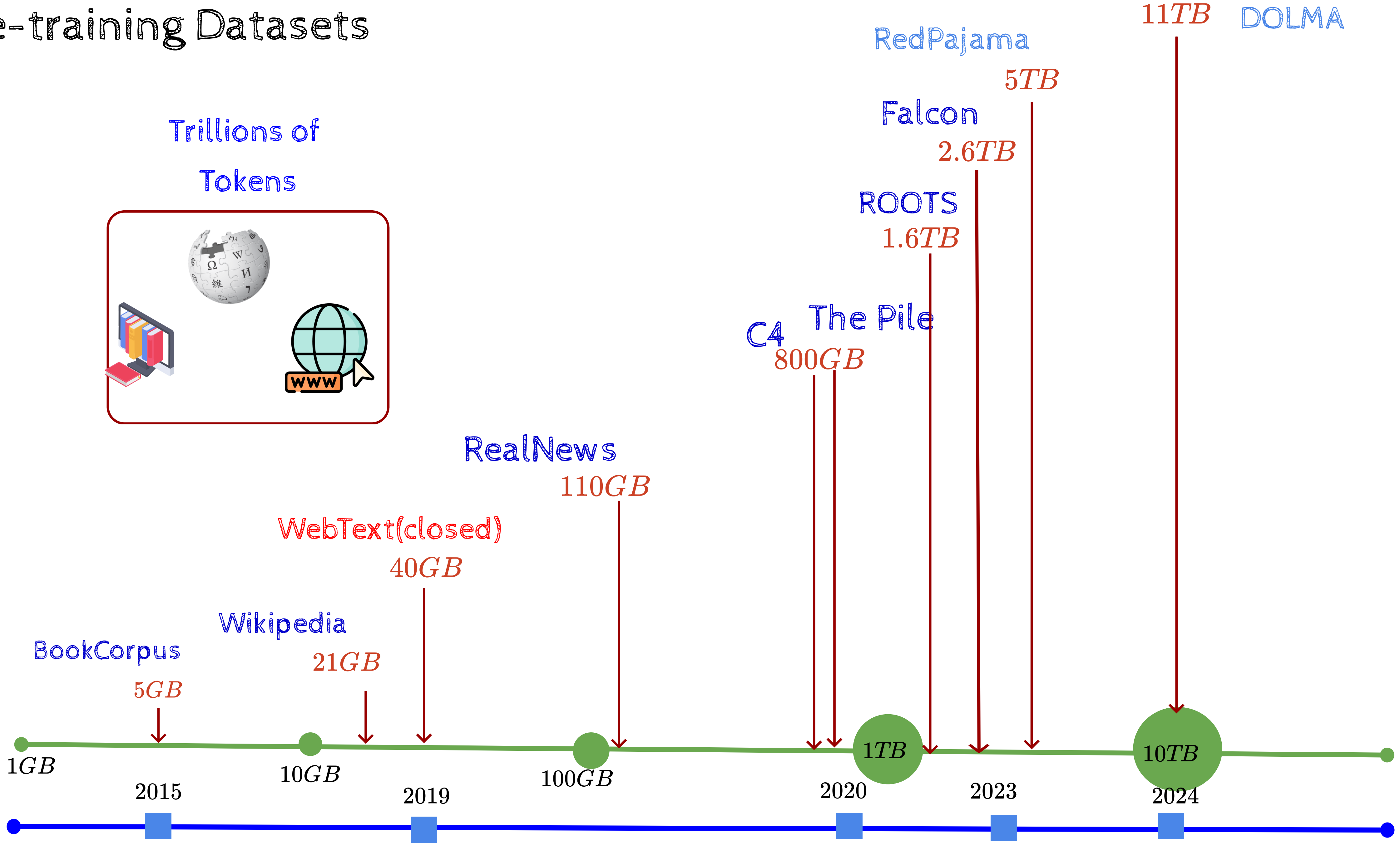
- Amazon review
- IMDB Moview review
- Twitter financial news sentiment
- poem sentiment

Similarly, one can find numerous datasets for other tasks

All these datasets typically have thousands to millions (if not billions) of words

In modern NLP, the pretraining datasets have billions and trillions of tokens

Pre-training Datasets



The datasets come with different formats and different sizes

We need to understand the format and may need to write a script to parse and load the text data

If the memory is limited, we need to implement mechanisms to stream the samples in the dataset .

However, it would be convenient if we have a single place to load any of these datasets with a consistent call signature

That's where Hugging Face's datasets module helps us

What is pre-training?

What are tokens?

Are tokens and the words in a sentence one and the same?

Well, we will discuss all these in subsequent lectures.

For now, let us just look at the datasets module from HF