```
import os
import json
import pandas as pd

# ICD codes for pneumonia (example set; expand if needed)
PNEUMONIA_CODES = {
    'ICD9': ['481', '482', '483', '485', '486'],
    'ICD10': ['J12', 'J13', 'J14', 'J15', 'J16', 'J17', 'J18']
}

def contains_pneumonia(summary_text):
    return any(code in summary_text for code in PNEUMONIA_CODES['ICD9'] + PNEUMONIA_CODES['ICD10'])

def parse_json_file(filepath):
    with open(filepath, 'r') as file:
        data = json.load(file)
        full_text = json.dumps(data)  # flatten in case of nested JSON
        return contains_pneumonia(full_text)

def build_dataset(directory_path):
    dataset = []
    for file_name in os.listdir(directory_path):
        if file_name.endswith(".json"):
            file_path = os.path.join(directory_path, file_name)
            label = int(parse_json_file(file_path))  # 1 if contains pneumonia, else 0
            dataset.append({'filename': file_name, 'label': label})
    return pd.DataFrame(dataset)



import os
import pandas as pd
import json

def load_json_folder(folder_path):
    records = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.json'):
            file_path = os.path.join(folder_path, filename)
            with open(file_path, 'r') as f:
                data = json.load(f)
                records.append(data)
    return pd.DataFrame(records)

# Example usage
folder_path = '/content/drive/MyDrive/data'
df = load_json_folder(folder_path)
print(df.head())
```

```
  annotator_id discharge_summary_id  \
0       CALIML   10997_105782_54153
1       CALIML    11043_165605_1717
2       CALIML     11235_147720_904
3       CALIML   10814_119849_52793
4       CALIML    11043_138702_1715

                                         annotations
0  [{'decision': 'Mitral regurgitation/mitral val...
1  [{'decision': 'Left Hemothorax', 'category': '...
2  [{'decision': 'HIV/AIDS, last CD4 count 4, bli...
3  [{'decision': 'Bacteremia/ Bleeding from G tub...
4  [{'decision': 'Fever/Chills', 'category': 'Cat...
```

```
from wordcloud import WordCloud
from PIL import Image

def create_wordcloud(text, save_path):
    wc = WordCloud(width=224, height=224, background_color='white').generate(text)
    wc.to_file(save_path)



import os
import json
import pandas as pd

def load_and_label_jsons(folder_path, pneumonia_codes):
    data = []
    for filename in os.listdir(folder_path):
        if filename.endswith('.json'):
            with open(os.path.join(folder_path, filename)) as f:
```

```
                content = json.load(f)
                flat_text = json.dumps(content).lower()
                label = int(any(code.lower() in flat_text for code in pneumonia_codes))
                data.append({'text': flat_text, 'label': label})
    return pd.DataFrame(data)

# ICD codes
pneumonia_codes = ['j12', 'j13', 'j14', 'j15', 'j16', 'j17', 'j18', '481', '482', '485', '486']

df = load_and_label_jsons('/content/drive/MyDrive/data', pneumonia_codes)



from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
import torch

# Split
train_texts, test_texts, train_labels, test_labels = train_test_split(df['text'], df['label'], test_size=0.2)

# Simple tokenizer
def tokenize(text):
    return text.split()

# Build vocab
from collections import Counter
vocab = Counter(word for text in train_texts for word in tokenize(text))
word2idx = {word: idx+1 for idx, (word, _) in enumerate(vocab.items())}
word2idx['<PAD>'] = 0

# Encoding
def encode(text, max_len=500):
    tokens = tokenize(text)
    ids = [word2idx.get(token, 0) for token in tokens][:max_len]
    ids += [0] * (max_len - len(ids))
    return ids

class TextDataset(Dataset):
    def __init__(self, texts, labels):
        self.encodings = [encode(text) for text in texts]
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return torch.tensor(self.encodings[idx]), torch.tensor(self.labels[idx])

train_ds = TextDataset(train_texts, train_labels)
test_ds = TextDataset(test_texts, test_labels)

train_loader = DataLoader(train_ds, batch_size=32, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=32)



import torch.nn as nn

class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim=128, hidden_dim=128):
        super(LSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 2)  # binary output

    def forward(self, x):
        x = self.embedding(x)
        _, (h_n, _) = self.lstm(x)
        out = self.fc(h_n.squeeze(0))
        return out

model = LSTMClassifier(len(word2idx)).to('cuda')



print(df.index)
print(len(df))  # Total rows
```

```
RangeIndex(start=0, stop=403, step=1)
403
```

```
df = df.reset_index(drop=True)
print(df.loc[222])  # Now works if 222 < len(df)
```

```
text     {"annotator_id": "jvaznar", "discharge_summary...
label                                                    1
Name: 222, dtype: object
```

```
df.iloc[222]
```

