

```
!pip install transformers datasets faiss-cpu google-cloud-storage
```

```

Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.9.0, >=2023.1.0 (from fsspec[http]<=2024.9.0, >=2023.1.0->datasets)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.10.10)
Requirement already satisfied: google-auth<3.0dev, >=1.25.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-storage) (2.27.0)
Requirement already satisfied: google-api-core!=2.0.*, !=2.1.*, !=2.2.*, !=2.3.0, <3.0.0dev, >=1.31.5 in /usr/local/lib/python3.10/dist-packages (from google-clou
Requirement already satisfied: google-cloud-core<3.0dev, >=2.3.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-storage) (2.4.1)
Requirement already satisfied: google-resumable-media>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from google-cloud-storage) (2.7.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0, >=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: yarl<2.0, >=1.12.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.17.1)
Requirement already satisfied: async-timeout<5.0, >=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: googleapis-common-protos<2.0.dev0, >=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core!=2.0.*, !=2.1.*, !=2
Requirement already satisfied: protobuf!=3.20.0, !=3.20.1, !=4.21.0, !=4.21.1, !=4.21.2, !=4.21.3, !=4.21.4, !=4.21.5, <6.0.0.dev0, >=3.19.5 in /usr/local/lib/python3
Requirement already satisfied: proto-plus<2.0.0dev, >=1.22.3 in /usr/local/lib/python3.10/dist-packages (from google-api-core!=2.0.*, !=2.1.*, !=2.2.*, !=2.3.0, <
Requirement already satisfied: cachetools<6.0, >=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0dev, >=1.25.0->google-cloud-storage) (5.
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0dev, >=1.25.0->google-cloud-storage) (0.4
Requirement already satisfied: rsa<5, >=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0dev, >=1.25.0->google-cloud-storage) (4.9)
Requirement already satisfied: google-crc32c<2.0dev, >=1.0 in /usr/local/lib/python3.10/dist-packages (from google-resumable-media>=2.3.2->google-cloud-storag
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0, >=0.23.2->transformers) (4.12.
Requirement already satisfied: charset-normalizer<4, >=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4.0)
Requirement already satisfied: idna<4, >=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3, >=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.30)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: pyasn1<0.7.0, >=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3.0dev, >=1.25.0->goog
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.16.0)

```

gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is incompatible.

