HUGGING FACE

Question and Answering with Hugging Face

G R Priyavarshini (47)
Sangeetha Sankar (32)
Shafeena Farheen (31)

Objective

- Question Answering (QA) in Natural Language Processing (NLP) is a task that involves using computational methods to automatically answer questions posed in natural language.
- This task involves identifying the relevant information within a large corpus of text and extracting the answer that best addresses the question.

Two Main objective of this case study:

1.Review of what Question Answering is and where we use it.

2.How to use Hugging face Transformers for Question Answering with just a few lines of code.

How Question Answering works in NLP? Technique

- Question Answering (QA) in NLP involves identifying the answer to a question from a large corpus of text, such as a document or a collection of documents. The process typically involves the following steps:

[redacted] arison, etc.)

ts that

3. Candidate Answer Generation: Possible answers are extracted from the retrieved text snippets and candidate answers are generated.
4. Answer Selection: The candidate answers are evaluated against the question and the best answer is selected based on various criteria, such as relevance, accuracy, and confidence.

- The process can be aided by techniques such as natural language processing (NLP), information retrieval (IR), and machine learning (ML), among others. The performance of QA systems can be improved by using large pre-trained language models, such as BERT or GPT, to encode the context of the question and candidate answers.

Techniques that can be used

There are 2 different categories of QA Modeling

- Domain - systems that are constrained by the input data; we have open and closed systems
- **Open domain systems** are for broad questions, not specific to what category of discussion (Wikipedia, World wide web, Alexa, etc...)

- **Closed domain systems** are more narrow in its vocabulary and focus on a specific industry or topic (Football, Finance, Tech, Law etc..)
- Question Type - Open ended questions, Yes/No questions, inference questions etc...)
- Once you determine what type of system you want to establish, you then need to figure out which question type you want your model to focus on **Types of Question Answering Models**
- *Extractive Question Answering* is a deep learning model that can provide an answer when given a corpus of text (i.e context). So, when you provide a question to the model, the model then "searches" the documents to pinpoint the best answer to the question. It's essentially a searching tool in many ways...
- *Open Generative Question Answering* is a deep learning model that generates text based on context. So, unlike the extractive question answering model, the answer does not **literally** have to be in the text.
- *Closed Generative Question Answering* is a deep learning model where no context is provided and the answer is generated by the model.

Technique that is selected to do the case study

- The QA Models are essentially extensions of the BERT model with slightly different output layers. We are creating Extractive Question answering system as it is quite easy to implement. • RoBERTa is a new and improved version of BERT

- Removes the Next Sentence Prediction (NSP) objective
- Trained on bigger batch sizes & longer sequences
- Dynamically changes the masking pattern
- TRAINED on a large corpus of English data with **no** labeling whatsoever. (just the raw texts)
- Masks 15% of the input; RoBERTa runs the entire masked sentence through the model and the model attempts to predict the masked terms correctly.
- This is where the QA model learns the context and have a basic understanding to the language modeling!
- Fine tuned using SQuAD 2.0 dataset. It's been trained on question answer pairs, including unanswerable questions, for the task of Question Answering.

Use of the Question Answering

1. Customer Service: QA systems can be used to automate customer service by providing instant answers to common questions.

