Installations

```
!pip install spacy PyPDF2
!python -m spacy download en_core_web_sm
!pip install transformers sentence-transformers scikit-learn
```

Show hidden output

Dependencies

```
# Standard library imports
import os
import re
import json
from pathlib import Path
import string
from collections import Counter

# Third-party library imports
import numpy as np
import pandas as pd
import torch
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# PyPDF2 import for PDF processing
import PyPDF2

# HuggingFace transformers models and tokenizers
from transformers import (
    T5Tokenizer, T5ForConditionalGeneration,
    BartTokenizer, BartForConditionalGeneration, BartForQuestionAnswering,
    AutoTokenizer, AutoModelForQuestionAnswering
)
```

1. Load the Document and Clean the Data:

- Load PDF Documents: Read content from specified PDF files.
- Clean Text: Convert text to lowercase and remove unnecessary spaces.

```
# 1. Load spaCy with only the sentencizer (fastest for sentence splitting)
nlp = spacy.load("en_core_web_sm", disable=["ner", "tagger", "parser", "lemmatizer", "tok

# 2. Add the sentencizer pipe to the pipeline if it's not already present
if "sentencizer" not in nlp.pipe_names:
```

```
    nlp.add_pipe("sentencizer")

# 3. Verify the active components in the spaCy pipeline
nlp.pipe_names
```

```
['sentencizer']
```

```
# Define the function to read and clean multiple text or PDF documents
def read_and_clean_documents(file_paths: list[str]) -> dict[str, str]:
    """
    Reads and cleans multiple text or PDF documents.
    Returns a dictionary with document names as keys and cleaned text as values.
    """
    cleaned_docs = {}  # Dictionary to store cleaned documents.

    for path in file_paths:
        try:
            ext = os.path.splitext(path)[1].lower()  # Get file extension.

            # 1. Process PDF files
            if ext == '.pdf':
                with open(path, 'rb') as file:
                    reader = PyPDF2.PdfReader(file)
                    text = ""
                    for page in reader.pages:
                        page_text = page.extract_text()
                        if page_text:
                            text += page_text

            # 2. Process text files
            elif ext == '.txt':
                with open(path, 'r', encoding='utf-8') as file:
                    text = file.read()

            else:
                print(f"Unsupported file type: {ext}")
                continue

            # 3. Clean the extracted text
            text = text.lower()
            text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces.

            doc_name = Path(path).stem  # Get document name (without extension).
            cleaned_docs[doc_name] = text  # Add to dictionary.

        except Exception as e:
            print(f"Error processing {path}: {e}")

    # 4. Return the cleaned documents
    return cleaned_docs


# Example usage
file_paths = ['/content/Machine Learning.pdf',]
```

```
cleaned_data = read_and_clean_documents(file_paths)
cleaned_data
```

Show hidden output

Tokenization

```
def split_sentences(text: str) -> list[str]:
    """
    Splits text into sentences using spaCy
    """
    doc = nlp(text)
    sentences = [sent.text.strip() for sent in doc.sents]

    return [s for s in sentences if s]


def tokenize_with_overlap_from_dict(cleaned_data: dict[str, str], sentences_per_token: in
    """ Tokenizes documents into overlapping chunks of sentences """
    if sentences_per_token <= overlap:
        raise ValueError("sentences_per_token must be greater than overlap.")

    tokenized_data = {}
    step = sentences_per_token - overlap

    for doc_name, text in cleaned_data.items():
        sentences = split_sentences(text)
        tokens = [
            " ".join(sentences[start:start + sentences_per_token])
            for start in range(0, len(sentences), step)
        ]
        tokenized_data[doc_name] = tokens

    return tokenized_data


# Example: Tokenize cleaned data

tokenized_data = tokenize_with_overlap_from_dict(cleaned_data, sentences_per_token=3, ove

print(tokenized_data['Machine Learning'][3])
print(tokenized_data['Machine Learning'][4])
```

```
unsupervised learning, on the other hand, deals with unlabeled data and is used to un
key algorithms and techniques there are a variety of algorithms used across different
```

