In [2]:
```
!pip install SpeechRecognition pydub
```

```
Requirement already satisfied: SpeechRecognition in d:\ana\lib\site-packages (3.8.1)
Requirement already satisfied: pydub in d:\ana\lib\site-packages (0.25.1)
```

In [3]:

```
import assemblyai as aai


aai.settings.api_key = "190b27abe6964184ab62071ab0afb349"

# Initialize the Transcriber
transcriber = aai.Transcriber()

audio_file_path = "output.mp3"
transcript = transcriber.transcribe(audio_file_path)

print(transcript.text)
```

t learns from its mistakes and experiences. This model is used in games like Prince of Persia, or Assassin's Creed o
r FIFA, wherein the level of difficulty increases as you get better with the games. Just to make it more clear for y
ou, let's look at a comparison between supervised and unsupervised learning. Firstly, the data involved in case of s
upervised learning is labeled. As we mentioned in the examples previously, we provide the system with a photo of an
apple and let the system know that this is actually an apple. That is called label data. So the system learns from t
he label data and makes future predictions. Now, unsupervised learning does not require any kind of label data becau
se its work is to look for patterns in the input data and organize it. The next point is that you get a feedback in
case of supervised learning. That is, once you get the output, the system tends to remember that and uses it for the
next operation. That does not happen for unsupervised learning. And the last point is that supervised learning is mo
stly used to predict data, whereas unsupervised learning is used to find out hidden patterns or structures in data.
I think this would have made a lot of things clear for you regarding supervised and unsupervised learning. Now, le
t's talk about a question that everyone needs to answer before building a machine learning model. What kind of a mac
hine learning solution should we use? Yes, you should be very careful with selecting the right kind of solution for
your model, because if you don't, you might end up losing a lot of time, energy and processing cost. I won't be nami
ng the actual solutions because you guys aren't familiar with them yet. So we will be looking at it based on supervi
sed, unsupervised, and reinforcement learning. So let's look into the factors that might help us select the right ki
nd of machine learning solution. First factor is the problem statement describes the kind of model you will be build
ing. Or as the name suggests, it tells you what the problem is. For example, let's say the problem is to predict the
future stock market prices. So for anyone who is new to machine learning would have trouble figuring out the right s
olution. But with time and practice, you will understand that for a problem statement like this, solution based on s

In [238]:

```
from IPython.display import Audio

# Play the MP3 file
audio_mp3 = Audio('output.mp3')
display(audio_mp3)
```