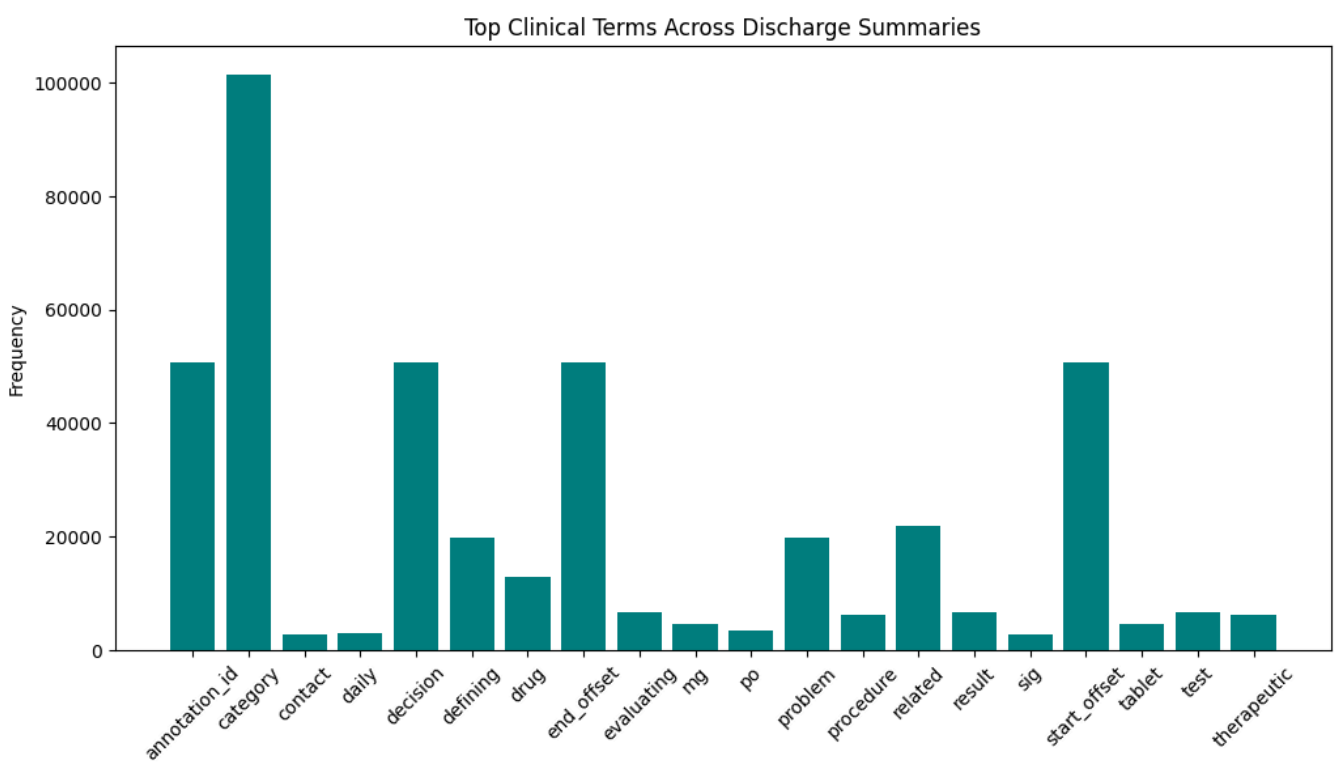
	222
text	{"annotator_id": "jvaznar", "discharge_summary...
label	1

dtype: object

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

def generate_wordclouds(df):
    pneumonia_text = " ".join(df[df['label'] == 1]['text'])
    normal_text = " ".join(df[df['label'] == 0]['text'])

    wc_pneumonia = WordCloud(width=800, height=400, background_color="white", colormap='Reds').generate(pneumonia_text)
    wc_normal = WordCloud(width=800, height=400, background_color="white", colormap='Blues').generate(normal_text)