Indexer

```
# Step 1: Prepare corpus and metadata
def prepare_token_corpus(tokenized_data: dict[str, list[str]]) -> tuple[list[str], list[d
    # Flattens tokenized data into a corpus and creates associated metadata.
    corpus = []
    metadata = []

    for doc_name, tokens in tokenized_data.items():
        for idx, token in enumerate(tokens):
            corpus.append(token)
            metadata.append({
                "doc": doc_name,
                "token_index": idx,
                "text": token
            })

    return corpus, metadata


# Prepare the corpus and metadata

corpus, metadata = prepare_token_corpus(tokenized_data)
corpus[3], metadata[3]
```

Show hidden output

```
# Indexing step

def build_tfidf_index(corpus: list[str]) -> tuple[TfidfVectorizer, any]:
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(corpus)
    return vectorizer, tfidf_matrix


# Building of the TF-IDF index

vectorizer, tfidf_matrix = build_tfidf_index(corpus)
tfidf_matrix
```

```
<18x318 sparse matrix of type '<class 'numpy.float64'>'
        with 703 stored elements in Compressed Sparse Row format>
```

Retriever

```
def retrieve_top_k(query: str, vectorizer: TfidfVectorizer, tfidf_matrix, metadata: list[

    """Retrieves top-k relevant text chunks based on cosine similarity to the query."""

    query_vec = vectorizer.transform([query])
    similarities = cosine_similarity(query_vec, tfidf_matrix).flatten()
    top_indices = similarities.argsort()[::-1][:top_k]

    top_results = []
```

```
    for idx in top_indices:
        result = metadata[idx].copy()
        result["similarity"] = similarities[idx]
        top_results.append(result)

    return top_results


# Example: Retrieve relevant chunks for a query

query = "Which three main types of machine learning are mentioned?"
top_k = 2

# Retrieve top k tokens based on cosine similarity
top_results = retrieve_top_k(query, vectorizer, tfidf_matrix, metadata, top_k)
top_results
```

Show hidden output

Model Setup

```
# Template for extracting answers from context using a question
prompt_template = """
        Given the following context and question, extract the exact answer from the conte
        If the answer is not present, return 'Answer not found.'

        Context: {context}
        Question: {question}
        Answer:
"""
```

Model FLAN-T5 Answer Generation

```
tokenizer_flan = T5Tokenizer.from_pretrained("google/flan-t5-base")
model_flan = T5ForConditionalGeneration.from_pretrained("google/flan-t5-base")

def generate_answer_flan_t5(context, question):
    prompt = prompt_template.format(context=context, question=question)
    inputs = tokenizer_flan(prompt, return_tensors="pt", truncation=True)
    outputs = model_flan.generate(**inputs, max_length=100)
    return tokenizer_flan.decode(outputs[0], skip_special_tokens=True).strip()
```

Show hidden output

```
# Load model and tokenizer
tokenizer_bart = BartTokenizer.from_pretrained("facebook/bart-base")
model_bart = BartForConditionalGeneration.from_pretrained("facebook/bart-base")

def generate_answer_with_bart(context, question):
    # Clean the context before passing the content to the model
```

```
    # Remove unnecessary spaces or artifacts
    context_cleaned = context.strip()

    prompt = prompt_template.format(context=context_cleaned, question=question)

    # Tokenize input
    inputs = tokenizer_bart(prompt, return_tensors="pt", truncation=True, max_length=512)

    # Generate output
    outputs = model_bart.generate(**inputs, max_length=100, num_beams=4, early_stopping=T

    # Decode and return answer
    return tokenizer_bart.decode(outputs[0], skip_special_tokens=True).strip()
```

Show hidden output

BART QA (SQuADv2 Fine-tuned)

```
tokenizer_bart_squad = BartTokenizer.from_pretrained('a-ware/bart-squadv2')
model_bart_squad = BartForQuestionAnswering.from_pretrained('a-ware/bart-squadv2')

def generate_answer_with_bart_squadv2(context: str, question: str) -> str:
    # Generates span-based answer using BART fine-tuned on SQuADv2
    inputs = tokenizer_bart_squad(question, context, return_tensors='pt', truncation=True

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

    start_idx = torch.argmax(outputs.start_logits)
    end_idx = torch.argmax(outputs.end_logits)

    if end_idx < start_idx:
        return "Answer not found."

    answer = tokenizer_bart_squad.decode(inputs['input_ids'][0][start_idx : end_idx + 1],
    answer = answer.lstrip(string.punctuation + " ").strip()
    return answer
```

