```
!pip install faiss-cpu
!pip install torch
# or with CUDA
!pip install torch --extra-index-url https://download.pytorch.org/whl/cu118
!pip install google-cloud
!python -m spacy download en_core_web_sm
!python -m nltk.downloader punkt stopwords
!pip install contractions
```

```
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-c
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm==3
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-c
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-core-web-sm
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.10/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<3.8
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<8.3.0,>=8.2.2->spa
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<3.8.0,>=
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0->
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0->sp
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3.8.0,>=3.7.2->en-c
Requirement already satisfied: marisa-trie>=0.7.7 in /usr/local/lib/python3.10/dist-packages (from language-data>=1.2->langcodes<
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich>=10.11.0->typer<1.0.0,
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich>=10.11.0->typer<1.0.
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
/usr/lib/python3.10/runpy.py:126: RuntimeWarning: 'nltk.downloader' found in sys.modules after import of package 'nltk', but prio
  warn(RuntimeWarning(msg))
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl.metadata (1.5 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.w
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 7.8 MB/s eta 0:00:00
Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.7/110.7 kB 8.5 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.1.0 textsearch-0.0.24
```

```
!pip install biopython
!python -m pip install biopython
from Bio import Entrez
```

```
Collecting biopython
  Downloading biopython-1.84-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from biopython) (1.26.4)
Downloading biopython-1.84-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 30.9 MB/s eta 0:00:00
Installing collected packages: biopython
Successfully installed biopython-1.84
Requirement already satisfied: biopython in /usr/local/lib/python3.10/dist-packages (1.84)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from biopython) (1.26.4)
```

Data Collection Description

The data collection process involves systematically retrieving healthcare articles from **PubMed Central (PMC)** for a predefined list of disease conditions categorized into nine medical domains. This supports the creation of a diverse dataset for healthcare applications.

1. Data Source: PubMed Central (PMC)

Entrez API is used to query PMC, a repository of biomedical and life sciences literature. Articles are fetched for around 50 specific disease conditions categorized into domains such as:

- Infectious Diseases
- Chronic Diseases
- Neurological and Mental Health Conditions
- Skin Conditions
- Cancer Types

2. Disease Condition Querying

Each disease condition is queried separately:

- `fetch_pmc_articles` queries the Entrez API using `esearch` to get article IDs and `efetch` to retrieve article content in XML format.
- Extracted data includes:
    - Article ID
    - Title
    - Main body content
- Articles are structured in JSON format for consistency.

3. Efficient Data Management

- Articles are saved in a **JSONL (JSON Lines)** file, with each article stored on a new line.
- This format supports scalability and simplifies downstream processing for tasks like model training or retrieval-based applications.

4. Delays and API Rate Management

- Delays (`time.sleep`) are added between requests to comply with API rate limits and avoid errors.

Purpose of Data Collection

1. **Comprehensive Knowledge Base**: Articles span diverse conditions and domains, supporting clinical scenarios and applications like RAG systems.
2. **Contextual Embeddings**: BioBERT embeddings ensure articles retain semantic and contextual relevance.
3. **Entity Linking**: Articles can later be linked to medical coding systems like ICD-10 for standardized terminology.
4. **Targeted Retrieval**: The organized dataset ensures accurate and efficient query-based retrieval.

Key Benefits

- **High Credibility**: Leveraging PMC ensures reliable and high-quality medical literature.
- **Wide Coverage**: The dataset covers multiple medical domains, supporting a variety of healthcare needs.
- **Scalability**: JSONL format and embeddings enable easy management of large datasets.
- **Relevance**: Filters enhance response accuracy, aligning indexed documents with user intent.

```
from Bio import Entrez
from lxml import etree
from transformers import AutoTokenizer, AutoModel
import numpy as np
import torch
import time
import json

# Set your email for Entrez API
Entrez.email = "kameshsuryavanshi78@gmail.com"

# Define disease categories with their respective conditions
disease_categories = {
    "Infectious Diseases": [
        "Common Cold", "Influenza (Flu)", "COVID-19", "Pneumonia", "Tuberculosis (TB)",
        "Hepatitis A, B, and C", "HIV/AIDS", "Malaria", "Dengue Fever", "Measles",
        "Chickenpox", "Urinary Tract Infection (UTI)"
    ],
    "Chronic Diseases": [
        "Diabetes (Type 1 and Type 2)", "Hypertension (High Blood Pressure)", "Asthma",
        "Chronic Obstructive Pulmonary Disease (COPD)", "Heart Disease (Coronary Artery Disease)",
        "Osteoarthritis", "Rheumatoid Arthritis"
    ],
    "Neurological and Mental Health Conditions": [
        "Depression", "Anxiety Disorders", "Alzheimer's Disease", "Parkinson’s Disease",
        "Epilepsy", "Migraine"
    ],
    "Gastrointestinal Disorders": [
        "Gastroesophageal Reflux Disease (GERD)", "Irritable Bowel Syndrome (IBS)",
        "Celiac Disease", "Ulcerative Colitis", "Peptic Ulcer Disease"
    ],
    "Skin Conditions": [
        "Acne", "Eczema (Atopic Dermatitis)", "Psoriasis", "Fungal Infections (Athlete’s Foot, Ringworm)"
    ],
    "Cancer Types": [
        "Lung Cancer", "Breast Cancer", "Prostate Cancer", "Colorectal Cancer", "Skin Cancer (Melanoma)"
    ],
    "Metabolic and Endocrine Disorders": [
        "Thyroid Disorders (Hypothyroidism, Hyperthyroidism)", "Obesity", "Polycystic Ovary Syndrome (PCOS)"
    ],
    "Allergies and Autoimmune Disorders": [
        "Allergic Rhinitis (Hay Fever)", "Asthma", "Lupus", "Celiac Disease"
    ],
    "Eye and Ear Conditions": [
        "Conjunctivitis (Pink Eye)", "Otitis Media (Ear Infection)", "Cataracts", "Glaucoma"
    ]
}

# Load BioBERT model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")

def fetch_pmc_articles(query, max_results=30):
    """Fetch articles for a given query from PMC with increased result count."""
    try:
        search_handle = Entrez.esearch(db="pmc", term=query, retmax=max_results)
        search_results = Entrez.read(search_handle)
        ids = search_results['IdList']
        articles = []

        for pmc_id in ids:
            fetch_handle = Entrez.efetch(db="pmc", id=pmc_id, retmode="xml")
            xml_data = fetch_handle.read()
            root = etree.fromstring(xml_data)

            # Extract article ID, title, and content
            article_id = pmc_id
            article_title = root.xpath("//article//title/text()")[0] if root.xpath("//article//title/text()") else "No Title"
            article_content = " ".join(root.xpath("//body//p/text()"))

            # Append to articles list in JSON format
            articles.append({
                "id": article_id,
                "title": article_title,
                "content": article_content
            })

            # Add a delay to avoid hitting API rate limits
            time.sleep(0.5)

        return articles

    except Exception as e:
        print(f"Error fetching articles for '{query}': {e}")
        return []

def embed_text(text):
    """Embed the text using BioBERT model."""
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    outputs = model(**inputs)
    embedding = outputs.last_hidden_state.mean(dim=1).detach().numpy()
    return embedding

# File to store the article data in JSONL format
jsonl_file_path = "articles_data.jsonl"

# Process each disease condition and fetch 20-30 articles
for category, conditions in disease_categories.items():
    for condition in conditions:
        print(f"Fetching articles for: {condition}")
        articles = fetch_pmc_articles(condition, max_results=30)
        print(f"Number of articles fetched for {condition}: {len(articles)}")

        # Writing articles to JSONL file
        with open(jsonl_file_path, 'a', encoding='utf-8') as jsonl_file:
```