Successfully installed datasets-3.1.0 dill-0.3.8 faiss-cpu-1.9.0 fsspec-2024.9.0 multiprocess-0.70.16 xxhash-3.5.0

```
import torch
from transformers import DPRContextEncoder, DPRQuestionEncoder, DPRReader, DPRReaderTokenizer
import faiss
import numpy as np

# Initialize Tokenizer and Models for the encoder (Question Encoder) and generator (Reader for QA)
context_encoder = DPRContextEncoder.from_pretrained("facebook/dpr-ctx_encoder-single-nq-base")
question_encoder = DPRQuestionEncoder.from_pretrained("facebook/dpr-question_encoder-single-nq-base")
reader_model = DPRReader.from_pretrained("facebook/dpr-reader-single-nq-base")
reader_tokenizer = DPRReaderTokenizer.from_pretrained("facebook/dpr-reader-single-nq-base")

# FAISS Index Setup
def build_faiss_index(documents):
    """Build FAISS index for the context documents."""
    context_embeddings = []
    for doc in documents:
        # Tokenize and process each document
        inputs = reader_tokenizer(doc, return_tensors='pt', truncation=True, padding=True, max_length=512)
        with torch.no_grad():
            embeddings = context_encoder(**inputs).pooler_output
        context_embeddings.append(embeddings.cpu().numpy())

    # Convert the embeddings list to a numpy array
    context_embeddings = np.vstack(context_embeddings)

    # FAISS index for efficient retrieval
    index = faiss.IndexFlatL2(context_embeddings.shape[1]) # Flat L2 index
    index.add(context_embeddings)
    return index

# Function to retrieve the top-k relevant documents based on the query
def retrieve_context(query, index, documents, k=3):
    """Retrieve top-k documents relevant to the query."""
    # Encode the query with the question encoder
    inputs = reader_tokenizer(query, return_tensors="pt", truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        query_embedding = question_encoder(**inputs).pooler_output.cpu().numpy()

    # Perform the search on the FAISS index
    D, I = index.search(query_embedding, k) # D = distances, I = indices of retrieved documents

    # Check if indices are valid and retrieve documents
    retrieved_docs = []
    for i in I[0]:
        if i < len(documents): # Check to avoid out-of-bounds access
            retrieved_docs.append(documents[i])
        else:
            print(f"Warning: Retrieved index {i} is out of range for documents list.")
    return retrieved_docs

# Function to generate the answer using the DPRReader (Facebook's model for QA)
```

```

def generate_answer(query, retrieved_docs):
    """Generate an answer to the query using the DPRReader model."""
    # Combine retrieved documents into a context for the model
    context = " ".join(retrieved_docs)
    print("Context: ", context)

    # Encode the query and context to generate the answer
    inputs = reader_tokenizer(query, context, return_tensors="pt", truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        outputs = reader_model(**inputs)

    # Get the best answer (usually in the 'start' and 'end' logits)
    start_idx = outputs.start_logits.argmax()
    end_idx = outputs.end_logits.argmax()

    # Decode the answer from the tokens
    answer_tokens = inputs.input_ids[0][start_idx:end_idx + 1]
    answer = reader_tokenizer.decode(answer_tokens, skip_special_tokens=True)
    return answer

# Example documents
documents = [
    "Apple's stock price has seen significant fluctuations recently. The company reached a high of $300 per share before dipping, driven by challenges in the glo",
    "Tesla's stock has reached new highs, now trading at $900 per share. Strong earnings reports and growing optimism about the electric vehicle market have driv",
    "Amazon's stock saw a rise to $150 in March 2024, but concerns about market volatility have led to mixed predictions. Some analysts believe Amazon will conti",
    "The Federal Reserve raised interest rates by 0.25% in early 2024, causing increased volatility in the tech sector. Companies like Apple, Tesla, and Amazon h",
    "Microsoft stock price has surged following strong performance in its cloud business, reaching $380 per share. Analysts are optimistic about the company's gr",
    "Meta stock price has dropped 25 % after announcing major layoffs, but the company remains committed to investing in the metaverse. Investors are divided on"
]

queries = [
    "What is the recent trend in Apple's stock price?",
    "How did Tesla's stock perform after the latest earnings report?",
    "How much is federal reserve interest rate in early 2024?"
]


# Build FAISS index from the documents
index = build_faiss_index(documents)

for query in queries:

    # Get the retrieved context documents
    retrieved_docs = retrieve_context(query, index, documents)

    # Generate the answer using the DPRReader model
    answer = generate_answer(query, retrieved_docs)
    print('Query: ', query)
    print("Answer:", answer)

```

 Some weights of the model checkpoint at facebook/dpr-ctx_encoder-single-nq-base were not used when initializing DPRContextEncoder: ['ctx_encoder.bert_model.p
- This IS expected if you are initializing DPRContextEncoder from the checkpoint of a model trained on another task or with another architecture (e.g. initia
- This IS NOT expected if you are initializing DPRContextEncoder from the checkpoint of a model that you expect to be exactly identical (initializing a BertF
Some weights of the model checkpoint at facebook/dpr-question_encoder-single-nq-base were not used when initializing DPRQuestionEncoder: ['question_encoder.b

- This IS expected if you are initializing DPRQuestionEncoder from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification)
- This IS NOT expected if you are initializing DPRQuestionEncoder from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification)

Some weights of the model checkpoint at facebook/dpr-reader-single-nq-base were not used when initializing DPRReader: ['span_predictor.encoder.bert_model.pooler.dense.weight', 'span_predictor.encoder.bert_model.pooler.dense.bias']

- This IS expected if you are initializing DPRReader from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a DPRQuestionEncoder)
- This IS NOT expected if you are initializing DPRReader from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification)

The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result in unexpected tokenization. The tokenizer class you load from this checkpoint is 'DPRQuestionEncoderTokenizer'.