Show hidden output

FLAN-T5 QA (SQuADv2 Fine-tuned)

```
# Load tokenizer and model
tokenizer_flan_squad = AutoTokenizer.from_pretrained('sjrhuschlee/flan-t5-large-s
model_flan_squad = AutoModelForQuestionAnswering.from_pretrained('sjrhuschlee/fla

def generate_answer_with_flan_t5_squad2(context: str, question: str) -> str:
    # Uses FLAN-T5 fine-tuned on SQuADv2 for span-based QA
    inputs = tokenizer_flan_squad(f"{tokenizer_flan_squad.cls_token}{question}",

    with torch.no_grad():
        outputs = model_flan_squad(**inputs)
```

```
    start_idx = torch.argmax(outputs.start_logits)
    end_idx = torch.argmax(outputs.end_logits)

    if end_idx < start_idx:
        return "Answer not found."

    answer = tokenizer_flan_squad.decode(inputs['input_ids'][0][start_idx : end_i
    answer = answer.lstrip(string.punctuation + " ").strip()
    return answer
```

Show hidden output

Testing a sample question

```
context = """
ML has become integral in numerous real-world applications such as email filterin
Types of machine learning: Machine learning is typically divided into three main
In supervised learning, models are trained using labeled datasets where the desir
"""

question = "Which three main types of machine learning are mentioned?"


# Should return: "supervised learning, unsupervised learning, and reinforcement learning"

answer = generate_answer_flan_t5(context, question)
print(answer)
```

```
supervised learning, unsupervised learning, and reinforcement learning
```

```
answer = generate_answer_with_bart(context, question)
print(answer)
```

```
Question: Which is the correct answer? Answer:  Â  .  Given the following context
```

```
answer = generate_answer_with_flan_t5_squad2(context, question)
print(answer)
```

```
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformer
supervised learning, unsupervised learning, and reinforcement learning
```

```
answer = generate_answer_with_bart_squadv2(context, question)
print(answer)
```

```
supervised learning, unsupervised learning, and reinforcement learning
```

Evaluation Metrics

```
def normalize_answer(s: str) -> str:
    """
    Normalize text by:
    - Lowercasing
    - Removing extra spaces
    - Stripping leading/trailing whitespace
    """
    s = s.lower()
    s = re.sub(r'\s+', ' ', s)  # Collapse multiple whitespace into one
    return s.strip()


def exact_match_score(prediction: str, ground_truth: str) -> int:
    """
    Returns 1 if normalized prediction matches ground truth, else 0.
    """
    return int(normalize_answer(prediction) == normalize_answer(ground_truth))


def f1_score(prediction: str, ground_truth: str) -> float:
    """
    Calculates F1 score between prediction and ground truth using your cleaning rules.
    """
    pred_tokens = normalize_answer(prediction).split()
    gt_tokens = normalize_answer(ground_truth).split()

    pred_counts = Counter(pred_tokens)
    gt_counts = Counter(gt_tokens)

    common = set(pred_counts) & set(gt_counts)
    if not common:
        return 0.0

    true_positives = sum(min(pred_counts[token], gt_counts[token]) for token in common)
    precision = true_positives / len(pred_tokens) if pred_tokens else 0
    recall = true_positives / len(gt_tokens) if gt_tokens else 0

    return 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else
```