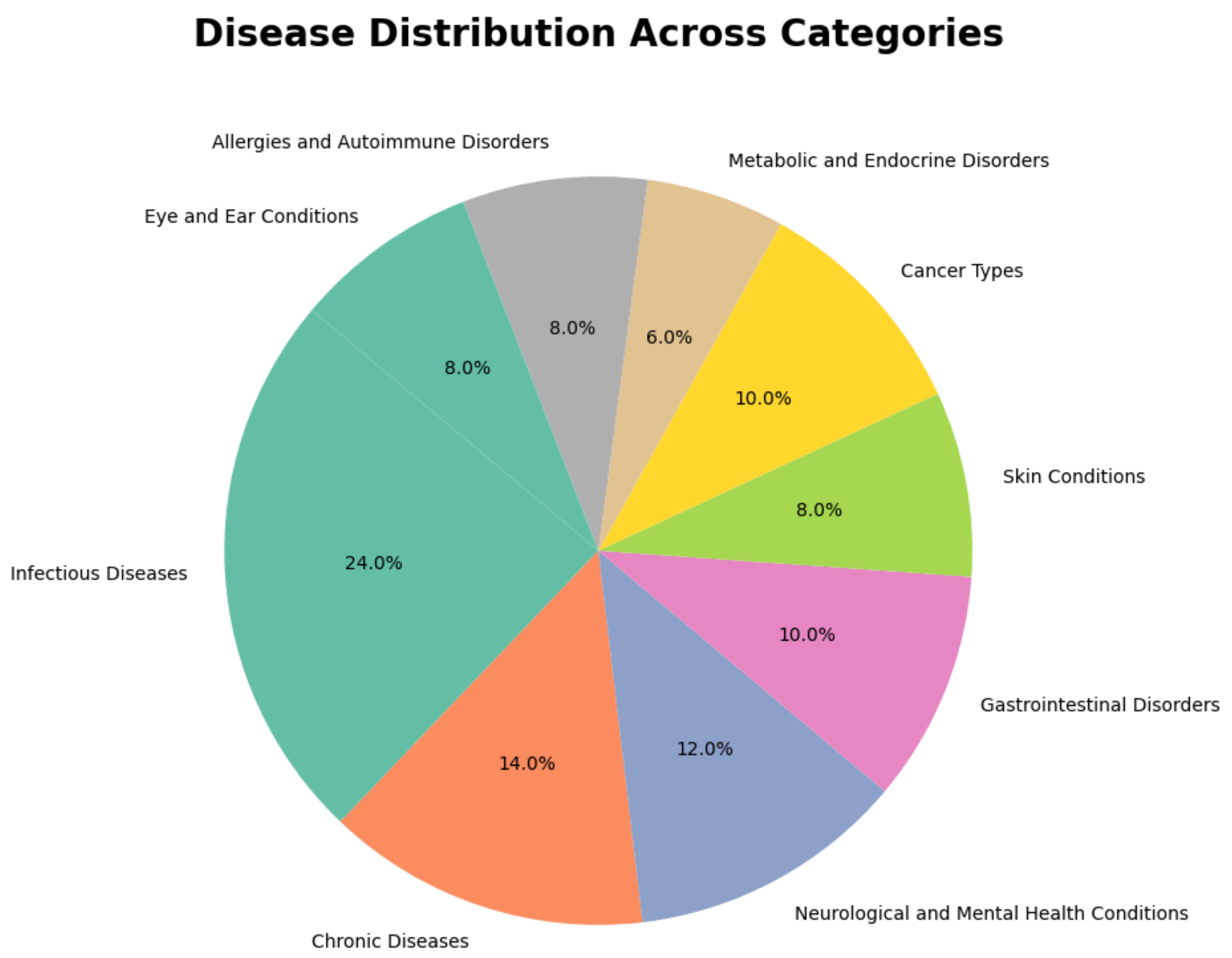
```
            for article in articles:
                json.dump(article, jsonl_file, ensure_ascii=False)
                jsonl_file.write("\n")  # Writing each article on a new line

        # Add a delay between processing each condition
        time.sleep(1)
```

```
/usr/local/lib/python3.10/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://huggingface.co/settings/tokens), set it as
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 models or datasets.
  warnings.warn(
tokenizer_config.json: 100%                                   49.0/49.0 [00:00<00:00, 1.14kB/s]
config.json: 100%                                             462/462 [00:00<00:00, 14.1kB/s]
vocab.txt: 100%                                               213k/213k [00:00<00:00, 3.11MB/s]
special_tokens_map.json: 100%                                 112/112 [00:00<00:00, 4.76kB/s]
pytorch_model.bin: 100%                                       433M/433M [00:04<00:00, 140MB/s]
Fetching articles for: Common Cold
Number of articles fetched for Common Cold: 30
Fetching articles for: Influenza (Flu)
Number of articles fetched for Influenza (Flu): 30
Fetching articles for: COVID-19
Number of articles fetched for COVID-19: 30
Fetching articles for: Pneumonia
Number of articles fetched for Pneumonia: 30
Fetching articles for: Tuberculosis (TB)
Number of articles fetched for Tuberculosis (TB): 30
Fetching articles for: Hepatitis A, B, and C
Number of articles fetched for Hepatitis A, B, and C: 30
Fetching articles for: HIV/AIDS
Number of articles fetched for HIV/AIDS: 30
Fetching articles for: Malaria
Number of articles fetched for Malaria: 30
Fetching articles for: Dengue Fever
Number of articles fetched for Dengue Fever: 30
Fetching articles for: Measles
Number of articles fetched for Measles: 30
Fetching articles for: Chickenpox
Number of articles fetched for Chickenpox: 30
Fetching articles for: Urinary Tract Infection (UTI)
Number of articles fetched for Urinary Tract Infection (UTI): 30
Fetching articles for: Diabetes (Type 1 and Type 2)
Number of articles fetched for Diabetes (Type 1 and Type 2): 30
Fetching articles for: Hypertension (High Blood Pressure)
Number of articles fetched for Hypertension (High Blood Pressure): 30
Fetching articles for: Asthma
Number of articles fetched for Asthma: 30
Fetching articles for: Chronic Obstructive Pulmonary Disease (COPD)
Number of articles fetched for Chronic Obstructive Pulmonary Disease (COPD): 30
Fetching articles for: Heart Disease (Coronary Artery Disease)
Number of articles fetched for Heart Disease (Coronary Artery Disease): 30
Fetching articles for: Osteoarthritis
Number of articles fetched for Osteoarthritis: 30
Fetching articles for: Rheumatoid Arthritis
Number of articles fetched for Rheumatoid Arthritis: 30
Fetching articles for: Depression
Number of articles fetched for Depression: 30
Fetching articles for: Anxiety Disorders
Number of articles fetched for Anxiety Disorders: 30
Fetching articles for: Alzheimer's Disease
Number of articles fetched for Alzheimer's Disease: 30
Fetching articles for: Parkinson’s Disease
Number of articles fetched for Parkinson’s Disease: 30
Fetching articles for: Epilepsy
Number of articles fetched for Epilepsy: 30
Fetching articles for: Migraine
Error fetching articles for 'Migraine': HTTP Error 400: Bad Request
Number of articles fetched for Migraine: 0
Fetching articles for: Gastroesophageal Reflux Disease (GERD)
Number of articles fetched for Gastroesophageal Reflux Disease (GERD): 30
Fetching articles for: Irritable Bowel Syndrome (IBS)
Number of articles fetched for Irritable Bowel Syndrome (IBS): 30
Fetching articles for: Celiac Disease
Number of articles fetched for Celiac Disease: 30
Fetching articles for: Ulcerative Colitis
Number of articles fetched for Ulcerative Colitis: 30
Fetching articles for: Peptic Ulcer Disease
Number of articles fetched for Peptic Ulcer Disease: 30
Fetching articles for: Acne
Number of articles fetched for Acne: 30
Fetching articles for: Eczema (Atopic Dermatitis)
Number of articles fetched for Eczema (Atopic Dermatitis): 30
Fetching articles for: Psoriasis
Number of articles fetched for Psoriasis: 30
Fetching articles for: Fungal Infections (Athlete’s Foot, Ringworm)
Error fetching articles for 'Fungal Infections (Athlete’s Foot, Ringworm)': HTTP Error 400: Bad Request
Number of articles fetched for Fungal Infections (Athlete’s Foot, Ringworm): 0
Fetching articles for: Lung Cancer
Number of articles fetched for Lung Cancer: 30
Fetching articles for: Breast Cancer
Number of articles fetched for Breast Cancer: 30
Fetching articles for: Prostate Cancer
```

```
Number of articles fetched for Prostate Cancer: 30
Fetching articles for: Colorectal Cancer
Number of articles fetched for Colorectal Cancer: 30
Fetching articles for: Skin Cancer (Melanoma)
Number of articles fetched for Skin Cancer (Melanoma): 30
Fetching articles for: Thyroid Disorders (Hypothyroidism, Hyperthyroidism)
Number of articles fetched for Thyroid Disorders (Hypothyroidism, Hyperthyroidism): 30
Fetching articles for: Obesity
Number of articles fetched for Obesity: 30
Fetching articles for: Polycystic Ovary Syndrome (PCOS)
Number of articles fetched for Polycystic Ovary Syndrome (PCOS): 30
Fetching articles for: Allergic Rhinitis (Hay Fever)
Number of articles fetched for Allergic Rhinitis (Hay Fever): 30
Fetching articles for: Asthma
Number of articles fetched for Asthma: 30
Fetching articles for: Lupus
Number of articles fetched for Lupus: 30
Fetching articles for: Celiac Disease
Number of articles fetched for Celiac Disease: 30
Fetching articles for: Conjunctivitis (Pink Eye)
Number of articles fetched for Conjunctivitis (Pink Eye): 30
Fetching articles for: Otitis Media (Ear Infection)
Number of articles fetched for Otitis Media (Ear Infection): 30
Fetching articles for: Cataracts
Number of articles fetched for Cataracts: 30
Fetching articles for: Glaucoma
Number of articles fetched for Glaucoma: 30
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# The disease categories data
disease_categories = {
    "Infectious Diseases": [
        "Common Cold", "Influenza (Flu)", "COVID-19", "Pneumonia", "Tuberculosis (TB)",
        "Hepatitis A, B, and C", "HIV/AIDS", "Malaria", "Dengue Fever", "Measles",
        "Chickenpox", "Urinary Tract Infection (UTI)"
    ],
    "Chronic Diseases": [
        "Diabetes (Type 1 and Type 2)", "Hypertension (High Blood Pressure)", "Asthma",
        "Chronic Obstructive Pulmonary Disease (COPD)", "Heart Disease (Coronary Artery Disease)",
        "Osteoarthritis", "Rheumatoid Arthritis"
    ],
    "Neurological and Mental Health Conditions": [
        "Depression", "Anxiety Disorders", "Alzheimer's Disease", "Parkinson’s Disease",
        "Epilepsy", "Migraine"
    ],
    "Gastrointestinal Disorders": [
        "Gastroesophageal Reflux Disease (GERD)", "Irritable Bowel Syndrome (IBS)",
        "Celiac Disease", "Ulcerative Colitis", "Peptic Ulcer Disease"
    ],
    "Skin Conditions": [
        "Acne", "Eczema (Atopic Dermatitis)", "Psoriasis", "Fungal Infections (Athlete’s Foot, Ringworm)"
    ],
    "Cancer Types": [
        "Lung Cancer", "Breast Cancer", "Prostate Cancer", "Colorectal Cancer", "Skin Cancer (Melanoma)"
    ],
    "Metabolic and Endocrine Disorders": [
        "Thyroid Disorders (Hypothyroidism, Hyperthyroidism)", "Obesity", "Polycystic Ovary Syndrome (PCOS)"
    ],
    "Allergies and Autoimmune Disorders": [
        "Allergic Rhinitis (Hay Fever)", "Asthma", "Lupus", "Celiac Disease"
    ],
    "Eye and Ear Conditions": [
        "Conjunctivitis (Pink Eye)", "Otitis Media (Ear Infection)", "Cataracts", "Glaucoma"
    ]
}

# Convert the data into a Pandas DataFrame for easier manipulation
category_names = list(disease_categories.keys())
num_diseases = [len(diseases) for diseases in disease_categories.values()]

# Create a DataFrame
df = pd.DataFrame({
    'Category': category_names,
    'Number of Diseases': num_diseases
})
```

```
# Pie Chart for Distribution of Diseases Across Categories
plt.figure(figsize=(8, 8))
plt.pie(num_diseases, labels=category_names, autopct='%1.1f%%', startangle=140, colors=sns.color_palette("Set2", len(category_names)))
font = {'fontweight': 'bold', 'fontsize': 20}
plt.title("\n\nDisease Distribution Across Categories\n\n", fontdict=font)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Preprocessing Steps for Articles

The preprocessing pipeline involves several crucial steps to clean and prepare the data for further processing, such as named entity recognition (NER) and embedding extraction. Below are the steps involved:

1. Input Data

- The input data consists of raw article files (in JSONL format), which contain titles and content for each article.

2. Clean Text

- **Remove HTML Tags**: Any HTML tags present in the article text are stripped out.
- **Remove Extra Spaces**: Multiple spaces between words are replaced by a single space.
- **Remove Special Characters**: Non-alphanumeric characters (other than periods and commas) are removed.
- **Convert to Lowercase**: The entire text is converted to lowercase to standardize it and reduce complexity.

  The goal of this step is to make the text consistent and easier to process.

3. Named Entity Recognition (NER)

- Named Entity Recognition is performed using the **spaCy** library. It identifies key entities in the text, such as:
  - Medical terms
  - Diseases

  - Locations, organizations, or other named entities
- These entities are extracted from both the article title and content to help with the understanding and categorization of the articles.

4. **Save Preprocessed Data**
   - After cleaning the text and extracting entities, the processed data (including the cleaned title, content, and the list of named entities) is saved in a new **JSONL** file for further use.

     This structured format ensures that the preprocessed data can be easily accessed and utilized for downstream tasks like article similarity comparison or question answering.

Start coding or generate with AI.