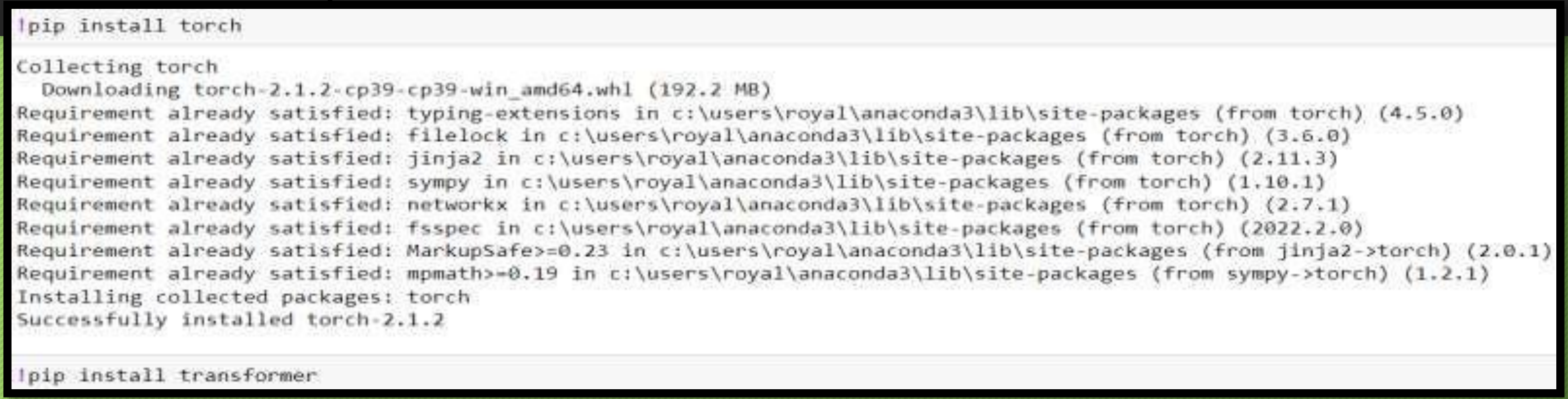
2. Knowledge Management: QA systems can be used to manage and retrieve information from large knowledge bases, such as company FAQs and product manuals.
3. Education: QA systems can be used to provide students with instant answers to their questions and help them learn more effectively.
4. Healthcare: QA systems can be used to provide healthcare professionals with instant answers to clinical questions, improving patient outcomes and reducing diagnostic errors.
5. News and Media: QA systems can be used to provide quick answers to factual questions about current events, politics, sports, and more.
6. Legal: QA systems can be used to provide lawyers and legal researchers with instant answers to legal questions, improving the speed and efficiency of their work.

Advantages of using Transformers for Question Answering

1. Contextual Embeddings: Transformers can capture the context of a question and candidate answer, allowing them to better understand the relationship between the two and make more accurate predictions.
2. Pre-training: Transformers can be pre-trained on large amounts of text data, giving them a strong understanding of language and enabling them to perform well on a wide range of tasks, including QA.
3. Attention Mechanisms: Transformers use attention mechanisms, which allow them to focus on specific parts of the input when making predictions, leading to improved performance on QA tasks.
4. Transfer Learning: Transformers can be fine-tuned for specific QA tasks, allowing for transfer learning from a pre-trained model and reducing the amount of labeled data needed for training.
5. Performance: Transformers have demonstrated state-of-the-art performance on a wide range of QA tasks, outperforming traditional methods in terms of accuracy and speed.

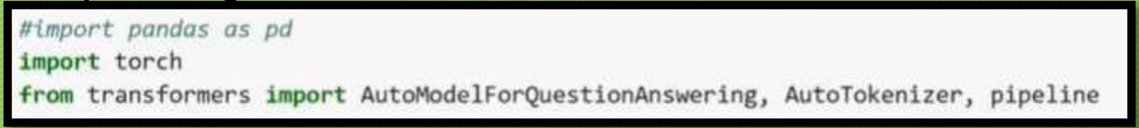
Code

- Install the required Libraries



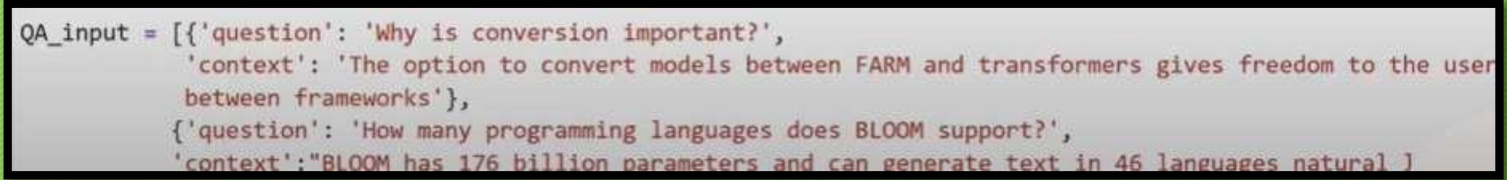
```
!pip install torch

Collecting torch
  Downloading torch-2.1.2-cp39-cp39-win_amd64.whl (192.2 MB)
Requirement already satisfied: typing-extensions in c:\users\royal\anaconda3\lib\site-packages (from torch) (4.5.0)
Requirement already satisfied: filelock in c:\users\royal\anaconda3\lib\site-packages (from torch) (3.6.0)
Requirement already satisfied: jinja2 in c:\users\royal\anaconda3\lib\site-packages (from torch) (2.11.3)
Requirement already satisfied: sympy in c:\users\royal\anaconda3\lib\site-packages (from torch) (1.10.1)
Requirement already satisfied: networkx in c:\users\royal\anaconda3\lib\site-packages (from torch) (2.7.1)
Requirement already satisfied: fsspec in c:\users\royal\anaconda3\lib\site-packages (from torch) (2022.2.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\royal\anaconda3\lib\site-packages (from jinja2->torch) (2.0.1)
Requirement already satisfied: mpmath>=0.19 in c:\users\royal\anaconda3\lib\site-packages (from sympy->torch) (1.2.1)
Installing collected packages: torch
Successfully installed torch-2.1.2

!pip install transformer
```