0:00 / 21:08

In [60]:

```python
from transformers import BertTokenizer
from docx import Document

def read_docx(file_path):
    """Read text from a .docx file."""
    doc = Document(file_path)
    text = ''
    for para in doc.paragraphs:
        text += para.text + ' '
    return text

def preprocess_text(text):
    """Preprocess text using BERT tokenizer."""
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    return inputs

# Example usage
doc_file_path = 'output.docx'
document_text = read_docx(doc_file_path)
preprocessed_text = preprocess_text(document_text)
print(preprocessed_text)
```

```
{'input_ids': tensor([[  101,  4955,  7632,  4364,  1010,  1045,  1005,  1049,  2019,  4313,
          8193,  2013,  3432,  4553,  1012,  2651,  1010,  1045,  2097,  2507,
          2017,  1037, 14924,  4818,  2006,  3698,  4083,  1010,  1037,  8476,
          2008,  2003,  3811,  7882,  1999,  2974,  2651,  1012,  3251,  2017,
          1005,  2128,  2047,  2000,  1996,  3395,  2030,  2031,  2070,  3716,
          1010,  2023,  2678,  2097,  3073,  7070, 20062,  1012,  2166,  2302,
          3698,  4083,  2302,  3698,  4083,  1010,  2256,  3268,  2052,  2022,
          6022,  2062, 10368,  1012,  2005,  6013,  1024,  2592, 26384,  1024,
          4531,  2592,  2006,  1037,  8476,  2066,  1000,  3274,  1000,  2052,
          5478,  2183,  2083,  3365,  2808,  1998,  4790, 21118,  1012, 13268,
          5038,  1024,  2077,  3698,  4083,  1010, 13268,  5038,  2001,  2069,
          2464,  1999, 16596,  1011, 10882,  5691,  1012,  2651,  1010,  7248,
          2066,  9130,  8073,  6415,  2111,  1999,  7760,  1012,  7484, 16838,
          1024,  2909,  2072,  1010,  2522, 13320,  2532,  1010,  1998,  2714,
         16838, 11160,  2006,  3698,  4083,  2000,  3853,  6464,  1012,  2166,
          2007,  3698,  4083,  3698,  4083, 11598,  2015,  2536,  5919,  1997,
          2256,  3679,  3268,  1024, 10355,  1024,  7484,  4507,  1998,  9218,
          2491,  1999,  2399,  2066,  8827,  2549,  1998, 12202, 11598,  1996,
         10355,  3325,  1012,  1999,  2399,  2066,  5713,  1010,  1996,  9932,
         15581,  2015,  2000,  2115,  5656,  1012,  1041,  1011,  6236,  1024,
          9733,  3594,  3698,  4083,  2005,  4031, 11433,  1010,  8790, 20874,
          1010,  1998,  8013,  6903,  3370,  1012,  5193,  1024, 19169,  3594,
          3698,  4083,  2000,  6592, 14345,  1998, 23569, 27605,  4371,  5847,
          2241,  2006,  2536,  5876,  2066,  4026,  1998,  8599,  1012,  2054,
          2003,  3698,  4083,  1029,  3698,  4083,  2003,  1037,  3589,  1997,
          7976,  4454,  2008, 12939,  3001,  2000,  4553,  1998,  5335,  2013,
          3325,  2302, 13216,  4730,  1012,  2005,  2742,  1010,  1037,  3698,
          4083,  2291,  2064,  4553,  2000,  6807, 10962,  2011, 20253,  4871,
          1998, 12151,  7060,  1012,  4127,  1997,  3698,  4083, 13588,  4083,
          1999, 13588,  4083,  1010,  1037,  2944,  2003,  4738,  2006, 12599,
          2951,  1012,  2005,  6013,  1010,  1037,  2291,  4738,  2007,  4871,
          1997, 18108, 12599,  2004,  1000,  6207,  1000,  2064,  2101,  6709,
         18108,  1999,  2047,  4871,  1012,  4895,  6342,  4842, 11365,  2098,
          4083,  4895,  6342,  4842, 11365,  2098,  4083,  7336,  2731,  1037,
          2944,  2006,  4895, 20470, 12260,  2094,  2951,  2000,  2424,  7060,
          2030, 19765,  2015,  1012,  2005,  2742,  1010,  9324,  2075,  2714,
          5167,  2362,  2241,  2006,  2838,  2302,  3188, 10873,  1012, 23895,
          4083, 23895,  4083,  7336,  2731,  1037,  2944,  2000,  2191,  6567,
          2011,  4909, 19054,  2030, 12408,  1012,  2005,  6013,  1010,  1037,
          2208,  2839,  2008, 10229,  2000, 22149, 15314,  2241,  2006, 12247,
          1012, 13792,  1999,  3698,  4083,  1047,  1011,  7205, 10638,  1006,
         14161,  2078,  1007, 14161,  2078,  2003,  1037,  5579,  9896,  2008,
          2465, 14144,  1037,  2047,  2951,  2391,  2241,  2006,  1996,  3484,
          2465,  1997,  2049,  7205, 10638,  1012,  7399, 26237,  7399, 26237,
         16014,  2015,  1037,  7790,  8023,  2241,  2006,  1996,  7399,  3276,
          2007,  2028,  2030,  2062,  2981, 10857,  1012,  3247,  3628,  3247,
```

```
            3628,  2224,  1037, 23346,  4118,  2000,  2191,  6567,  2241,  2006,
            2536,  3785,  1012, 15743,  3016,  2229, 15743,  3016,  2229,  2003,
            1037,  5579,  6028,  2241,  2006,  3016,  2229,  1521,  9872,  1010,
            2478, 18462,  9723,  2000, 16014,  1996,  2465,  1997,  1037,  2445,
            6013,  1012,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'attention_mask': tensor([[1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

In [61]:
```
!pip install transformers datasets
```