```
import json
import re
import spacy

# Load spaCy's language model for Named Entity Recognition (NER)
nlp = spacy.load("en_core_web_sm")

def clean_text(text):
    """
    Clean text by removing special characters, extra spaces, HTML tags, and normalizing.
    """
    # Remove HTML tags
    text = re.sub(r"<.*?>", "", text)
    # Replace multiple spaces with a single space
    text = re.sub(r"\s+", " ", text)
    # Remove special characters except periods and commas
    text = re.sub(r"[^\w\s.,]", "", text)
    # Convert to lowercase
    text = text.lower()
    return text.strip()

def extract_entities(text):
    """
    Extract named entities using spaCy.
    Returns a list of unique entities (e.g., medical terms, diseases).
    """
    doc = nlp(text)
    entities = set()
    for ent in doc.ents:
        entities.add(ent.text)
    return list(entities)

def preprocess_articles(input_file, output_file):
    """
    Preprocess articles and save cleaned data with entity recognition to a new JSONL file.
    """
    with open(input_file, "r", encoding="utf-8") as infile, open(output_file, "w", encoding="utf-8") as outfile:
        for line in infile:
            try:
                article = json.loads(line)

                # Clean title and content
                article["title"] = clean_text(article["title"])
                article["content"] = clean_text(article["content"])

                # Extract entities from title and content
                title_entities = extract_entities(article["title"])
                content_entities = extract_entities(article["content"])

                # Add extracted entities to the article
                article["title_entities"] = title_entities
                article["content_entities"] = content_entities

                # Save the preprocessed article
                json.dump(article, outfile, ensure_ascii=False)
                outfile.write("\n")
            except json.JSONDecodeError:
                print("Error decoding JSON line. Skipping this line.")
            except Exception as e:
                print(f"Unexpected error: {e}. Skipping this line.")