Build Model

```
# all documents
file_paths = ['/content/Machine Learning.pdf','/content/artificial intelligence.pdf',
              '/content/cybersecurity.pdf', '/content/social networking.pdf',
              '/content/web technology.pdf',]

# 1. Load the documents and clean the data
```

```
cleaned_data = read_and_clean_documents(file_paths)

# 2. Split the cleaned data with 3 sentences per token and an overlap of one sentence.
tokenized_data = tokenize_with_overlap_from_dict(cleaned_data, sentences_per_token=3, ove

# 3. Create the corpus of data and metadata
corpus, metadata = prepare_token_corpus(tokenized_data)

# 4. Get the tfidf matrix by feeding the corpus to the vectorizer and save the vectorizer
vectorizer, tfidf_matrix = build_tfidf_index(corpus)


tfidf_matrix
```

```
<83x948 sparse matrix of type '<class 'numpy.float64'>'
        with 2854 stored elements in Compressed Sparse Row format>
```

Prediction

```
def predict_answer_for_question(question: str, model_func, vectorizer, tfidf_matrix, meta
    """
    Predicts an answer for a single question using top-k relevant contexts from all docum

    Parameters:
    - question: the input question string
    - model_func: the model function (e.g., generate_answer_flan_t5)
    - vectorizer, tfidf_matrix, metadata: RAG components
    - top_k: number of chunks to retrieve

    Returns:
    - predicted answer string
    """
    # Retrieve top-k relevant contexts using the retrieve_top_k function
    top_results = retrieve_top_k(question, vectorizer, tfidf_matrix, metadata, top_k)

    # Extract only the "text" field from each dictionary in top_results
    context = " ".join([result["text"] for result in top_results])

    # Generate answer
    return model_func(context, question)



question = "What are the three types of machine learning?"

prediction = predict_answer_for_question(question, generate_answer_with_flan_t5_squad2, v

print(prediction, end='\n\n')
```

```
supervised learning, unsupervised learning, and reinforcement learning
```

```
def generate_predictions_for_dataset(test_dataset: pd.DataFrame, model_func, vectorizer,
    """
    Adds a 'Prediction' column to the dataset by applying the model to each question.

    Parameters:
    - test_dataset: DataFrame with at least 'Question' column
    - model_func: function to generate answer (e.g., generate_answer_flan_t5)
    - vectorizer, tfidf_matrix, metadata: RAG components
    - top_k: number of context chunks to retrieve per question

    Returns:
    - DataFrame with an added 'Prediction' column
    """
    predictions = []

    for idx, row in test_dataset.iterrows():
        question = row["Question"]

        # Call predict_answer_for_question to get the prediction for each question
        prediction = predict_answer_for_question(question, model_func, vectorizer, tfidf_

        # Append the prediction to the list
        predictions.append(prediction)

    # Create a copy of the dataset and add the 'Prediction' column
    test_dataset = test_dataset.copy()
    test_dataset["Prediction"] = predictions

    return test_dataset


test_file = '/content/test_dataset.csv'

test_dataset = pd.read_csv(test_file)
test_dataset.head(10)
```

	Question	Answer	Document
0	what are the common evaluation metrics in supe...	Accuracy, Precison, recall, F1 score and roc-auc	Machine Learning
1	What the the different types in ML?	Supervised Learning, Unsupervised Learning and...	Machine Learning
2	What are foundational techniques used in predi...	Linear regression and logistic regression	Machine Learning
3	Which technique is used to validate the model'...	Cross-validation	Machine Learning
4	How is ML used in healthcare?	disease prediction, medical imaging, and perso...	Machine Learning
5	What are the things that impacts the model per...	Data quality, quantity, and representativeness	Machine Learning
6	How does Reinforcement learning works?	agent interacting with an environment and lear...	Machine Learning
7	What are the realworld applications of ML?	email filtering, speech recognition, recommend...	Machine Learning
8	What is an ML ethical issue?	Bias in data	Machine Learning
9	what does the network security do?	protecting internal networks from intrusions	Cybersecurity

```
test_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44 entries, 0 to 43
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Question  44 non-null     object
 1   Answer    44 non-null     object
 2   Document  44 non-null     object
dtypes: object(3)
memory usage: 1.2+ KB
```

```
predictions_df_flan_t5 = generate_predictions_for_dataset(test_dataset, generate_answer_f

# Convert the first 3 rows of the DataFrame to JSON format
json_output = predictions_df_flan_t5.head(3).to_json(orient='records', lines=False)
parsed_json = json.loads(json_output)
print(json.dumps(parsed_json, indent=4))
```