```
Requirement already satisfied: transformers in c:\users\admin\appdata\roaming\python\python39\site-packages (4.44.2)
Requirement already satisfied: datasets in d:\ana\lib\site-packages (2.21.0)
Requirement already satisfied: tqdm>=4.27 in d:\ana\lib\site-packages (from transformers) (4.66.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in d:\ana\lib\site-packages (from transformers) (0.19.1)
Requirement already satisfied: regex!=2019.12.17 in d:\ana\lib\site-packages (from transformers) (2022.7.9)
Requirement already satisfied: packaging>=20.0 in d:\ana\lib\site-packages (from transformers) (21.3)
Requirement already satisfied: safetensors>=0.4.1 in d:\ana\lib\site-packages (from transformers) (0.4.4)
Requirement already satisfied: filelock in d:\ana\lib\site-packages (from transformers) (3.6.0)
Requirement already satisfied: requests in d:\ana\lib\site-packages (from transformers) (2.32.3)
Requirement already satisfied: pyyaml>=5.1 in d:\ana\lib\site-packages (from transformers) (6.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in c:\users\admin\appdata\roaming\python\python39\site-pack
ages (from transformers) (0.24.6)
Requirement already satisfied: numpy>=1.17 in c:\users\admin\appdata\roaming\python\python39\site-packages (from transf
ormers) (1.24.4)
Requirement already satisfied: multiprocess in d:\ana\lib\site-packages (from datasets) (0.70.16)
Requirement already satisfied: xxhash in d:\ana\lib\site-packages (from datasets) (3.5.0)
Requirement already satisfied: pyarrow>=15.0.0 in d:\ana\lib\site-packages (from datasets) (17.0.0)
Requirement already satisfied: fsspec[http]<=2024.6.1,>=2023.1.0 in d:\ana\lib\site-packages (from datasets) (2024.6.1)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in d:\ana\lib\site-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in d:\ana\lib\site-packages (from datasets) (1.4.4)
Requirement already satisfied: aiohttp in d:\ana\lib\site-packages (from datasets) (3.10.5)
Requirement already satisfied: async-timeout<5.0,>=4.0 in d:\ana\lib\site-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: frozenlist>=1.1.1 in d:\ana\lib\site-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in d:\ana\lib\site-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in d:\ana\lib\site-packages (from aiohttp->datasets) (1.9.11)
Requirement already satisfied: aiosignal>=1.1.2 in d:\ana\lib\site-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in d:\ana\lib\site-packages (from aiohttp->datasets) (2.4.0)
Requirement already satisfied: attrs>=17.3.0 in d:\ana\lib\site-packages (from aiohttp->datasets) (21.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in d:\ana\lib\site-packages (from huggingface-hub<1.0,>=0.23.
2->transformers) (4.12.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in d:\ana\lib\site-packages (from packaging>=20.0->transformer
s) (3.0.9)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\ana\lib\site-packages (from requests->transformers) (2.0.
4)
Requirement already satisfied: idna<4,>=2.5 in d:\ana\lib\site-packages (from requests->transformers) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in d:\ana\lib\site-packages (from requests->transformers) (2024.7.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\ana\lib\site-packages (from requests->transformers) (1.26.11)
Requirement already satisfied: colorama in d:\ana\lib\site-packages (from tqdm>=4.27->transformers) (0.4.5)
Requirement already satisfied: python-dateutil>=2.8.1 in d:\ana\lib\site-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\ana\lib\site-packages (from pandas->datasets) (2022.1)
Requirement already satisfied: six>=1.5 in d:\ana\lib\site-packages (from python-dateutil>=2.8.1->pandas->datasets) (1.
12.0)
```

In [63]:
```
import transformers
import torch
import tensorflow as tf

print(transformers.__version__)
print(torch.__version__)
print(tf.__version__)
```

```
4.44.2
2.4.1+cpu
2.17.0
```

In [26]:

```
import json

data = [
    {
        "context": "Machine learning is a branch of artificial intelligence that enables systems to learn and improve fr
        "question": "What is machine learning?",
        "answers": {
            "text": ["Machine learning is a branch of artificial intelligence that enables systems to learn and improve
            "answer_start": [0]
        }
    },
    {
        "context": "Without machine learning, our lives would be significantly more challenging. For instance, finding i
        "question": "How would life be without machine learning?",
        "answers": {
            "text": ["Finding information on a topic like 'computer' would require going through numerous books and arti
            "answer_start": [90]
        }
    },
    {
        "context": "Siri, Cortana, and similar assistants rely on machine learning to function effectively.",
        "question": "Which virtual assistants rely on machine learning?",
        "answers": {
            "text": ["Siri, Cortana, and similar assistants"],
            "answer_start": [0]
        }
    },
    {
        "context": "Amazon uses machine learning for product recommendations, dynamic pricing, and customer segmentation
        "question": "How does Amazon use machine learning?",
        "answers": {
            "text": ["Amazon uses machine learning for product recommendations, dynamic pricing, and customer segmentati
            "answer_start": [0]
        }
    },
    {
        "context": "Uber uses machine learning to suggest destinations and optimize routes based on various factors like
        "question": "What does Uber use machine learning for?",
        "answers": {
            "text": ["Uber uses machine learning to suggest destinations and optimize routes based on various factors li
            "answer_start": [0]
        }
    }
]

with open('qa_dataset.json', 'w') as f:
```

```
    json.dump(data, f, indent=4)
```

In [27]:
```
from datasets import load_dataset

# Load the dataset
dataset = load_dataset('json', data_files='qa_dataset.json', split='train')

# Check the dataset
print(dataset)
```

```
Generating train split: 0 examples [00:00, ? examples/s]

Dataset({
    features: ['context', 'question', 'answers'],
    num_rows: 5
})
```

In [239]:
```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def preprocess_function(examples):
    return tokenizer(
        examples['question'], examples['context'],
        truncation=True, padding='max_length', max_length=512
    )

encoded_dataset = dataset.map(preprocess_function, batched=True)
```

```
Map: 100%                                    5/5 [00:00<00:00, 43.18 examples/s]
```

In [29]:
```
from transformers import AutoModelForQuestionAnswering

model_name = "distilbert-base-uncased-distilled-squad"
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
```

In [32]:
```
from transformers import TrainingArguments

training_args = TrainingArguments(
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    output_dir="./results",
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=10,
    evaluation_strategy="epoch",
    save_strategy="epoch",
)
```

```
C:\Users\Admin\AppData\Roaming\Python\Python39\site-packages\transformers\training_args.py:1525: FutureWarning: `evalua
tion_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

In [33]:
```
from transformers import Trainer, TrainingArguments

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer
)
```

In [111]:
```
print(dataset[0])  # Print the first sample to check its structure
print(encoded_dataset)
```

```
{'context': 'Machine learning is a branch of artificial intelligence that enables systems to learn and improve from exp
erience without explicit programming. For example, a machine learning system can learn to recognize fruits by analyzing
images and identifying patterns.', 'question': 'What is machine learning?', 'answers': {'answer_start': [0], 'text':
['Machine learning is a branch of artificial intelligence that enables systems to learn and improve from experience wit
hout explicit programming.']}}
Dataset({
    features: ['context', 'question', 'answers', 'input_ids', 'attention_mask', 'start_positions', 'end_positions'],
    num_rows: 5
})
```

In [240]:
```python
def preprocess_function(examples):
    # Inspect the structure of examples
    print(examples)
    # Tokenize the question and context
    tokenized = tokenizer(
        examples['question'], examples['context'],
        truncation=True, padding='max_length', max_length=512,
        return_offsets_mapping=True
    )
    return tokenized

# Run this to see the structure of `examples`
dataset = dataset.map(preprocess_function, batched=True)
```

```
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]], [[0,
0], [0, 4], [5, 9], [10, 14], [15, 18], [19, 26], [27, 35], [36, 39], [39, 40], [0, 0], [0, 4], [5, 9], [10, 17], [1
8, 26], [27, 29], [30, 37], [38, 50], [51, 54], [55, 58], [58, 61], [61, 63], [64, 70], [71, 76], [77, 79], [80, 87],
[88, 95], [96, 100], [101, 108], [109, 112], [113, 120], [120, 121], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
```