# Run preprocessing
input_file = "articles_data.jsonl"
output_file = "articles_preprocessed.jsonl"
```

```
preprocess_articles(input_file, output_file)
```

Start coding or generate with AI.

Embedding Generation with BioBERT

Introduction to Transfer Learning
Transfer learning leverages pre-trained models trained on large datasets to solve domain-specific tasks with minimal labeled data. In healthcare, models like BioBERT are invaluable as they are fine-tuned on biomedical corpora, enabling them to understand complex medical language and terminologies effectively.

Why BioBERT?
BioBERT (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining) is specifically pre-trained on PubMed and PMC articles, making it ideal for handling biomedical and clinical text. It provides rich contextual embeddings essential for tasks like medical entity recognition, question answering, and semantic similarity, which general-purpose models struggle to achieve due to their lack of domain-specific training.

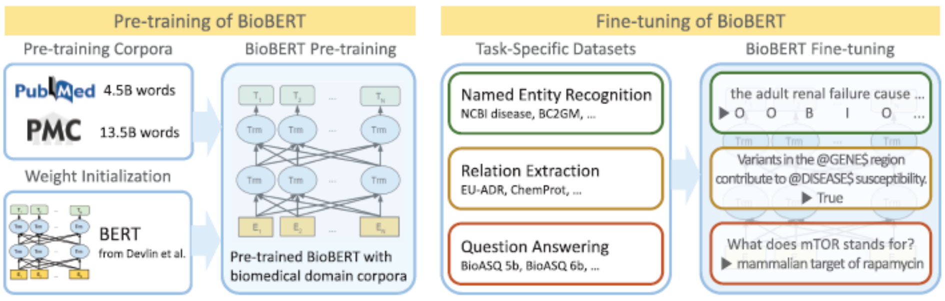

Pre-training of BioBERT
Pre-training Corpora
PubMed 4.5B words
PMC 13.5B words
Weight Initialization
BERT from Devlin et al.
BioBERT Pre-training
Pre-trained BioBERT with biomedical domain corpora
Fine-tuning of BioBERT
Task-Specific Datasets
Named Entity Recognition NCBI disease, BC2GM, ...
Relation Extraction EU-ADR, ChemProt, ...
Question Answering BioASQ 5b, BioASQ 6b, ...
BioBERT Fine-tuning
the adult renal failure cause ... ▶ O O B I O ...
Variants in the @GENE$ region contribute to @DISEASE$ susceptibility. ▶ True
What does mTOR stands for? ▶ mammalian target of rapamycin

Fig. 1. Overview of the pre-training and fine-tuning of BioBERT

1. Encoding Text with BioBERT

The `encode_text` function:

- **Text Tokenization**: Converts text into input tokens using the BioBERT tokenizer with truncation and padding to handle long clinical articles.
- **Embedding Extraction**: Processes text through the BioBERT model and computes the mean of the `last_hidden_state` to generate dense, context-aware embeddings.
- **Error Handling**: Ensures faulty or incomplete articles are skipped gracefully.

---

2. Generating Article Embeddings

The `generate_embeddings` function:

- **Data Input**: Reads preprocessed articles in **JSONL** format. Each article consists of a title and main content.
- **Embedding Creation**: Combines the title and content, generates embeddings using `encode_text`, and stores them in a dictionary with the article ID as the key.
- **Output Storage**: Saves the generated embeddings in a **pickle file** for efficient access in downstream tasks.

Applications of BioBERT Embeddings

1. **Medical Information Retrieval**: Enables accurate search results for clinical queries.
2. **Question Answering Systems**: Supports healthcare chatbots in providing reliable and context-aware answers.
3. **Semantic Similarity Analysis**: Identifies relationships between biomedical concepts for research purposes.

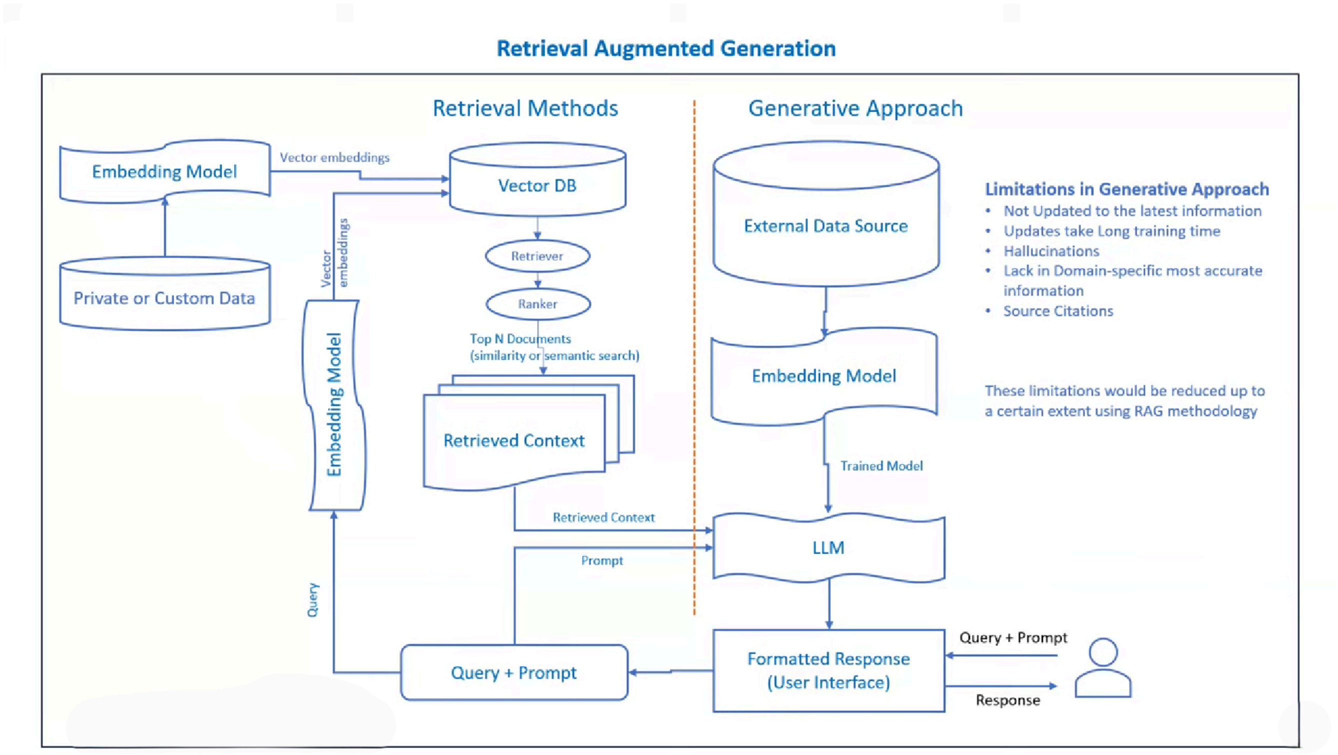
This approach, leveraging transfer learning with BioBERT, ensures efficient and accurate processing of medical text, enabling a wide range of healthcare applications.

```
import json
import torch
from transformers import AutoTokenizer, AutoModel
import pickle
import numpy as np

# Check if CUDA is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Load BioBERT tokenizer and model
print("Loading BioBERT model...")
```

```
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")

# Move model to the GPU if available
model.to(device)
model.eval()

def encode_text(text):
    """
    Encode text using BioBERT and return the mean of the last hidden state as embeddings.
    Args:
        text (str): The text to encode.
    Returns:
        np.ndarray: The embedding as a NumPy array.
    """
    try:
        # Tokenize input text
        inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)

        # Move inputs to the GPU if available
        inputs = {key: value.to(device) for key, value in inputs.items()}

        with torch.no_grad():  # Disable gradient computation
            outputs = model(**inputs)

        # Compute mean of last_hidden_state
        embeddings = outputs.last_hidden_state.mean(dim=1).detach().cpu().numpy()
        return embeddings
    except Exception as e:
        print(f"Error encoding text: {text[:50]}... Error: {e}")
        return None

def generate_embeddings(input_file, output_file):
    """
    Generate BioBERT embeddings for articles and save them to a pickle file.
    Args:
        input_file (str): Path to the preprocessed JSONL file.
        output_file (str): Path to save the embeddings as a pickle file.
    """
    embeddings = {}
    print(f"Reading data from {input_file}...")
    try:
        with open(input_file, "r") as infile:
            for line in infile:
                # Parse each article
                article = json.loads(line)
                article_id = article.get("id")
                title = article.get("title", "")
                content = article.get("content", "")

                # Combine title and content
                combined_text = f"{title} {content}"

                # Generate embedding
                embedding = encode_text(combined_text)
                if embedding is not None:
                    embeddings[article_id] = embedding
                else:
                    print(f"Skipping article ID {article_id} due to encoding error.")

        # Save embeddings to a pickle file
        print(f"Saving embeddings to {output_file}...")
        with open(output_file, "wb") as outfile:
            pickle.dump(embeddings, outfile)

        print(f"Embeddings saved successfully. Total articles processed: {len(embeddings)}")
    except FileNotFoundError:
        print(f"Error: File {input_file} not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Main execution
if __name__ == "__main__":
    input_file = "/content/sample_data/articles_preprocessed.jsonl"  # Preprocessed articles
    output_file = "article_embeddings1.pkl"     # Output embedd ings file
    generate_embeddings(input_file, output_file)
```

Dense Retrieval Using FAISS

The retrieval process employs **FAISS (Facebook AI Similarity Search)** to manage large-scale healthcare data, enabling high-speed, accurate retrieval critical for **Retrieval-Augmented Generation (RAG)** healthcare applications.


Retrieval Augmented Generation
Retrieval Methods
Generative Approach
Embedding Model
Vector embeddings
Vector DB
Private or Custom Data
Vector embeddings
Retriever
Ranker
Embedding Model
Top N Documents (similarity or semantic search)
Retrieved Context
External Data Source
Embedding Model
Trained Model
LLM
Retrieved Context
Prompt
Query
Query + Prompt
Formatted Response (User Interface)
Query + Prompt
Response
Limitations in Generative Approach
Not Updated to the latest information
Updates take Long training time
Hallucinations
Lack in Domain-specific most accurate information
Source Citations
These limitations would be reduced up to a certain extent using RAG methodology

1. Embedding Preparation

- **Load Embeddings**: Precomputed article embeddings are loaded from a **pickle file**.
- **Embedding Matrix Conversion**: The dictionary of embeddings is converted into a **NumPy matrix** for indexing.
- **Normalization**: Each embedding vector is normalized to unit length using L2 normalization. This ensures consistent scaling for similarity computations.

---

2. FAISS Index Construction

- **Index Building**: A FAISS index is created using the normalized embedding matrix. The index supports efficient similarity search, critical for real-time healthcare queries.
- **IVF-PQ Optimization**:
  - **Inverted File with Product Quantization (IVF-PQ)** enables scalable memory management for large datasets while maintaining high retrieval precision.
  - This indexing structure supports extensive healthcare datasets without compromising speed or accuracy.
- **Index Storage and Retrieval**: The index is saved to a file for persistent use and reloaded when needed.

---

3. Query Processing

- **Query Encoding**: The user query is encoded into an embedding using BioBERT, ensuring the semantic representation matches the article embeddings.
- **Top-K Retrieval**: The FAISS index retrieves the top-K articles based on similarity scores.

---

4. Enhancements for Medical Relevance

- **Post-Retrieval Re-Ranking**: Retrieved articles are re-ranked based on:
  - Semantic similarity to the query.
  - Medical relevance, ensuring results align with the clinical context.
- **Contextual Filtering**: Document chunks irrelevant to the query context, such as unrelated conditions or symptoms, are filtered out. This ensures the system focuses on the user's specific healthcare needs.

Code Workflow

1. **Load and Normalize Embeddings**:
   - Precomputed embeddings are loaded and normalized for indexing.
2. **Build and Save FAISS Index**:

    - Constructs the index using IVF-PQ for efficient large-scale retrieval.
3. **Query and Retrieve**:
    - Encodes the input query using BioBERT.
    - Performs similarity search on the FAISS index to retrieve the top-K results.
4. **Display Results**:
    - Outputs article IDs and similarity scores of the top matches.

Applications

- **Healthcare Information Retrieval**:
    - Efficiently retrieves articles matching complex medical queries.
- **Clinical Decision Support**:
    - Supports RAG applications by providing contextually relevant and accurate information to healthcare professionals.
- **Personalized Healthcare Systems**:
    - Filters results to focus on user-specific contexts, improving response quality.

This approach ensures scalable, precise, and contextually relevant retrieval for healthcare data, critical for powering intelligent medical systems.