Show hidden output

```
predictions_df_flan_t5_squad2 = generate_predictions_for_dataset(test_dataset, generate_a
```

```
# Convert the first 3 rows of the DataFrame to JSON format
json_output = predictions_df_flan_t5_squad2.head(3).to_json(orient='records', lines=False
parsed_json = json.loads(json_output)
print(json.dumps(parsed_json, indent=4))
```


```
[
    {
        "Question": "what are the common evaluation metrics in supervised Learning",
        "Answer": "Accuracy, Precison, recall, F1 score and roc-auc",
        "Document": "Machine Learning",
        "Prediction": "accuracy, precision, recall, f1-score, and roc-auc"
    },
    {
        "Question": "What the the different types in ML?",
        "Answer": "Supervised Learning, Unsupervised Learning and Reinforcement Learn
        "Document": "Machine Learning",
        "Prediction": "supervised learning, unsupervised learning, and reinforcement
    },
    {
        "Question": "What are foundational techniques used in predictive modeling?",
        "Answer": "Linear regression and logistic regression",
        "Document": "Machine Learning",
        "Prediction": "linear regression and logistic regression"
    }
]
```


```
predictions_df_bart = generate_predictions_for_dataset(test_dataset, generate_answer_with

# Convert the first 3 rows of the DataFrame to JSON format
json_output = predictions_df_bart.head(3).to_json(orient='records', lines=False)
parsed_json = json.loads(json_output)
print(json.dumps(parsed_json, indent=4))
```

Show hidden output

```
predictions_df_bart_squad2 = generate_predictions_for_dataset(test_dataset, generate_answ

# Convert the first 3 rows of the DataFrame to JSON format
json_output = predictions_df_bart_squad2.head(3).to_json(orient='records', lines=False)
parsed_json = json.loads(json_output)
print(json.dumps(parsed_json, indent=4))
```

Show hidden output

Evaluation

```
# Function to evaluate predictions for each row
def evaluate_predictions(df: pd.DataFrame) -> pd.DataFrame:
    exact_match_scores = []
    f1_scores = []

    for idx, row in df.iterrows():
```

```
            prediction = row["Prediction"]
            ground_truth = row["Answer"]  # Ground truth is in the 'Answer' colum

            # Calculate exact match score
            em_score = exact_match_score(prediction, ground_truth)
            exact_match_scores.append(em_score)

            # Calculate F1 score
            f1 = f1_score(prediction, ground_truth)
            f1_scores.append(f1)

        # Add the evaluation scores to the dataframe
        df["Exact Match Score"] = exact_match_scores
        df["F1 Score"] = f1_scores

        return df


    # Function to get average EM and F1
    def get_avg_scores(df: pd.DataFrame) -> dict:
        return {
            "Exact Match": df["Exact Match Score"].mean(),
            "F1 Score": df["F1 Score"].mean()
        }


    # Evaluate and compute scores for each model's prediction DataFrame
    predictions_with_scores_flan_t5 = evaluate_predictions(predictions_df_flan_t5)
    avg_scores_flan_t5 = get_avg_scores(predictions_with_scores_flan_t5)

    predictions_with_scores_flan_t5_squad2 = evaluate_predictions(predictions_df_flar
    avg_scores_flan_t5_squad2 = get_avg_scores(predictions_with_scores_flan_t5_squad2

    predictions_with_scores_bart = evaluate_predictions(predictions_df_bart)
    avg_scores_bart = get_avg_scores(predictions_with_scores_bart)

    predictions_with_scores_bart_squad2 = evaluate_predictions(predictions_df_bart_sc
    avg_scores_bart_squad2 = get_avg_scores(predictions_with_scores_bart_squad2)


    # Print the results clearly
    print("Average Scores:")
    print("FLAN-T5:", avg_scores_flan_t5)
    print("FLAN-T5 (SQuAD2):", avg_scores_flan_t5_squad2)
    print("BART:", avg_scores_bart)
    print("BART (SQuAD2):", avg_scores_bart_squad2)
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.