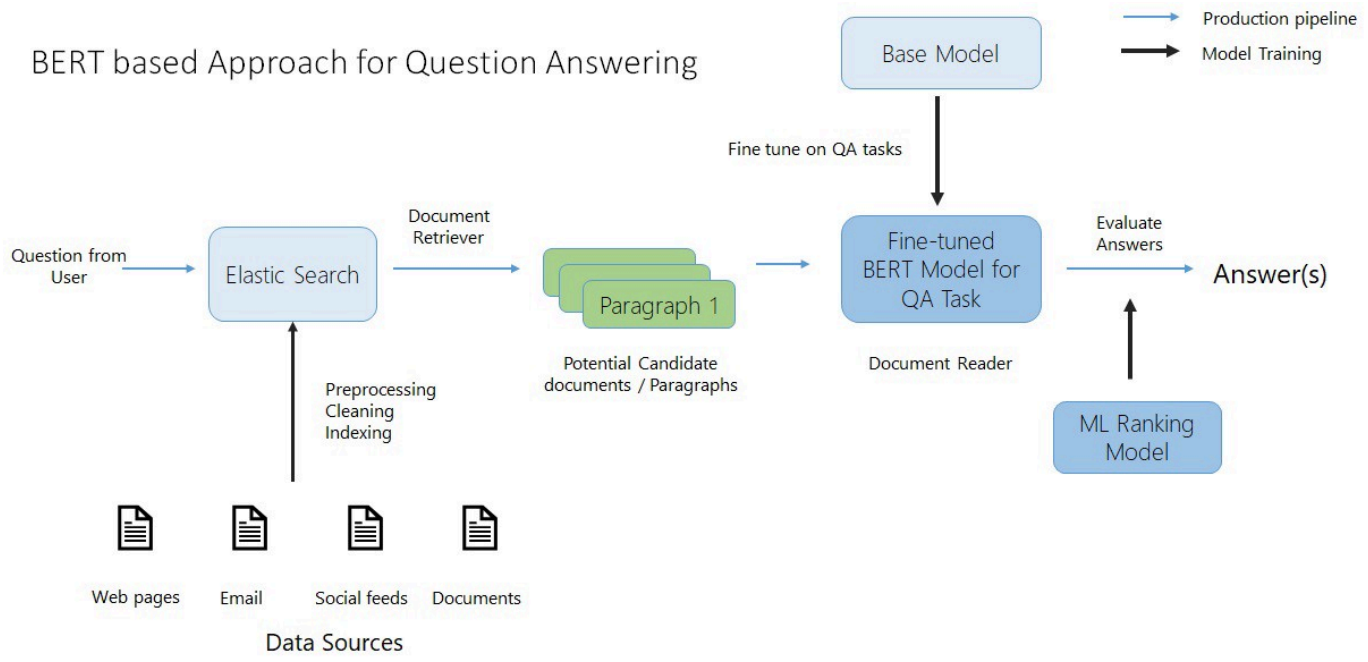

Task 3: Question Answering with Transformers

- Dataset (Recommended): SQuAD v1.1 - Stanford Question Answering (Kaggle)
- Build a system that answers questions based on a given context or passage
- Use pre-trained transformer models (e.g., BERT) fine-tuned for question answering
- Feed the model both the context and the question, and extract the correct answer span
- Evaluate with exact match and F1 score

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/d
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-pack
```

```
import json
import pandas as pd
import numpy as np
from pathlib import Path
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForQuestionAnswering
import time
```

```
is_cuda = torch.cuda.is_available()

# If we have a GPU available, we'll set our device to GPU.
if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")
else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")
```

```
GPU is available
```

```
!mkdir squad
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v1.1.json -O squad/train-v1.1
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v1.1.json -O squad/dev-v1.1.jsc
```

```
--2025-08-27 13:12:41--  https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v1.1.j
Resolving rajpurkar.github.io (rajpurkar.github.io)... 185.199.108.153, 185.199.109.153,
Connecting to rajpurkar.github.io (rajpurkar.github.io)|185.199.108.153|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: 30288272 (29M) [application/json]
Saving to: ‘squad/train-v1.1.json’

squad/train-v1.1.js 100%[===================>]  28.88M  --.-KB/s    in 0.06s

2025-08-27 13:12:42 (447 MB/s) - ‘squad/train-v1.1.json’ saved [30288272/30288272]