The class this function is called from is 'DPRReaderTokenizer'.

Context: Apple's stock price has seen significant fluctuations recently. The company reached a high of \$300 per share before dipping, driven by challenges in the market.

Query: What is the recent trend in Apple's stock price?

Answer: seen significant fluctuations

Context: Tesla's stock has reached new highs, now trading at \$900 per share. Strong earnings reports and growing optimism about the electric vehicle market.

Query: How did Tesla's stock perform after the latest earnings report?

Answer: \$ 900 per share

Context: The Federal Reserve raised interest rates by 0.25% in early 2024, causing increased volatility in the tech sector. Companies like Apple, Tesla, and Amazon have seen their stock prices fluctuate.

Query: How much is federal reserve interest rate in early 2024?

Answer: 0. 25 %

Start coding or [generate](#) with AI.

```
import unittest
import numpy as np

class TestRAGPipeline(unittest.TestCase):

    def setUp(self):
        # Example documents (same as in your code)
        self.documents = [
            "Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms, including object-oriented, functional, and imperative programming.",
            "The Amazon rainforest is a large tropical rainforest in South America, known for its biodiversity and vast ecosystems. It is often referred to as the 'lungs of the planet' due to its role in producing oxygen.",
            "The capital of Japan is Tokyo. It is known for its towers, including the Tokyo Skytree, and its busy districts like Shibuya and Shinjuku.",
            "The Eiffel Tower is a wrought-iron lattice tower on the Champ de Mars in Paris, France. It was named after the engineer Gustave Eiffel, whose company constructed it from 1889 to 1909.",
            "Tom is friend of jerry."
        ]

        self.index = build_faiss_index(self.documents)

    def test_build_faiss_index(self):
        # Check if the index is built correctly (e.g., it's not None and has the right dimension)
        self.assertIsNotNone(self.index)
        self.assertEqual(self.index.ntotal, len(self.documents))

    def test_retrieve_context(self):
        query = "What is capital of Japan?"
        retrieved_docs = retrieve_context(query, self.index, self.documents)
        self.assertEqual(len(retrieved_docs), 3) # Check if 3 documents are retrieved (k=3 by default)
        self.assertIn("The capital of Japan is Tokyo.", " ".join(retrieved_docs)) # Check if relevant document is in the result

    def test_generate_answer(self):
        query = "Who is friend of Tom?"
        retrieved_docs = retrieve_context(query, self.index, self.documents)
        answer = generate_answer(query, retrieved_docs)
        self.assertIn("jerry", answer)

        query = "What is capital of Japan?"
        retrieved_docs = retrieve_context(query, self.index, self.documents)
        answer = generate_answer(query, retrieved_docs)
```

```
answer = generate_answer(query, retrieved_docs)
self.assertIn("tokyo", answer) # Check if the answer contains the expected keyword.

query = "Who designed the Eiffel Tower?"
retrieved_docs = retrieve_context(query, self.index, self.documents)
answer = generate_answer(query, retrieved_docs)
self.assertIn("gustave eiffel", answer)

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```



```
.Context: Tom is friend of jerry. Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple pr
Context: The capital of Japan is Tokyo. It is known for its towers, including the Tokyo Skytree, and its busy districts like Shibuya and Shinjuku. Python is
Context: The Eiffel Tower is a wrought-iron lattice tower on the Champ de Mars in Paris, France. It was named after the engineer Gustave Eiffel, whose compa
..
```

Ran 3 tests in 4.089s

OK