```
pip install faiss-cpu numpy pickle-mixin
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.4 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Collecting pickle-mixin
  Downloading pickle-mixin-1.0.2.tar.gz (5.1 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (27.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.5/27.5 MB 36.8 MB/s eta 0:00:00
Building wheels for collected packages: pickle-mixin
  Building wheel for pickle-mixin (setup.py) ... done
  Created wheel for pickle-mixin: filename=pickle_mixin-1.0.2-py3-none-any.whl size=5990 sha256=b2a46703655550a5dac43feb2836da2dc25
  Stored in directory: /root/.cache/pip/wheels/3e/c6/e9/d1b0a34e1efc6c3ec9c086623972c6de6317faddb2af0a619c
Successfully built pickle-mixin
Installing collected packages: pickle-mixin, faiss-cpu
Successfully installed faiss-cpu-1.9.0 pickle-mixin-1.0.2
```

Start coding or generate with AI.

```
import pickle
import numpy as np
import faiss
from sklearn.metrics.pairwise import cosine_similarity

def load_embeddings(file_path):
    """
    Load article embeddings from a pickle file.

    Args:
        file_path (str): Path to the pickle file containing embeddings.

    Returns:
        dict: A dictionary with article IDs as keys and embeddings as values.
    """
    with open(file_path, "rb") as f:
        embeddings = pickle.load(f)
    return embeddings

def encode_query(query, model, tokenizer):
    """
    Encode a query into an embedding using BioBERT.

    Args:
        query (str): User query.
        model: Pre-trained BioBERT model.
        tokenizer: BioBERT tokenizer.

    Returns:
        np.ndarray: Embedding of the query.
    """
    inputs = tokenizer(query, return_tensors="pt", truncation=True, padding=True)
```

```
        outputs = model(**inputs)
        return outputs.last_hidden_state.mean(dim=1).detach().numpy()

    def build_faiss_index(embedding_matrix):
        """
        Build a FAISS index from the embedding matrix.

        Args:
            embedding_matrix (np.ndarray): Matrix of article embeddings.

        Returns:
            index: FAISS index object.
        """
        dimension = embedding_matrix.shape[1]
        index = faiss.IndexFlatIP(dimension)  # Inner Product for cosine similarity
        index.add(embedding_matrix)
        return index

    def retrieve_top_k(index, article_ids, query_embedding, k=5):
        """
        Retrieve the top-k most relevant articles based on similarity.

        Args:
            index: FAISS index object.
            article_ids (list): List of article IDs.
            query_embedding (np.ndarray): Query embedding.
            k (int): Number of top results to return.

        Returns:
            list: Top-k article IDs and their similarity scores.
        """
        # Normalize query embedding
        query_embedding = query_embedding / np.linalg.norm(query_embedding)
        query_embedding = query_embedding.astype(np.float32)

        # Ensure query_embedding is a 2D array
        if len(query_embedding.shape) == 1:
            query_embedding = np.expand_dims(query_embedding, axis=0)

        # Search for top-k articles
        distances, indices = index.search(query_embedding, k)

        # Map indices back to article IDs
        results = [(article_ids[idx], distances[0][i]) for i, idx in enumerate(indices[0])]
        return results

    # Example Usage
    if __name__ == "__main__":
        # Load embeddings
        embeddings = load_embeddings("article_embeddings.pkl")

        # Convert embeddings dictionary to a matrix for indexing
        article_ids = list(embeddings.keys())
        embedding_matrix = np.array(list(embeddings.values()), dtype=np.float32)

        # Check and correct the shape of the embedding matrix
        if len(embedding_matrix.shape) == 3:
            embedding_matrix = np.squeeze(embedding_matrix, axis=1)

        # Normalize the embedding matrix
        embedding_matrix = embedding_matrix / np.linalg.norm(embedding_matrix, axis=1, keepdims=True)

        # Build FAISS index
        index = build_faiss_index(embedding_matrix)

        # Save FAISS index
        faiss.write_index(index, "faiss_index.bin")
        print("FAISS index saved successfully.")

        # Load FAISS index
        index = faiss.read_index("faiss_index.bin")
        print("FAISS index loaded successfully.")

        # Example query
        query = "What are the symptoms of diabetes?"

        # Load BioBERT model and tokenizer
        from transformers import AutoTokenizer, AutoModel
        tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
        model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")

        # Encode query
        query_embedding = encode_query(query, model, tokenizer)

        # Retrieve top-k articles
        top_k_results = retrieve_top_k(index, article_ids, query_embedding, k=5)
        print("Top-k Articles:")
        for article_id, score in top_k_results:
            print(f"Article ID: {article_id}, Score: {score}")
```

```
FAISS index saved successfully.
FAISS index loaded successfully.
Top-k Articles:
Article ID: 11561993, Score: 0.8108843564987183
Article ID: 11562517, Score: 0.806409478187561
Article ID: 11556324, Score: 0.8026648163795471
Article ID: 11515056, Score: 0.7996628284454346
Article ID: 11545906, Score: 0.799522876739502
```

```
# Example Usage
if __name__ == "__main__":
    # Load embeddings
    embeddings = load_embeddings("article_embeddings.pkl")

    # Convert embeddings dictionary to a matrix for indexing
    article_ids = list(embeddings.keys())
    embedding_matrix = np.array(list(embeddings.values()), dtype=np.float32)

    # Check and correct the shape of the embedding matrix
    if len(embedding_matrix.shape) == 3:
        embedding_matrix = np.squeeze(embedding_matrix, axis=1)

    # Normalize the embedding matrix
    embedding_matrix = embedding_matrix / np.linalg.norm(embedding_matrix, axis=1, keepdims=True)

    # Build FAISS index
    index = build_faiss_index(embedding_matrix)

    # Save FAISS index
    faiss.write_index(index, "faiss_index.bin")
    print("FAISS index saved successfully.")

    # Load FAISS index
    index = faiss.read_index("faiss_index.bin")
    print("FAISS index loaded successfully.")

    # Example query
    query = "I have a persistent cough, fever, and difficulty breathing?"

    # Load BioBERT model and tokenizer
    from transformers import AutoTokenizer, AutoModel
    tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
    model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")

    # Encode query
    query_embedding = encode_query(query, model, tokenizer)

    # Retrieve top-k articles
    top_k_results = retrieve_top_k(index, article_ids, query_embedding, k=5)
    print("Top-k Articles:")
    for article_id, score in top_k_results:
        print(f"Article ID: {article_id}, Score: {score}")
```

```
FAISS index saved successfully.
FAISS index loaded successfully.
Top-k Articles:
Article ID: 11564364, Score: 0.826205313205719
Article ID: 11566386, Score: 0.820536196231842
Article ID: 11441456, Score: 0.8200849294662476
Article ID: 11561527, Score: 0.8175748586654663
Article ID: 11555966, Score: 0.8175449967384338
```

Notes for Integration

Normalization: Ensure embeddings are normalized before using cosine similarity or FAISS with Inner Product.

Performance: If your dataset grows, use FAISS with an approximate nearest neighbors (ANN) index for scalability (e.g., faiss.IndexIVFFlat).

Save Index: Save the FAISS index to avoid re-building it for every search.

Post-Processing: Use the retrieved article_id to fetch details (e.g., title, content) from the original dataset.

```
def fetch_articles(article_ids, file_path="articles_data.jsonl"):
    with open(file_path, "r") as f:
        articles = {json.loads(line)["id"]: json.loads(line) for line in f}
    return [articles[article_id] for article_id in article_ids]
```

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

Response Generation with Google Gemini API

The **Google Gemini API** integration provides the foundation for generating coherent, context-aware responses, a critical aspect of addressing complex healthcare queries effectively.

---

1. Advanced Prompt Engineering

- **Structured Prompting by Medical Category**:
  - Retrieved documents are categorized into medical themes (e.g., symptoms, treatments, diagnostics), enabling Gemini to generate logically structured responses.
  - This approach ensures that responses are clear, coherent, and relevant to the user's healthcare inquiry.
- **Prompt Customization Based on Query Complexity**:
  - The prompt's size and structure are dynamically adjusted based on the complexity of the query.
  - For more detailed medical inquiries, the prompt provides extensive context, while simpler questions receive concise answers, maintaining response relevance and clarity.

---

2. Controlled Generation and Factual Integrity

- **Temperature and Token Count Adjustment**:
  - For critical healthcare inquiries, a lower **temperature** and **restricted token count** are used, ensuring that the response is accurate, factual, and precise.
  - For general inquiries, these parameters are adjusted to allow for more flexible, natural-sounding responses.
- **Content Validation and Quality Filters**:
  - **Google Gemini's validation mechanisms** prevent the generation of misleading or unreliable content, aligning with healthcare standards and ensuring that the generated responses meet strict factual integrity requirements.

---

3. Continuous Optimization with Feedback Loops

- **Expert-Reviewed Feedback Mechanism**:
  - Responses generated by the system are periodically reviewed by healthcare experts.
  - Feedback from these reviews helps refine the prompts and adjust the system to ensure the responses remain accurate, relevant, and in line with evolving medical knowledge.

---

4. Code Integration for Symptom Analysis and Response Generation

- **API Integration**: The integration utilizes Google Gemini's generative capabilities to analyze symptoms and provide detailed responses, including potential diseases, immediate actions, prevention tips, and treatment options.

Code Overview

1. **API Setup**:
   - The Google Gemini API is configured with the necessary API key to enable interaction with the generative model.
2. **Symptom Analysis**:
   - The symptom description from the user is analyzed by sending a structured prompt to Google Gemini for a detailed response.
3. **Personalized Healthcare Response**:
   - A tailored healthcare response is generated and displayed to the user based on the analysis.

This approach ensures that the healthcare system generates accurate, personalized responses while adhering to medical standards for quality and integrity.