- Importing Libraries



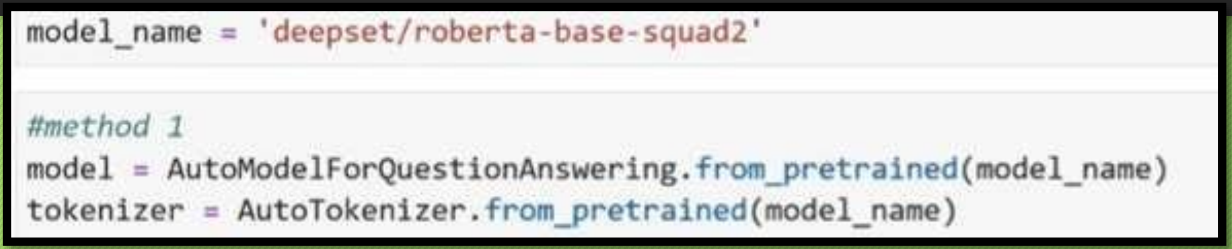
```python
#import pandas as pd
import torch
from transformers import AutoModelForQuestionAnswering, AutoTokenizer, pipeline
```

- Question and Answer pairs that can be passed to the transformers-Extractive QA system

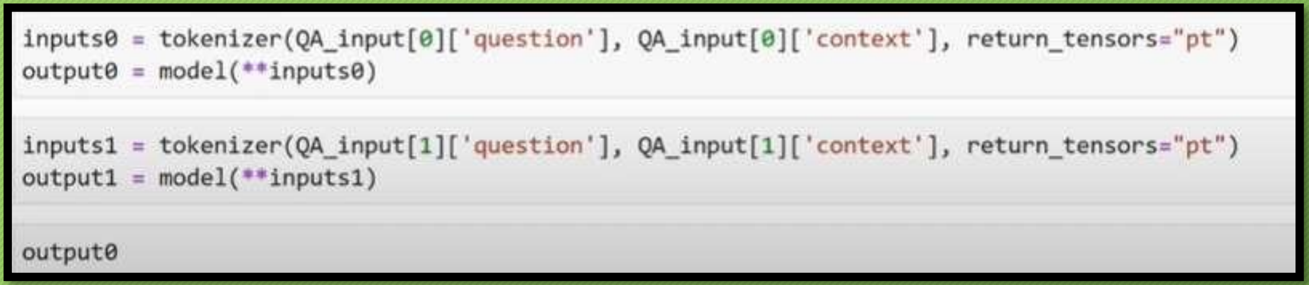


```python
QA_input = [{'question': 'Why is conversion important?',
             'context': 'The option to convert models between FARM and transformers gives freedom to the user
             between frameworks'},
            {'question': 'How many programming languages does BLOOM support?',
             'context':"BLOOM has 176 billion parameters and can generate text in 46 languages natural ]
```