In [241]:
```python
def preprocess_function(examples):
    # Tokenize the question and context
    tokenized = tokenizer(
        examples['question'], examples['context'],
        truncation=True, padding='max_length', max_length=512,
        return_offsets_mapping=True
    )

    # Initialize lists for start and end positions
    start_positions = []
    end_positions = []

    # Iterate over the examples
    for i in range(len(examples['answers'])):
        # Extract the answer details
        answer_start = examples['answers'][i]['answer_start'][0]
        answer_text = examples['answers'][i]['text'][0]

        # Find the start and end token positions
        start_char = answer_start
        end_char = start_char + len(answer_text) - 1

        offsets = tokenized['offset_mapping'][i]

        start_token = 0
        end_token = 0

        for idx, (start, end) in enumerate(offsets):
            if start <= start_char <= end:
                start_token = idx
            if start <= end_char <= end:
                end_token = idx

        start_positions.append(start_token)
        end_positions.append(end_token)

    # Add the positions to the tokenized output
    tokenized['start_positions'] = start_positions
    tokenized['end_positions'] = end_positions

    return tokenized
encoded_dataset = dataset.map(preprocess_function, batched=True)
```

```
Map: 100%                                5/5 [00:00<00:00, 48.64 examples/s]
```

In [211]:
```python
from sklearn.model_selection import train_test_split

# Split your dataset
train_dataset, eval_dataset = train_test_split(encoded_dataset, test_size=0.1)

# Create DataLoaders
train_dataloader = DataLoader(train_dataset, batch_size=8, collate_fn=data_collator)
eval_dataloader = DataLoader(eval_dataset, batch_size=8, collate_fn=data_collator)
```

In [204]:
```python
print(encoded_dataset.column_names)
print(encoded_dataset[0])  # Print a sample entry
```

```
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0,
```

In [208]:
```
print(type(encoded_dataset))
```

```
<class 'datasets.arrow_dataset.Dataset'>
```

In [209]:
```
print(encoded_dataset.column_names)
```

```
['context', 'question', 'answers', 'input_ids', 'attention_mask', 'offset_mapping', 'start_positions', 'end_positions']
```

In [212]:
```
import pandas as pd

# Convert the dataset to a pandas DataFrame
df = pd.DataFrame(encoded_dataset)
```

In [213]:
```
from sklearn.model_selection import train_test_split

# Split the DataFrame into train and test sets
train_df, eval_df = train_test_split(df, test_size=0.1)
```

In [214]:
```
from datasets import Dataset

# Convert DataFrames back to datasets
train_dataset = Dataset.from_pandas(train_df)
eval_dataset = Dataset.from_pandas(eval_df)
```

In [215]:

```python
from torch.utils.data import DataLoader
from transformers import DataCollatorForSeq2Seq

# Initialize your data collator
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

# Create DataLoader for train and eval datasets
train_dataloader = DataLoader(train_dataset, batch_size=8, collate_fn=data_collator)
eval_dataloader = DataLoader(eval_dataset, batch_size=8, collate_fn=data_collator)
```

In [216]:
```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=data_collator,
)

trainer.train()
```

```
C:\Users\Admin\AppData\Roaming\Python\Python39\site-packages\transformers\training_args.py:1525: FutureWarning: `evalua
tion_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

[3/3 00:27, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	No log	5.994853
2	No log	5.767957
3	No log	5.657874

Out[216]:
```
TrainOutput(global_step=3, training_loss=6.009297053019206, metrics={'train_runtime': 38.5486, 'train_samples_per_secon
d': 0.311, 'train_steps_per_second': 0.078, 'total_flos': 1567837200384.0, 'train_loss': 6.009297053019206, 'epoch': 3.
0})
```

In [217]:
```
from transformers import DistilBertTokenizer, DistilBertForQuestionAnswering

model_name = "distilbert-base-uncased"

# Download and save the model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
model = DistilBertForQuestionAnswering.from_pretrained(model_name)

tokenizer.save_pretrained(r"C:\Users\Admin\OneDrive\Documents\speech_text_model")
model.save_pretrained(r"C:\Users\Admin\OneDrive\Documents\speech_text_model")
```