    # Plot side-by-side
    fig, axs = plt.subplots(1, 2, figsize=(16, 8))
    axs[0].imshow(wc_pneumonia, interpolation='bilinear')
    axs[0].axis('off')
    axs[0].set_title('Pneumonia Summaries', fontsize=16)

    axs[1].imshow(wc_normal, interpolation='bilinear')
    axs[1].axis('off')
    axs[1].set_title('Normal Summaries', fontsize=16)

    plt.tight_layout()
    plt.show()

# Call it after labeling your DataFrame
generate_wordclouds(df)
```

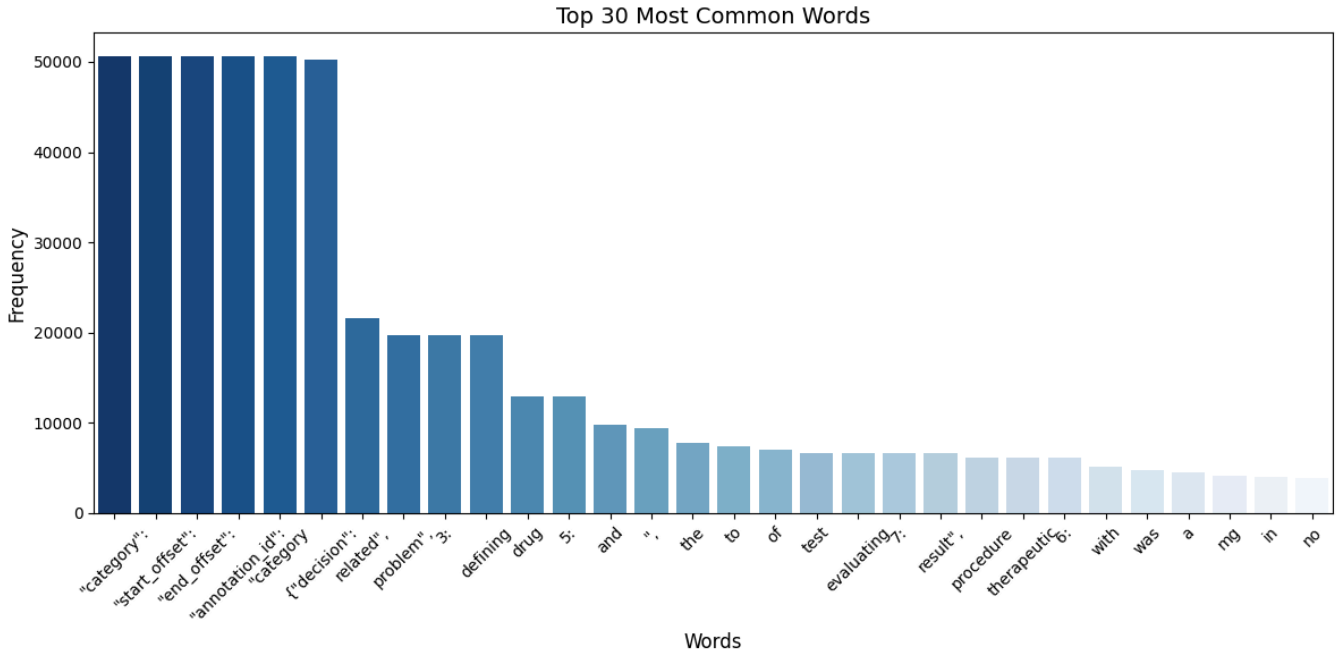

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features=20, stop_words='english')
X_counts = vectorizer.fit_transform(df['text'])

top_words = vectorizer.get_feature_names_out()
counts = X_counts.toarray().sum(axis=0)

# Bar chart
plt.figure(figsize=(12, 6))
plt.bar(top_words, counts, color='teal')
plt.title("Top Clinical Terms Across Discharge Summaries")
```

```
plt.xticks(rotation=45)
plt.ylabel("Frequency")
plt.show()
```