Code



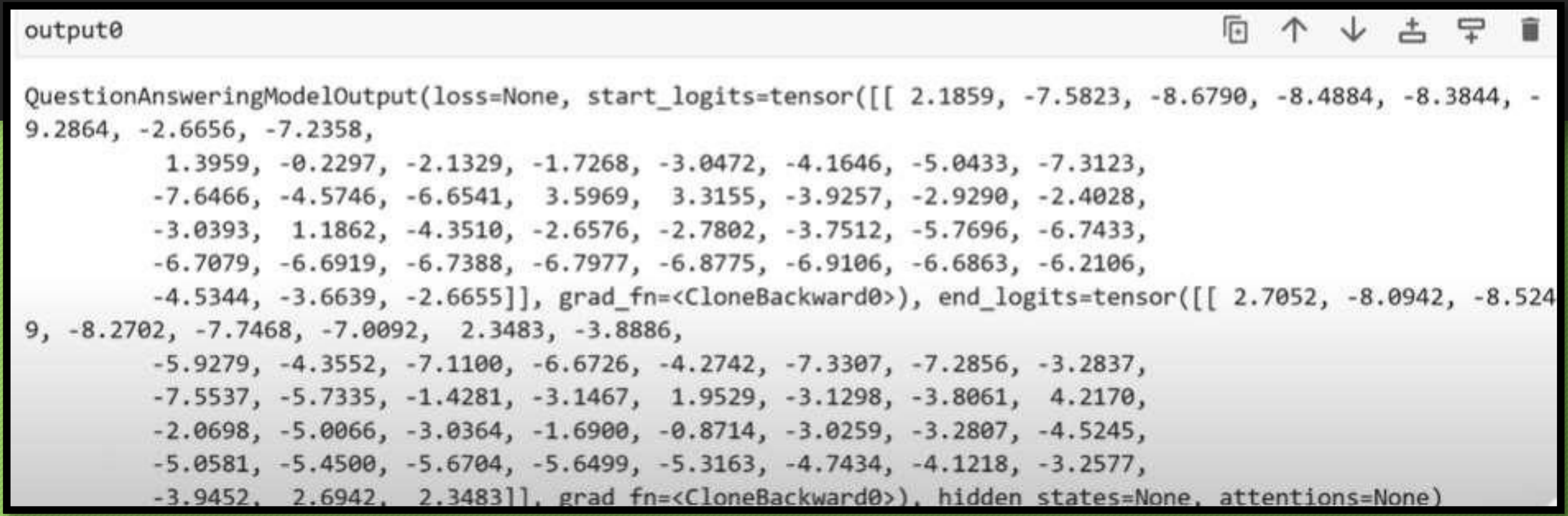
```
model_name = 'deepset/roberta-base-squad2'

#method 1
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```



```
inputs0 = tokenizer(QA_input[0]['question'], QA_input[0]['context'], return_tensors="pt")
output0 = model(**inputs0)

inputs1 = tokenizer(QA_input[1]['question'], QA_input[1]['context'], return_tensors="pt")
output1 = model(**inputs1)