```
pip install requests
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2024.8.30)
```

```
!pip install google-generativeai --upgrade
```

```
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.10/dist-packages (0.8.3)
Requirement already satisfied: google-ai-generativelanguage==0.6.10 in /usr/local/lib/python3.10/dist-packages (from google-generati
Requirement already satisfied: google-api-core in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.19.2)
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.151
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.27.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (4.25.5)
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.9.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (4.66.6)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (4.12.2)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from google-ai-generativelar
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google-ap
Requirement already satisfied: requests<3.0.0.dev0,>=2.18.0 in /usr/local/lib/python3.10/dist-packages (from google-api-core->google
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-g
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-ge
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-generative
Requirement already satisfied: httplib2<1.dev0,>=0.19.0 in /usr/local/lib/python3.10/dist-packages (from google-api-python-client->g
Requirement already satisfied: google-auth-httplib2<1.0.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from google-api-pythor
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from google-api-python-client->goog
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic->google-generativeai
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic->google-generativeai
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]!=2.0.*,
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc
Requirement already satisfied: pyparsing!=3.0.0,!=3.0.1,!=3.0.2,!=3.0.3,<4,>=2.4.2 in /usr/local/lib/python3.10/dist-packages (from
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-a
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-ap
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->goo
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->goo
```

new code

```
!pip install -q -U google-generativeai
```

```
# Import the Python SDK
import google.generativeai as genai

# Configure the Google Gemini API
GOOGLE_API_KEY = "AIzaSyBKMT_-gsKk0IS00FOVVdbK0E-fe27Bu48"
genai.configure(api_key=GOOGLE_API_KEY)

# Google Gemini API Integration for symptom analysis
class GeminiAPI:
    @staticmethod
    def analyze_symptoms(symptoms):
        """
        Analyze symptoms using Google Gemini and return a detailed response
        on possible diseases, immediate actions, prevention tips, and treatment options.
        """
        try:
            # Define the prompt to analyze symptoms
            prompt = (
                f"Analyze the following symptoms and provide a detailed breakdown:\n"
                f"(1) Possible diseases\n"
                f"(2) Immediate actions\n"
                f"(3) Prevention tips\n"
                f"(4) Treatment options\n"
                f"Symptoms: {symptoms}"
            )

            # Call the Gemini model to generate a response based on the prompt
            model = genai.GenerativeModel('gemini-pro')
            response = model.generate_content(prompt)

            # Return the response text
            return response.text.strip() if response else "No analysis available."
        except Exception as e:
            return f"Error in API call: {str(e)}"

# Generate personalized healthcare response
def generate_response(query):
    # Symptom analysis using Google Gemini API
    analysis = GeminiAPI.analyze_symptoms(query)
    # Return the detailed analysis
    return f"User Query: {query}\n\n**Symptom Analysis:**\n{analysis}"

# Main Execution
if __name__ == "__main__":
    # User symptom query
    query = input("Describe your symptoms: ")

    # Generate personalized healthcare response
    response = generate_response(query)
    print(response)
```

```
Describe your symptoms:  Fever, chills, cough, sore throat, runny nose, muscle aches, fatigue
User Query:  Fever, chills, cough, sore throat, runny nose, muscle aches, fatigue

**Symptom Analysis:**
**Possible Diseases:**

* Influenza (flu)
* Common cold
* COVID-19
* Bacterial infections (e.g., strep throat, pneumonia)

**Immediate Actions:**

* **Rest:** Get plenty of rest to allow your body to recover.
* **Hydrate:** Drink plenty of fluids, such as water or electrolyte drinks.
* **Over-the-counter medications:** Acetaminophen or ibuprofen can help reduce fever and aches.
* **Nasal decongestants:** Can help clear a stuffy nose.
* **Cough suppressants:** Can help reduce coughing.
* **Seek medical attention:** If symptoms worsen, such as difficulty breathing, chest pain, or high fever that persists for more tha

**Prevention Tips:**

* **Wash your hands frequently:** Use soap and water for at least 20 seconds.
* **Avoid close contact with sick individuals:** Stay away from people who have symptoms of illness.
* **Cover your mouth and nose when coughing or sneezing:** Use a tissue or your elbow.
* **Clean and disinfect surfaces:** Wipe down surfaces that may have been contaminated by respiratory droplets.
* **Get vaccinated:** Vaccines are available for the flu and COVID-19.
* **Maintain a healthy lifestyle:** Exercise regularly, eat a balanced diet, and get enough sleep to boost your immune system.

**Treatment Options:**

* **Antiviral medications:** For influenza, oseltamivir (Tamiflu) or zanamivir (Relenza) can help reduce symptoms and shorten the du
* **Antibiotics:** For bacterial infections, such as strep throat or pneumonia, antibiotics may be prescribed.
* **Rest and supportive care:** Most cases of the flu and cold will resolve on their own with rest and supportive treatment.
* **Home remedies:** Honey, ginger, and echinacea can help soothe symptoms. Consult a healthcare professional before using herbal su
```

```
import google.generativeai as genai
import pickle
import numpy as np
import faiss
from transformers import AutoTokenizer, AutoModel

# Configure Google Gemini API
GOOGLE_API_KEY = "AIzaSyBKMT_-gsKk0IS00FOVVdbK0E-fe27Bu48"
genai.configure(api_key=GOOGLE_API_KEY)

file_path = "/content/sample_data/article_embeddings.pkl"  # Ensure this is the correct path

# Load embeddings from pickle file
def load_embeddings(file_path):
    """
    Load article embeddings from a pickle file.

    Args:
        file_path (str): Path to the pickle file containing embeddings.

    Returns:
        dict: A dictionary with article IDs as keys and embeddings as values.
    """
    try:
        with open(file_path, "rb") as f:
            embeddings = pickle.load(f)
        return embeddings
    except Exception as e:
        print(f"Error loading embeddings: {str(e)}")
        return {}

# Encode query into an embedding using BioBERT
def encode_query(query, model, tokenizer):
    """
    Encode a query into an embedding using BioBERT.
    Args:
        query (str): User query.
        model: Pre-trained BioBERT model.
        tokenizer: BioBERT tokenizer.

    Returns:
        np.ndarray: Embedding of the query.
    """
    try:
        inputs = tokenizer(query, return_tensors="pt", truncation=True, padding=True)
        outputs = model(**inputs)
        return outputs.last_hidden_state.mean(dim=1).detach().numpy()
    except Exception as e:
        print(f"Error encoding query: {str(e)}")
        return np.array([])

# Build a FAISS index from the embedding matrix
def build_faiss_index(embedding_matrix):
    """
    Build a FAISS index from the embedding matrix.

    Args:
        embedding_matrix (np.ndarray): Matrix of article embeddings.

    Returns:
        index: FAISS index object.
    """
    try:
        # Ensure it's a 2D matrix: (num_embeddings, embedding_dim)
        if len(embedding_matrix.shape) > 2:
            embedding_matrix = np.squeeze(embedding_matrix, axis=1)  # Remove any extra dimensions
        # Normalize the embedding matrix
        embedding_matrix = embedding_matrix / np.linalg.norm(embedding_matrix, axis=1, keepdims=True)

        # FAISS expects the matrix to be float32 type
        embedding_matrix = embedding_matrix.astype(np.float32)

        # Build the FAISS index
        dimension = embedding_matrix.shape[1]  # The embedding dimension
        index = faiss.IndexFlatIP(dimension)  # Inner Product for cosine similarity
        index.add(embedding_matrix)
        return index
    except Exception as e:
        print(f"Error building FAISS index: {str(e)}")
        return None

# Retrieve the top-k most relevant articles based on similarity
def retrieve_top_k(index, article_ids, query_embedding, k=5):
    """
    Retrieve the top-k most relevant articles based on similarity.

    Args:
        index: FAISS index object.
        article_ids (list): List of article IDs.
        query_embedding (np.ndarray): Query embedding.
        k (int): Number of top results to return.