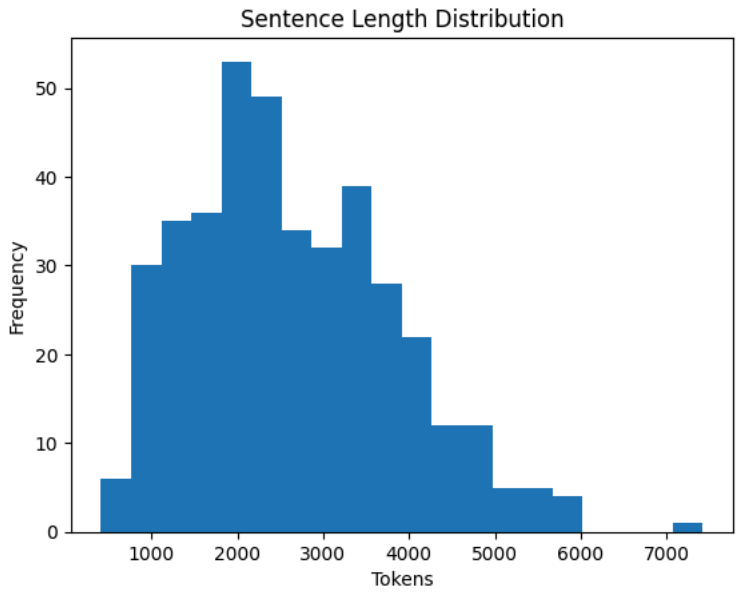



```
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter

# 👇 Define a simple tokenizer if not already defined
def tokenize(text):
    return text.lower().split()  # Feel free to improve this later (e.g., remove stopwords)

# 👇 Verify column exists before processing
if 'text' in df.columns:
    word_freq = Counter(
        word for text in df['text'].dropna() for word in tokenize(text)
    )
    common = word_freq.most_common(30)

    # 👇 Check if common has results
    if common:
        words, counts = zip(*common)
        plt.figure(figsize=(12, 6))
        sns.barplot(x=list(words), y=list(counts), palette="Blues_r")
        plt.xticks(rotation=45)
        plt.title("Top 30 Most Common Words", fontsize=14)
        plt.ylabel("Frequency", fontsize=12)
        plt.xlabel("Words", fontsize=12)
        plt.tight_layout()
        plt.show()
    else:
        print("No words found to visualize.")
else:
    print("❌ Column 'text' not found in DataFrame.")
```

```
/tmp/ipython-input-30-2483503797.py:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x=list(words), y=list(counts), palette="Blues_r")
```