output0
```

- Select from the hub of pretrained models
- Initialize the model and use the corresponding tokenizer(Tokens to ids and then further passed to the model)
- Tokenize the question and answer and then returned as Tensors
- Output is generated by passing the input to the model

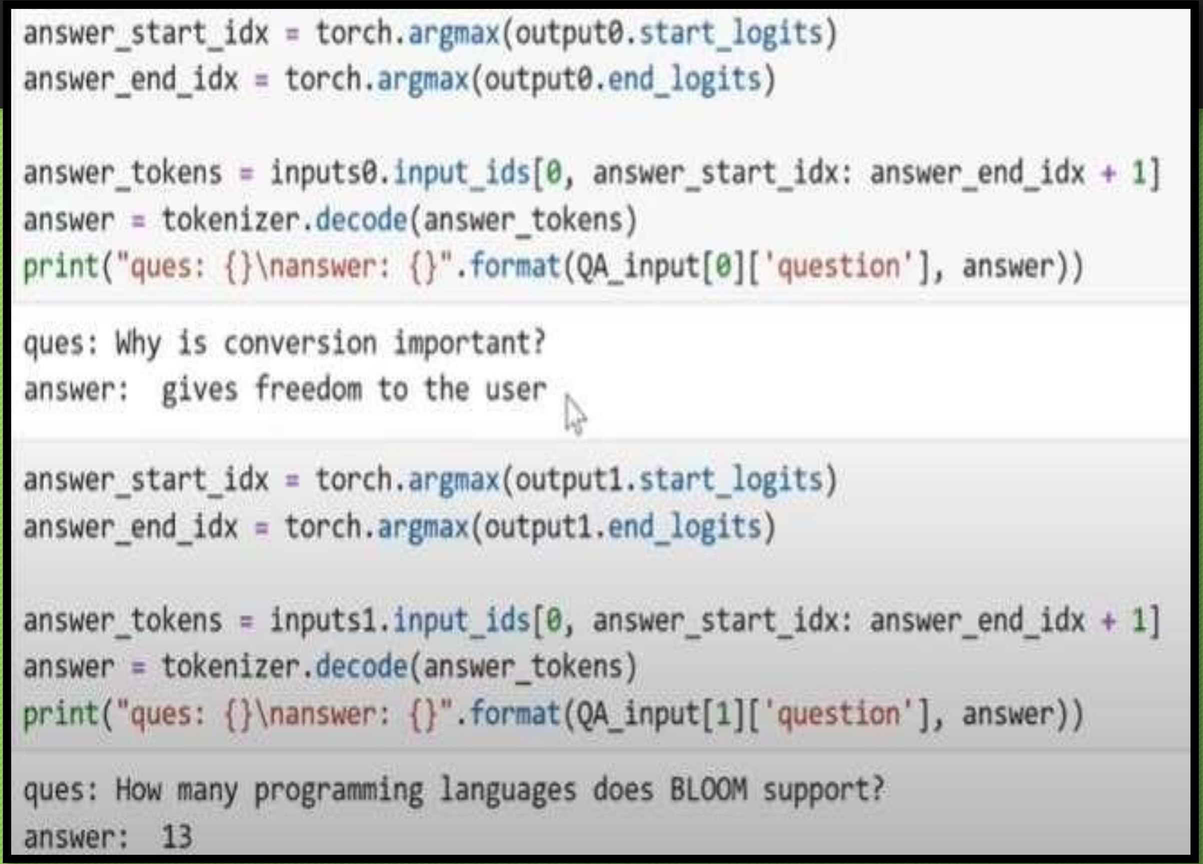
Code and Output



```
output0

QuestionAnsweringModelOutput(loss=None, start_logits=tensor([[ 2.1859, -7.5823, -8.6790, -8.4884, -8.3844, -
9.2864, -2.6656, -7.2358,
          1.3959, -0.2297, -2.1329, -1.7268, -3.0472, -4.1646, -5.0433, -7.3123,
         -7.6466, -4.5746, -6.6541,  3.5969,  3.3155, -3.9257, -2.9290, -2.4028,
         -3.0393,  1.1862, -4.3510, -2.6576, -2.7802, -3.7512, -5.7696, -6.7433,
         -6.7079, -6.6919, -6.7388, -6.7977, -6.8775, -6.9106, -6.6863, -6.2106,
         -4.5344, -3.6639, -2.6655]], grad_fn=<CloneBackward0>), end_logits=tensor([[ 2.7052, -8.0942, -8.524
9, -8.2702, -7.7468, -7.0092,  2.3483, -3.8886,
         -5.9279, -4.3552, -7.1100, -6.6726, -4.2742, -7.3307, -7.2856, -3.2837,
         -7.5537, -5.7335, -1.4281, -3.1467,  1.9529, -3.1298, -3.8061,  4.2170,
         -2.0698, -5.0066, -3.0364, -1.6900, -0.8714, -3.0259, -3.2807, -4.5245,
         -5.0581, -5.4500, -5.6704, -5.6499, -5.3163, -4.7434, -4.1218, -3.2577,
         -3.9452,  2.6942,  2.3483]], grad_fn=<CloneBackward0>), hidden_states=None, attentions=None)
```

- Since, we are only interested in getting answers so we haven’t specified hidden states, attentions or Loss as True.
- start_logits denote starting token id for the answer.
- end_logits denote ending token id for the answer.

Code and Output