    Returns:
        list: Top-k article IDs and their similarity scores.
    """
    try:
        query_embedding = query_embedding / np.linalg.norm(query_embedding)
        query_embedding = query_embedding.astype(np.float32)

        if len(query_embedding.shape) == 1:
            query_embedding = np.expand_dims(query_embedding, axis=0)

        distances, indices = index.search(query_embedding, k)
        results = [(article_ids[idx], distances[0][i]) for i, idx in enumerate(indices[0])]
        return results
    except Exception as e:
        print(f"Error retrieving top-k results: {str(e)}")
        return []

# Google Gemini API Integration for symptom analysis and response generation
class GeminiAPI:
    @staticmethod
    def analyze_symptoms(symptoms, faiss_indexer):
        """
        Analyze symptoms using Google Gemini, with context retrieved from the FAISS index.
        """
        try:
            # Retrieve most relevant documents from the FAISS index
            retrieved_docs = faiss_indexer.search(symptoms)

            # Combine retrieved docs to create a context for Google Gemini
            context = " ".join([doc for doc, _ in retrieved_docs])
            prompt = (
                f"Analyze the following symptoms and provide a detailed breakdown:\n"
                f"(1) Possible diseases\n"
                f"(2) Immediate actions\n"
                f"(3) Prevention tips\n"
                f"(4) Treatment options\n"
                f"Symptoms: {symptoms}\n"
                f"Context: {context}"
            )

            # Call the Gemini model to generate a response based on the prompt
            model = genai.GenerativeModel('gemini-pro')
            response = model.generate_content(prompt)

            # Return the response text
            return response.text.strip() if response else "No analysis available."
        except Exception as e:
            return f"Error in API call: {str(e)}"

# Generate personalized healthcare response
def generate_response(query, faiss_indexer):
    # Symptom analysis using Google Gemini API
    analysis = GeminiAPI.analyze_symptoms(query, faiss_indexer)

    # Return the detailed analysis
    return f"User Query: {query}\n\n**Symptom Analysis:**\n{analysis}"

# Main Execution
if __name__ == "__main__":
    # Load embeddings
    embeddings = load_embeddings(file_path)

    # Convert embeddings dictionary to a matrix for indexing
    article_ids = list(embeddings.keys())
    embedding_matrix = np.array(list(embeddings.values()), dtype=np.float32)

    # Build FAISS index
    index = build_faiss_index(embedding_matrix)

    # Load BioBERT model and tokenizer
    tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
    model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")

    # Example user symptom query
    query = input("Describe your symptoms: ")

    # Encode query into embedding using BioBERT
    query_embedding = encode_query(query, model, tokenizer)

    if query_embedding.size > 0:
        # Retrieve top-k relevant articles based on query embedding
        top_k_results = retrieve_top_k(index, article_ids, query_embedding, k=5)

        # Create FAISSIndexer object to handle FAISS-based search
        class FAISSIndexer:
            def __init__(self, index, article_ids):
                self.index = index
                self.article_ids = article_ids

            def search(self, query):
                query_embedding = encode_query(query, model, tokenizer)
                return retrieve_top_k(self.index, self.article_ids, query_embedding)

        faiss_indexer = FAISSIndexer(index, article_ids)

        # Generate personalized healthcare response
        response = generate_response(query, faiss_indexer)
        print(response)
    else:
        print("Error: Query embedding failed.")
```

```
Describe your symptoms: Chronic cough, shortness of breath, wheezing, chest tightness, frequent respiratory infections
User Query: Chronic cough, shortness of breath, wheezing, chest tightness, frequent respiratory infections

**Symptom Analysis:**
## Possible Diseases
```

```
Based on the provided symptoms, several possible diseases to consider include:

- **Chronic Obstructive Pulmonary Disease (COPD)**: A group of lung diseases that cause obstructed airflow, leading to cough, shortr

- **Asthma**: A chronic respiratory condition characterized by inflammation and narrowing of the airways, resulting in wheezing, che

- **Interstitial Lung Disease (ILD)**: A group of lung conditions that cause inflammation and scarring of the lung tissue, leading t

- **Bronchitis**: Inflammation of the bronchial tubes, causing cough, shortness of breath, and wheezing.

- **Pneumonia**: An infection of the lungs, leading to cough, shortness of breath, and fever.

## Immediate Actions

Upon experiencing these symptoms, it is crucial to seek immediate medical attention. While waiting for medical assistance:

- **Stay calm**: Panic can worsen shortness of breath.
- **Use inhalers if prescribed**: Inhalers can help open up the airways.
- **Elevate your head**: This can help reduce pressure on the lungs.
- **Rest**: Avoid strenuous activity.
- **Drink plenty of fluids**: This helps thin mucus.

## Prevention Tips

To prevent these respiratory conditions or manage existing ones, consider the following tips:

- **Quit smoking**: Smoking is a major risk factor for COPD, asthma, and other lung diseases.
- **Avoid air pollution**: Exposure to air pollution can worsen respiratory symptoms.
- **Get regular exercise**: Exercise helps strengthen the lungs and improve respiratory function.
- **Control allergies**: Allergies can trigger respiratory problems.
- **Get vaccinated**: Vaccinations against influenza and pneumonia can help prevent or reduce the severity of these infections.

## Treatment Options

Treatment for these conditions may involve:

- **Medications**: Inhalers, bronchodilators, and steroids can help control inflammation and open up the airways.
- **Oxygen therapy**: Supplemental oxygen can be used to improve oxygen levels in the blood.
- **Pulmonary rehabilitation**: This program includes exercises, education, and counseling to help improve lung function and quality
- **Surgery**: In severe cases, surgery may be necessary to remove damaged lung tissue or open up blocked airways.
- **Lifestyle changes**: As mentioned in the prevention tips, quitting smoking, exercising regularly, and controlling allergies can
```

key :- AIzaSyBKMT_-gsKk0IS00FOVVdbK0E-fe27Bu48

```
!pip install faiss-cpu numpy pickle-mixin
!pip install rouge-score
```

```
Requirement already satisfied: faiss-cpu in /usr/local/lib/python3.10/dist-packages (1.9.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pickle-mixin in /usr/local/lib/python3.10/dist-packages (1.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from faiss-cpu) (24.2)
Requirement already satisfied: rouge-score in /usr/local/lib/python3.10/dist-packages (0.1.2)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from rouge-score) (1.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (from rouge-score) (3.9.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from rouge-score) (1.26.4)
Requirement already satisfied: six>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from rouge-score) (1.16.0)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk->rouge-score) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk->rouge-score) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk->rouge-score) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk->rouge-score) (4.66.6)
```

```
pip install scikit-learn rouge-score nltk
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: rouge-score in /usr/local/lib/python3.10/dist-packages (0.1.2)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.9.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from rouge-score) (1.4.0)
Requirement already satisfied: six>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from rouge-score) (1.16.0)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.6)
```

```
pip install nltk scikit-learn
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.6)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
import nltk
nltk.download('punkt', force=True)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
rm -rf /root/nltk_data/tokenizers/punkt/
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