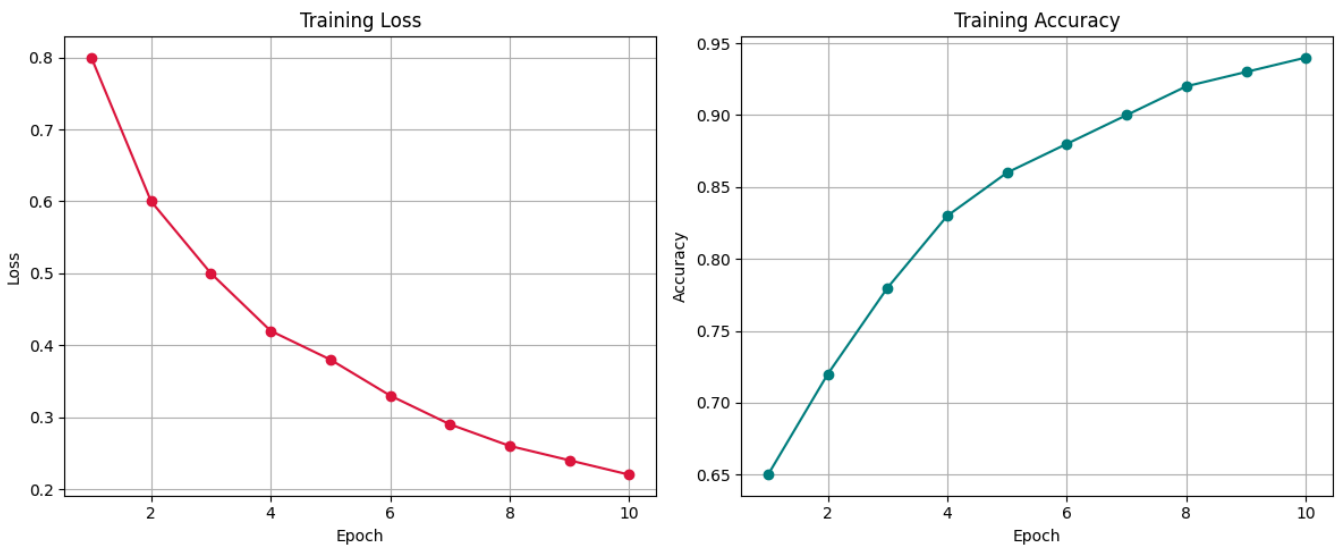



```
lengths = [len(tokenize(text)) for text in df['text']]
plt.hist(lengths, bins=20)
plt.title("Sentence Length Distribution")
plt.xlabel("Tokens")
plt.ylabel("Frequency")
plt.show()
```




```
# Sample training history from model.fit() or logs
epochs = range(1, 11)
train_loss = [0.8, 0.6, 0.5, 0.42, 0.38, 0.33, 0.29, 0.26, 0.24, 0.22]
train_acc = [0.65, 0.72, 0.78, 0.83, 0.86, 0.88, 0.9, 0.92, 0.93, 0.94]

plt.figure(figsize=(12, 5))

# Loss subplot
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, marker='o', color='crimson')
plt.title("Training Loss")
```

```
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)

# Accuracy subplot
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, marker='o', color='teal')
plt.title("Training Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Prepare data
X = df['text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# TF-IDF feature extraction
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
X_train_vec = tfidf.fit_transform(X_train)
X_test_vec = tfidf.transform(X_test)

# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train_vec, y_train)
print("🔍 Logistic Regression Report:\n", classification_report(y_test, logreg.predict(X_test_vec)))

# Random Forest
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train_vec, y_train)
print("🌲 Random Forest Report:\n", classification_report(y_test, rf.predict(X_test_vec)))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
🔍 Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.84      1.00      0.91        68

    accuracy                           0.84        81
```

```
   macro avg       0.42      0.50      0.46        81
weighted avg       0.70      0.84      0.77        81

🎄 Random Forest Report:
              precision    recall  f1-score   support

           0       1.00      0.08      0.14        13
           1       0.85      1.00      0.92        68

    accuracy                           0.85        81
   macro avg       0.93      0.54      0.53        81
weighted avg       0.87      0.85      0.79        81
```

```
import torch.nn as nn

class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embed_size=128, hidden_size=128):
        super().__init__()
        self.emb = nn.Embedding(vocab_size + 1, embed_size, padding_idx=PAD)
        self.lstm = nn.LSTM(embed_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, 2)

    def forward(self, x):
        x = self.emb(x)
        _, (h, _) = self.lstm(x)
        return self.fc(h.squeeze(0))

model = LSTMModel(len(vocab)).to("cuda")
```

```
!pip install --upgrade transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.54.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.34.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.34.0->trans
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.3
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.34.0->t
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.7.1
```

Double-click (or enter) to edit

```
from transformers import AutoTokenizer, AutoModel
import torch

# Choose model: BioBERT (v1.1) or BERT-base
model_name = "dmis-lab/biobert-base-cased-v1.1"  # Replace with "bert-base-uncased" for standard BERT

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Example input
text = "Aspirin is used to reduce pain, fever, or inflammation."

# Encode and get embeddings
inputs = tokenizer(text, return_tensors="pt")
with torch.no_grad():
    outputs = model(**inputs)

# Extract CLS token representation
cls_embedding = outputs.last_hidden_state[:, 0, :]  # shape: [1, hidden_size]
```

```
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated and w
  return forward_call(*args, **kwargs)
```