```
answer_start_idx = torch.argmax(output0.start_logits)
answer_end_idx = torch.argmax(output0.end_logits)

answer_tokens = inputs0.input_ids[0, answer_start_idx: answer_end_idx + 1]
answer = tokenizer.decode(answer_tokens)
print("ques: {}\nanswer: {}".format(QA_input[0]['question'], answer))
```

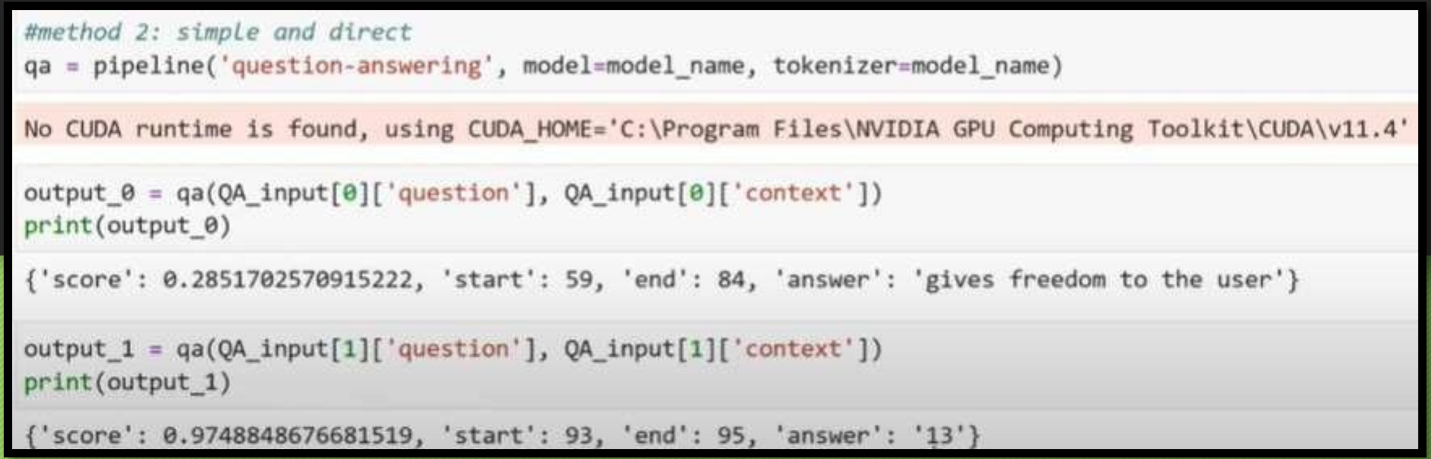
```
ques: Why is conversion important?
answer:  gives freedom to the user
```

```
answer_start_idx = torch.argmax(output1.start_logits)
answer_end_idx = torch.argmax(output1.end_logits)

answer_tokens = inputs1.input_ids[0, answer_start_idx: answer_end_idx + 1]
answer = tokenizer.decode(answer_tokens)
print("ques: {}\nanswer: {}".format(QA_input[1]['question'], answer))
```

```
ques: How many programming languages does BLOOM support?
answer:  13
```

Method 2

- Based on the start_logits and end_logits we are going to select starting and ending idex of the answers.
- These idex are passed to input ids that are already converted based on the text.
- So, we filter corresponding regions from the context and we decode it.
- Decode function is used to convert token ids to original text form.
- We can get correct answer for the corresponding question
- In this method we can get top k answers also that is one of the advantage of this method



```
#method 2: simple and direct
qa = pipeline('question-answering', model=model_name, tokenizer=model_name)
```

```
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4'
```

```
output_0 = qa(QA_input[0]['question'], QA_input[0]['context'])
print(output_0)
```

```
{'score': 0.2851702570915222, 'start': 59, 'end': 84, 'answer': 'gives freedom to the user'}
```

```
output_1 = qa(QA_input[1]['question'], QA_input[1]['context'])
print(output_1)
```

```
{'score': 0.9748848676681519, 'start': 93, 'end': 95, 'answer': '13'}
```

- Simple and easy to implement
- But one of the disadvantages is that currently all the pretrained models are not best for the pipeline
- Gives the answer for the particular input.

Reference Links

- https://huggingface.co/models
- https://huggingface.co/docs/transformers/main/main_classes/pipelines
- https://github.com/huggingface/transformers/issues/3207

Thank You