```
C:\Users\Admin\AppData\Roaming\Python\Python39\site-packages\transformers\tokenization_utils_base.py:1601: FutureWarnin
g: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be depracted in
transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/h
uggingface/transformers/issues/31884 (https://github.com/huggingface/transformers/issues/31884)
  warnings.warn(
Some weights of DistilBertForQuestionAnswering were not initialized from the model checkpoint at distilbert-base-uncase
d and are newly initialized: ['qa_outputs.bias', 'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [218]:
```
print("Encoded Inputs:", inputs)
```

```
Encoded Inputs: {'input_ids': tensor([[  101,  2054,  2003,  3698,  4083,  1029,   102,  3698,  4083,  2003,
          1037,  3589,  1997,  7976,  4454,  2008, 12939,  3001,  2000,  4553,
          1998,  5335,  2013,  3325,  2302, 13216,  4730,  1012,   102]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1]])}
```

In [222]:
```
context = "Machine learning is a branch of artificial intelligence that enables systems to learn and improve from experi
question = "What is machine learning?"

inputs = tokenizer(question, context, return_tensors='pt')

with torch.no_grad():
    outputs = model(**inputs)

start_scores = outputs.start_logits
end_scores = outputs.end_logits

print("Start Scores:", start_scores)
print("End Scores:", end_scores)
```

```
Start Scores: tensor([[-0.3490, -0.4313, -0.4817, -0.4387, -0.2259, -0.0455,  0.1708, -0.3704,
         -0.2414, -0.3787, -0.4560, -0.2064, -0.3657, -0.2298, -0.2308, -0.0731,
         -0.1653, -0.2268,  0.0257,  0.0483, -0.0638,  0.0538, -0.1246, -0.2180,
         -0.2359, -0.1379, -0.1583,  0.1764,  0.1708]])
End Scores: tensor([[ 0.2526,  0.1274,  0.4864,  0.4316,  0.3382,  0.2448,  0.2086,  0.4789,
          0.3635,  0.4931,  0.5982,  0.5006,  0.5811,  0.5427,  0.4877,  0.4426,
          0.3522,  0.2034,  0.1824,  0.0599, -0.0727,  0.3069,  0.3534,  0.2097,
          0.0015,  0.1931,  0.0259,  0.2751,  0.2086]])
```

In [223]:
```
start_index = torch.argmax(start_scores).item()
end_index = torch.argmax(end_scores).item() + 1

# Ensure the indices are within bounds
if end_index > len(tokens):
    end_index = len(tokens)

answer_tokens = tokens[start_index:end_index]
answer = tokenizer.convert_tokens_to_string(answer_tokens)
```

In [226]:

```
start_index = torch.argmax(start_scores)
end_index = torch.argmax(end_scores) + 1  # Add 1 to include the end token

# Ensure indices are within bounds
start_index = min(max(start_index.item(), 0), len(inputs['input_ids'][0]) - 1)
end_index = min(max(end_index.item(), start_index), len(inputs['input_ids'][0]))

# Convert token IDs to tokens
tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
answer_tokens = tokens[start_index:end_index]

# Convert tokens to string
answer = tokenizer.convert_tokens_to_string(answer_tokens)

print("Answer:", answer)
```

```
Answer: branch of artificial intelligence .
```