--2025-08-27 13:12:42--  https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v1.1.jsc
Resolving rajpurkar.github.io (rajpurkar.github.io)... 185.199.108.153, 185.199.109.153,
Connecting to rajpurkar.github.io (rajpurkar.github.io)|185.199.108.153|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: 4854279 (4.6M) [application/json]
Saving to: ‘squad/dev-v1.1.json’

squad/dev-v1.1.json 100%[===================>]   4.63M  --.-KB/s    in 0.02s

2025-08-27 13:12:42 (266 MB/s) - ‘squad/dev-v1.1.json’ saved [4854279/4854279]
```

Storing The Data

```
# Define the path to the SQuAD 1.1 training data
path = "squad/train-v1.1.json"

# Load and preprocess the SQuAD 2.0 data
def load_squad_data(path):
    with open(path, 'rb') as f:
        squad_dict = json.load(f)

    texts = []
    questions = []
    answers = []

    for group in squad_dict['data']:
        for passage in group['paragraphs']:
            context = passage['context']
            for qa in passage['qas']:
                question = qa['question']
                for answer in qa['answers']:
                    texts.append(context)
                    questions.append(question)
                    answers.append(answer)

    return texts, questions, answers

# Preprocess the data to find answer start and end positions
train_texts, train_queries, train_answers = load_squad_data(path)


# Give the path for SQuAD 1.1 validation data
path = Path('squad/dev-v1.1.json')

# Load and preprocess the SQuAD 2.0 data
def load_squad_data(path):
    with open(path, 'rb') as f:
        squad_dict = json.load(f)

    texts = []
    questions = []
    answers = []

    for group in squad_dict['data']:
        for passage in group['paragraphs']:
            context = passage['context']
            for qa in passage['qas']:
                question = qa['question']
                for answer in qa['answers']:
                    texts.append(context)
```

```
                    questions.append(question)
                    answers.append(answer)

    return texts, questions, answers

# Preprocess the data to find answer start and end positions
val_texts, val_queries, val_answers = load_squad_data(path)
```

Checking The Data

```
print(len(train_texts))
print(len(train_queries))
print(len(train_answers))
```

```
87599
87599
87599
```

```
print("Passage: ",train_texts[0])
print("Query: ",train_queries[0])
print("Answer: ",train_answers[0])
```

```
Passage:  Architecturally, the school has a Catholic character. Atop the Main Building's
Query:  To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?
Answer:  {'answer_start': 515, 'text': 'Saint Bernadette Soubirous'}
```

```
print(len(val_texts))
print(len(val_queries))
print(len(val_answers))
```

```
34726
34726
34726
```

Find The Start and End Position Character

```
for answer, text in zip(train_answers, train_texts):
    real_answer = answer['text']
    start_idx = answer['answer_start']
    # Get the real end index
    end_idx = start_idx + len(real_answer)

    # Deal with the problem of 1 or 2 more characters
    if text[start_idx:end_idx] == real_answer:
```

```
        answer['answer_end'] = end_idx
    # When the real answer is more by one character
    elif text[start_idx-1:end_idx-1] == real_answer:
        answer['answer_start'] = start_idx - 1
        answer['answer_end'] = end_idx - 1
    # When the real answer is more by two characters
    elif text[start_idx-2:end_idx-2] == real_answer:
        answer['answer_start'] = start_idx - 2
        answer['answer_end'] = end_idx - 2


for answer, text in zip(val_answers, val_texts):
    real_answer = answer['text']
    start_idx = answer['answer_start']
    # Get the real end index
    end_idx = start_idx + len(real_answer)

    # Deal with the problem of 1 or 2 more characters
    if text[start_idx:end_idx] == real_answer:
        answer['answer_end'] = end_idx
    # When the real answer is more by one character
    elif text[start_idx-1:end_idx-1] == real_answer:
        answer['answer_start'] = start_idx - 1
        answer['answer_end'] = end_idx - 1
    # When the real answer is more by two characters
    elif text[start_idx-2:end_idx-2] == real_answer:
        answer['answer_start'] = start_idx - 2
        answer['answer_end'] = end_idx - 2
```

Tokenize Passages And Queries

```
from huggingface_hub import login
login(token="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")
```

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https:/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public model
  warnings.warn(
```

tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 4.64kB/s]

config.json: 100% 570/570 [00:00<00:00, 57.2kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 1.07MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 22.9MB/s]

```
train_encodings = tokenizer(train_texts, train_queries, truncation=True, padding=True)
val_encodings = tokenizer(val_texts, val_queries, truncation=True, padding=True)


def add_token_positions(encodings, answers):
    start_positions = []
    end_positions = []

    count = 0

    for i in range(len(answers)):
        start_positions.append(encodings.char_to_token(i, answers[i]['answer_start']))
        end_positions.append(encodings.char_to_token(i, answers[i]['answer_end']))

        # if start position is None, the answer passage has been truncated
        if start_positions[-1] is None:
            start_positions[-1] = tokenizer.model_max_length

        # if end position is None, the 'char_to_token' function points to the space after th
        if end_positions[-1] is None:
            end_positions[-1] = encodings.char_to_token(i, answers[i]['answer_end'] - 1)

            # if end position is still None the answer passage has been truncated
            if end_positions[-1] is None:
                count += 1
                end_positions[-1] = tokenizer.model_max_length
    print(count)

    # Update the data in dictionary
    encodings.update({'start_positions': start_positions, 'end_positions': end_positions})

add_token_positions(train_encodings, train_answers)
add_token_positions(val_encodings, val_answers)
```

```
10
23
```

Create a Dataset Class And Use of DataLoader

```
class SquadDataset(torch.utils.data.Dataset):
    def __init__(self, encodings):
        self.encodings = encodings

    def __getitem__(self, idx):
        return {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

    def __len__(self):
        return len(self.encodings.input_ids)
```

```
train_dataset = SquadDataset(train_encodings)
val_dataset = SquadDataset(val_encodings)
```

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=True)
```

```
device = torch.device('cuda:0' if torch.cuda.is_available()
                      else 'cpu')
```

Build the Bert Model

```
model = BertForQuestionAnswering.from_pretrained('bert-base-uncased').to(device)
```

```
model.safetensors:  100%                                    440M/440M  [01:51<00:00,  4.72MB/s]
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint
You should probably TRAIN this model on a down-stream task to be able to use it for prec
```

```
from torch.optim import AdamW

optim = AdamW(model.parameters(), lr=5e-5)
epochs = 3
```

```
whole_train_eval_time = time.time()

train_losses = []
val_losses = []

print_every = 2000
```

```
for epoch in range(epochs):
    epoch_time = time.time()

    # Set model in train mode
    model.train()
    loss_of_epoch = 0

    print("############Train############")

    for batch_idx,batch in enumerate(train_loader):
        optim.zero_grad()

        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        start_positions = batch['start_positions'].to(device)
        end_positions = batch['end_positions'].to(device)

        outputs = model(input_ids, attention_mask=attention_mask, start_positions=start_pos
        loss = outputs[0]
        # do a backwards pass
        loss.backward()
        # update the weights
        optim.step()
        # Find the total loss
        loss_of_epoch += loss.item()

        if (batch_idx+1) % print_every == 0:
            print("Batch {:} / {:}".format(batch_idx+1,len(train_loader)),"\nLoss:", round

    loss_of_epoch /= len(train_loader)
    train_losses.append(loss_of_epoch)

    ##########Evaluation##################

    # Set model in evaluation mode
    model.eval()

    print("############Evaluate############")

    loss_of_epoch = 0

    for batch_idx,batch in enumerate(val_loader):

        with torch.no_grad():

            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            start_positions = batch['start_positions'].to(device)
            end_positions = batch['end_positions'].to(device)
```

```
            outputs = model(input_ids, attention_mask=attention_mask, start_positions=start
            loss = outputs[0]
            # Find the total loss
            loss_of_epoch += loss.item()

        if (batch_idx+1) % print_every == 0:
            print("Batch {:} / {:}".format(batch_idx+1,len(val_loader)),"\nLoss:", round(lo

    loss_of_epoch /= len(val_loader)
    val_losses.append(loss_of_epoch)

    # Print each epoch's time and train/val loss

    print("\n-------Epoch ", epoch+1,
          "-------"
          "\nTraining Loss:", train_losses[-1],
          "\nValidation Loss:", val_losses[-1],
          "\nTime: ",(time.time() - epoch_time),
          "\n----------------------",
          "\n\n")
```

```
############Train############
Batch 2000 / 5475
Loss: 1.8

Batch 4000 / 5475
Loss: 1.6
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Start coding or generate with AI.