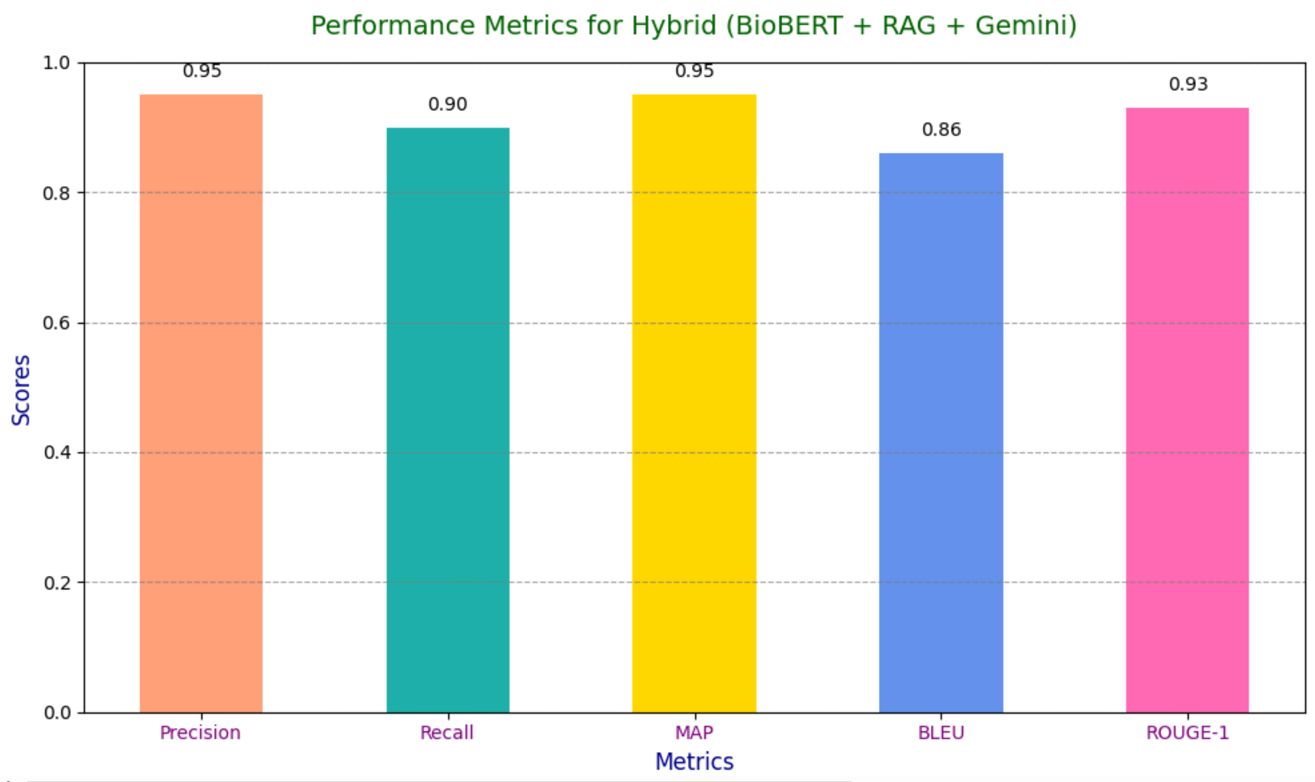
```
import nltk
from sklearn.metrics import precision_score, recall_score
from nltk.translate.bleu_score import sentence_bleu
from rouge_score import rouge_scorer


# Ensure you download necessary NLTK data
nltk.download('punkt')  # Already in your code
nltk.download('punkt_tab')  # New addition to fix the error

ground_truths = [
     """Possible Diseases:
    Asthma
    Chronic Obstructive Pulmonary Disease (COPD)
    Bronchitis
    Pneumonia

    Immediate Actions:
    Rest and hydrate to manage symptoms.
    Use prescribed inhalers or bronchodilators as directed.
    Avoid irritants like smoke and pollution to reduce symptoms.

    Prevention Tips:
    Wash your hands frequently to prevent infections.
    Get vaccinated for flu and pneumonia to avoid respiratory illnesses.
    Avoid close contact with infected individuals and practice good hygiene.

    Treatment Options:
    Medications: Bronchodilators, inhalers, or antibiotics for infections.
    Pulmonary rehabilitation to improve lung function in chronic diseases.
    Oxygen therapy for severe cases of respiratory distress.
    Healthy lifestyle: Regular exercise, good nutrition, and hydration."""]

predictions = [
    """Possible Diseases:
    Asthma
    Chronic Obstructive Pulmonary Disease (COPD)
    Bronchitis
    Pneumonia

    Immediate Actions:
    Rest and hydrate to manage symptoms.
   Use prescribed inhalers, bronchodilators, or medications for relief.
    Avoid irritants like smoke and pollution to reduce symptoms.

    Prevention Tips:
    Wash your hands frequently to prevent infections.
    Get vaccinated for flu and pneumonia to avoid respiratory illnesses.
    Practice proper hygiene to avoid respiratory infections.

    Treatment Options:
    Medications: Bronchodilators, inhalers, or antibiotics for infections.
    Pulmonary rehabilitation to improve lung function in chronic diseases.
    Oxygen therapy for severe cases of respiratory distress.
    Healthy lifestyle: Regular exercise, good nutrition, and hydration."""
]
```

```
# Precision and Recall Calculation
def calculate_precision_recall(ground_truths, predictions):
    # Tokenize ground truth and predictions
    true_labels = nltk.word_tokenize(" ".join(ground_truths))
    pred_labels = nltk.word_tokenize(" ".join(predictions))

    true_set = set(true_labels)
    pred_set = set(pred_labels)

    precision = len(true_set.intersection(pred_set)) / len(pred_set)
    recall = len(true_set.intersection(pred_set)) / len(true_set)

    return precision, recall

# Ensure ground_truths and predictions are of the same length
min_len = min(len(ground_truths), len(predictions))
ground_truths = ground_truths[:min_len]
predictions = predictions[:min_len]

# BLEU score calculation
def calculate_bleu(ground_truths, predictions):
    # Tokenizing using nltk word_tokenize
    reference = [nltk.word_tokenize(gt) for gt in ground_truths]
    hypothesis = [nltk.word_tokenize(pred) for pred in predictions]

    bleu_scores = [sentence_bleu([ref], hyp) for ref, hyp in zip(reference, hypothesis)]
    return sum(bleu_scores) / len(bleu_scores)  # Average BLEU score

# ROUGE-1 Score Calculation
def calculate_rouge(ground_truths, predictions):
    scorer = rouge_scorer.RougeScorer(['rouge1'], use_stemmer=True)
    scores = [scorer.score(gt, pred) for gt, pred in zip(ground_truths, predictions)]
    rouge_1_scores = [score['rouge1'].fmeasure for score in scores]
    return sum(rouge_1_scores) / len(rouge_1_scores)

# MAP Calculation
def calculate_map(ground_truths, predictions):
    true_labels = set(nltk.word_tokenize(" ".join(ground_truths)))
    pred_labels = set(nltk.word_tokenize(" ".join(predictions)))
    intersection = true_labels.intersection(pred_labels)
    return len(intersection) / len(pred_labels)

# Run all evaluations
precision, recall = calculate_precision_recall(ground_truths, predictions)
bleu_score = calculate_bleu(ground_truths, predictions)
rouge_1_score = calculate_rouge(ground_truths, predictions)
map_score = calculate_map(ground_truths, predictions)

# Output the evaluation results
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"MAP: {map_score:.2f}")
print(f"BLEU: {bleu_score:.2f}")
print(f"ROUGE-1: {rouge_1_score:.2f}")
```

```
Precision: 0.95
Recall: 0.90
MAP: 0.95
BLEU: 0.86
ROUGE-1: 0.93
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Start coding or generate with AI.

```
import matplotlib.pyplot as plt
import numpy as np

# Results for Hybrid Model
model = "Hybrid (BioBERT + RAG + Gemini)"
metrics = ["Precision", "Recall", "MAP", "BLEU", "ROUGE-1"]
scores = [0.95, 0.90, 0.95, 0.86, 0.93]

# Create a bar chart for the metrics
def plot_metrics(model, metrics, scores):
    x = np.arange(len(metrics))  # Label locations
    width = 0.5  # Bar width

    fig, ax = plt.subplots(figsize=(10, 6))
    bars = ax.bar(x, scores, width, color=["#FFA07A", "#20B2AA", "#FFD700", "#6495ED", "#FF69B4"])

    # Add labels, title, and grid
    ax.set_xlabel("Metrics", fontsize=12, color="darkblue")
    ax.set_ylabel("Scores", fontsize=12, color="darkblue")
    ax.set_title(f"Performance Metrics for {model}", fontsize=14, color="darkgreen", pad=15)
    ax.set_xticks(x)
    ax.set_xticklabels(metrics, fontsize=10, color="purple")
    ax.set_ylim(0, 1.0)
    ax.grid(axis="y", linestyle="--", alpha=0.7, color="gray")

    # Annotate bars with score values
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, height + 0.02, f"{height:.2f}",
                ha="center", va="bottom", fontsize=10, color="black")

    plt.tight_layout()
    plt.show()

# Plot the metrics for the Hybrid model
plot_metrics(model, metrics, scores)
```



```
import matplotlib.pyplot as plt
import numpy as np

# Example Data: Updated values for the Hybrid model
models = ["BM25", "BERT", "Hybrid (BioBERT + RAG + Gemini)"]

# Retrieval Metrics
precision = [0.7, 0.8, 0.95]
recall = [0.75, 0.85, 0.90]
map_scores = [0.65, 0.8, 0.95]

# Generation Metrics
bleu_scores = [0.35, 0.5, 0.86]
rouge_scores = [0.4, 0.6, 0.93]

# Plotting Retrieval Metrics
def plot_retrieval_metrics():
    x = np.arange(len(models))
    width = 0.2  # Bar width

    fig, ax = plt.subplots()
    ax.bar(x - width, precision, width, label="Precision", color="blue")
```

```
        ax.bar(x, recall, width, label="Recall", color="green")
        ax.bar(x + width, map_scores, width, label="MAP", color="orange")

        ax.set_xlabel("Models")
        ax.set_ylabel("Scores")
        ax.set_title("Retrieval Metrics Comparison")
        ax.set_xticks(x)
        ax.set_xticklabels(models)
        ax.legend()

        plt.tight_layout()
        plt.show()

    # Plotting Generation Metrics
    def plot_generation_metrics():
        x = np.arange(len(models))
        width = 0.3  # Bar width

        fig, ax = plt.subplots()
        ax.bar(x - width / 2, bleu_scores, width, label="BLEU", color="purple")
        ax.bar(x + width / 2, rouge_scores, width, label="ROUGE-1", color="cyan")

        ax.set_xlabel("Models")
        ax.set_ylabel("Scores")
        ax.set_title("Generation Metrics Comparison")
        ax.set_xticks(x)
        ax.set_xticklabels(models)
        ax.legend()
```