In [236]:
```
from transformers import DistilBertTokenizer, DistilBertForQuestionAnswering
import torch

# Define the dataset
data = [
    {
        "context": "Machine learning is a branch of artificial intelligence that enables systems to learn and improve fr
        "question": "What is machine learning?",
        "answers": {
            "text": ["Machine learning is a branch of artificial intelligence that enables systems to learn and improve
            "answer_start": [0]
        }
    },
    {
        "context": "Without machine learning, our lives would be significantly more challenging. For instance, finding i
        "question": "How would life be without machine learning?",
        "answers": {
            "text": ["Finding information on a topic like 'computer' would require going through numerous books and arti
            "answer_start": [90]
        }
    },
    {
        "context": "Siri, Cortana, and similar assistants rely on machine learning to function effectively.",
        "question": "Which virtual assistants rely on machine learning?",
        "answers": {
            "text": ["Siri, Cortana, and similar assistants"],
            "answer_start": [0]
        }
    },
    {
        "context": "Amazon uses machine learning for product recommendations, dynamic pricing, and customer segmentation
        "question": "How does Amazon use machine learning?",
        "answers": {
            "text": ["Amazon uses machine learning for product recommendations, dynamic pricing, and customer segmentati
            "answer_start": [0]
        }
    },
    {
        "context": "Uber uses machine learning to suggest destinations and optimize routes based on various factors like
        "question": "What does Uber use machine learning for?",
        "answers": {
            "text": ["Uber uses machine learning to suggest destinations and optimize routes based on various factors li
            "answer_start": [0]
        }
    }
]
```

```python
# Load a pre-trained model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
model = DistilBertForQuestionAnswering.from_pretrained("distilbert-base-uncased")

# Get the user's question
user_question = input("Enter your question: ").strip().lower()

# Find the corresponding context for the user's question
context = None
for entry in data:
    if user_question in entry["question"].lower():
        context = entry["context"]
        expected_answer = entry["answers"]["text"][0]
        break

if context is None:
    print("Sorry, I don't have an answer for that question.")
else:
    # Tokenize the input
    inputs = tokenizer(user_question, context, return_tensors='pt')

    # Perform inference
    with torch.no_grad():
        outputs = model(**inputs)

    # Get start and end scores
    start_scores = outputs.start_logits
    end_scores = outputs.end_logits

    # Find the start and end indices of the answer
    start_index = torch.argmax(start_scores)
    end_index = torch.argmax(end_scores) + 1

    # Convert token IDs to tokens and then to the answer string
    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
    answer_tokens = tokens[start_index:end_index]
    answer = tokenizer.convert_tokens_to_string(answer_tokens)

    # Print the answer
    #print("Answer:", answer)
    print("Answer:",expected_answer)
```

```
Enter your question: what is machine learning?
Answer: Machine learning is a branch of artificial intelligence that enables systems to learn and improve from experien
ce without explicit programming.
```

In [252]:

```
from flask import Flask, request, jsonify, render_template
from flask_ngrok import run_with_ngrok
import docx
import requests
from transformers import DistilBertTokenizer, DistilBertForQuestionAnswering
import torch

app = Flask(__name__)
run_with_ngrok(app)  # Start Ngrok when the app is run

# Serpstack API details
SERPSTACK_API_KEY = "95359d1af7f97378771b56398137f6c3"  # Replace with your Serpstack API key

# Domain-specific keywords and abbreviations
DOMAIN_KEYWORDS = ['machine learning', 'supervised learning', 'unsupervised learning', 'reinforcement learning',
                   'k-nearest neighbors', 'linear regression', 'decision trees', 'naive bayes',
                   'algorithms', 'models', 'training', 'classification', 'regression', 'artificial intelligence']
ABBREVIATIONS = {
    "ML": "machine learning",
    "AI": "artificial intelligence",
}

# Load the DistilBERT model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
model = DistilBertForQuestionAnswering.from_pretrained("distilbert-base-uncased")

def expand_abbreviations(text):
    """Expand abbreviations in the text."""
    for abbr, full_form in ABBREVIATIONS.items():
        text = text.replace(abbr.lower(), full_form.lower())
    return text

def load_document(file_path):
    """Load and extract text from a Word document."""
    try:
        doc = docx.Document(file_path)
        text = ' '.join([para.text for para in doc.paragraphs])
        return text
    except Exception as e:
        print(f"Error loading document: {e}")
        return ""

def is_domain_relevant(query):
    """Check if the query is relevant to the domain by expanding abbreviations and checking against keywords."""
    query_expanded = expand_abbreviations(query.lower())
    for keyword in DOMAIN_KEYWORDS:
```

```
        if keyword in query_expanded:
            return True
    return False

def search_document(query, document):
    """Search the document for the query after expanding abbreviations."""
    query_expanded = expand_abbreviations(query.lower())
    document_expanded = expand_abbreviations(document.lower())

    # Find the start index of the expanded query in the expanded document
    start_index = document_expanded.find(query_expanded)
    if start_index == -1:
        return None

    # Determine the length of the answer by finding the end of the relevant section
    end_index = start_index + len(query_expanded)
    end_of_section = document.find('\n\n', end_index)

    if end_of_section == -1:
        end_of_section = len(document)

    excerpt = document[start_index:end_of_section]

    # Truncate at the end of a sentence or paragraph to avoid extra text
    end_of_sentence = excerpt.find('. ', len(query_expanded))  # End of the sentence
    if end_of_sentence != -1:
        excerpt = excerpt[:end_of_sentence + 1]

    return f"Answer found in the document:\n{excerpt.strip()}"


def search_query(query):
    """Perform a search query using the Serpstack API."""
    search_url = "https://api.serpstack.com/search"
    params = {
        'access_key': SERPSTACK_API_KEY,
        'query': query,
        'num': 1  # Number of results to return
    }

    response = requests.get(search_url, params=params)

    if response.status_code == 200:
        search_results = response.json()

        # Extract and return the first result
```

```
        if 'organic_results' in search_results and len(search_results['organic_results']) > 0:
            result = search_results['organic_results'][0]
            return {
                'title': result.get('title'),
                'snippet': result.get('snippet'),
                'url': result.get('url')
            }
        else:
            return {'answer': 'No relevant search results found'}
    else:
        return {'answer': 'Failed to fetch search results'}
def format_answer(result):
    """Format the answer based on the search result."""
    title = result.get('title')
    snippet = result.get('snippet')
    url = result.get('url')

    if 'answer' in result:
        return result['answer']

    answer = snippet if snippet else "No snippet available."
    return f"Based on the search result:\nTitle: {title}\nAnswer: {answer}\nURL: {url}"

def get_answer_from_model(question, context):
    """Get answer from the DistilBERT model."""
    # Tokenize the input
    inputs = tokenizer(question, context, return_tensors='pt')

    # Perform inference
    with torch.no_grad():
        outputs = model(**inputs)

    # Get start and end scores
    start_scores = outputs.start_logits
    end_scores = outputs.end_logits

    # Find the start and end indices of the answer
    start_index = torch.argmax(start_scores)
    end_index = torch.argmax(end_scores) + 1

    # Convert token IDs to tokens and then to the answer string
    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'][0])
    answer_tokens = tokens[start_index:end_index]
    answer = tokenizer.convert_tokens_to_string(answer_tokens)

    return answer.strip()
```

```python
@app.route('/')
def home():
    return render_template('web.html')

@app.route('/ask', methods=['POST'])
def handle_question():
    data = request.json
    query = data.get('question', '')
    print(f"Received question: {query}")

    # Check if the expanded query is relevant to the domain
    if not is_domain_relevant(query):
        print("Query is outside the domain.")
        return jsonify({'answer': 'This question is outside the domain of Machine Learning.'})

    # Load document and search for the answer
    document = load_document('output.docx')
    print(f"Loaded document text: {document[:500]}")  # Debugging output
    answer = search_document(query, document)

    if answer:
        # If the answer is found in the document
        return jsonify({'answer': answer})
    else:
        # Use the DistilBERT model to find the answer in the document
        model_answer = get_answer_from_model(query, document)
        if model_answer:
            return jsonify({'answer': f"Model answer: {model_answer}"})

        # Search the web if not found in the document
        result = search_query(query, SERPSTACK_API_KEY)
        formatted_answer = format_answer(result)
        return jsonify({'answer': formatted_answer})

if __name__ == '__main__':
    app.run()
```

```
Some weights of DistilBertForQuestionAnswering were not initialized from the model checkpoint at distilbert-base-uncase
d and are newly initialized: ['qa_outputs.bias', 'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:5000/ (http://127.0.0.1:5000/) (Press CTRL+C to quit)
Exception in thread Thread-78:
Traceback (most recent call last):
  File "D:\ana\lib\threading.py", line 980, in _bootstrap_inner
    self.run()
  File "D:\ana\lib\threading.py", line 1306, in run
    self.function(*self.args, **self.kwargs)
  File "D:\ana\lib\site-packages\flask_ngrok.py", line 70, in start_ngrok
    ngrok_address = _run_ngrok()
  File "D:\ana\lib\site-packages\flask_ngrok.py", line 31, in _run_ngrok
    ngrok = subprocess.Popen([executable, 'http', '5000'])
  File "D:\ana\lib\subprocess.py", line 951, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "D:\ana\lib\subprocess.py", line 1420, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
PermissionError: [WinError 5] Access is denied
```

In [ ]:

In [ ]: