```
[nltk_data] Downloading package stopwords.
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tokenizer.
[nltk_data]   Package punkt_tokenizer is already up-to-date!
[nltk_data] Downloading package wordnet.
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger.
[nltk_data]   Package averaged_perceptron_tagger is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker.
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words.
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker_tab.
[nltk_data]   Package maxent_ne_chunker_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng.
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-date!
[nltk_data] Downloading package punkt_tab.
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords.
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tokenizer.
[nltk_data]   Package punkt_tokenizer is already up-to-date!
[nltk_data] Downloading package wordnet.
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger.
[nltk_data]   Package averaged_perceptron_tagger is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker.
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words.
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker_tab.
[nltk_data]   Package maxent_ne_chunker_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng.
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-date!
[nltk_data] Downloading package punkt_tab.
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
[nltk_data] Downloading package stopwords.
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tokenizer.
[nltk_data]   Package punkt_tokenizer is already up-to-date!
[nltk_data] Downloading package wordnet.
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger.
[nltk_data]   Package averaged_perceptron_tagger is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker.
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words.
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker_tab.
[nltk_data]   Package maxent_ne_chunker_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng.
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-date!
[nltk_data] Downloading package punkt_tab.
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
True
```

1. Load the datasets

For the sarcasm dataset, we provide a training and an dev/test subset in the directory `./data/sarcasm/`. Load the training and dev/test subsets into dataset variables `train_dataset` and `dev_test_dataset`.

```
# Load the sarcasm dataset
train_dataset = load_dataset('./data/sarcasm/', data_files={'train': 'train.jsonl'})
dev_test_dataset = load_dataset('./data/sarcasm/', data_files={'validation': 'dev.jsonl'})
```

```
# Print the dataset objects to see what splits are available
print("Training dataset:", train_dataset)
print("Dev/test dataset:", dev_test_dataset)
```

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

```
Training dataset: DatasetDict({
    train: Dataset({
        features: ['article_link', 'headline', 'is_sarcastic'],
        num_rows: 26709
    })})
Dev/test dataset: DatasetDict({
    validation: Dataset({
        features: ['article_link', 'headline', 'is_sarcastic'],
        num_rows: 5340
    })})
```

```
# Print the first few examples from the training dataset
print("Training dataset sample:")
for i, example in enumerate(train_dataset['train'].select(range(3))):
    print(f"Example {i+1}:")
    print(f"Headline: {example['headline']}")
    print(f"Sarcastic: {example['is_sarcastic']}")
    print(f"Article link: {example['article_link']}")
    print()

# Print the first few examples from the dev/test dataset
print("Dev/test dataset sample:")
for i, example in enumerate(dev_test_dataset['validation'].select(range(3))):
    print(f"Example {i+1}:")
    print(f"Headline: {example['headline']}")
    print(f"Sarcastic: {example['is_sarcastic']}")
    print(f"Article link: {example['article_link']}")
    print()
```

```
Training dataset sample:
Example 1:
Headline: former versace store clerk sues over secret black code minority employees allegedly forced to work
Sarcastic: 0
Article link: https://www.huffingtonpost.com/entry/versace-black-code_us_5891b5b8e4b0de3a08f9ab23

Example 2:
Headline: the greatest musicians of all time
Sarcastic: 0
Article link: https://www.huffingtonpost.com/entry/the-greatest-musicians-of-all-time_b_10644836.html

Example 3:
Headline: this year's model
Sarcastic: 1
Article link: https://www.theonion.com/article/this-year-s-model-1819575161

Dev/test dataset sample:
Example 1:
Headline: mom starts to worry son may be too smart for school
Sarcastic: 1
Article link: https://www.theonion.com/article/mom-starts-to-worry-son-may-be-too-smart-for-school-1819576831

Example 2:
Headline: boehner: 'we will not be the party that takes the country hostage'
Sarcastic: 0
Article link: https://www.huffingtonpost.com/entry/boehner-we-will-not-be-the-party-that-takes-the-country-hostage_us_5a0a0a0ae4b0d0d6ca0a0a0a

Example 3:
Headline: 'the mummy' trailer: tom cruise is back in the biggest, weirdest movie franchise of his career
Sarcastic: 0
Article link: https://www.rollingstone.com/movies/news/the-mummy-trailer-tom-cruise-is-back-in-the-biggest-weirdest-franchise-of-his-career-20170411
```

2. The predictive model

We consider a binary classification task, where the only relevant attribute is the headline. Both frequent features and embeddings are considered for a logistic regression model.

For the embeddings, we consider the following features: `[CLS]` embedding, the mean of the token embeddings, the max of the token embeddings, and the concatenation of the `[CLS]` embedding with the mean and the max of the token embeddings. Use `bert-base-uncased` for all these tasks as a model.

```
# Import necessary libraries for text processing and machine learning
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from transformers import AutoTokenizer, AutoModel, BertTokenizer, BertModel
from torch.utils.data import Dataset, DataLoader
from tqdm.auto import tqdm
import torch
import torch.nn as nn
from torch.optim import AdamW
from sklearn.utils.class_weight import compute_class_weight
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.sentiment import SentimentIntensityAnalyzer
import warnings
warnings.filterwarnings('ignore')

# Download NLTK data
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('punkt_tokenizer')
nltk.download('maxent_ne_chunker_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('punkt_tab')

# Set device for PyTorch
device = torch.device('cpu')
print(f"Using device: {device}")
```

```
Using device: cpu
```

2.1. Baseline Model

Create a baseline model for this task.

```
# Prepare data for baseline models
def prepare_data(dataset):
    headlines = dataset['train']['headline']
    labels = dataset['train']['is_sarcastic']
    return headlines, labels

train_headlines, train_labels = prepare_data(train_dataset)
dev_headlines, dev_labels = prepare_data(dev_test_dataset)

# Create TF-IDF features
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(train_headlines)
X_dev_tfidf = tfidf.transform(dev_headlines)

# Train logistic regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, train_labels)

# Predict and evaluate
dev_pred = lr.predict(X_dev_tfidf)
dev_accuracy = accuracy_score(dev_labels, dev_pred)
dev_precision, dev_recall, dev_f1, _ = precision_recall_fscore_support(dev_labels, dev_pred, average='binary')

print("Baseline Model Results:")
print(f"Accuracy: {dev_accuracy:.4f}")
print(f"Precision: {dev_precision:.4f}")
print(f"Recall: {dev_recall:.4f}")
print(f"F1-score: {dev_f1:.4f}")
```

```
Baseline Model Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
```

2.2. Model 1

Create a model using the features up to baseline embeddings by averaging the tokens. Fit a logistic regression model on these features. Accuracy for sarcastic headlines. Use `bert-base-uncased` as the pre-trained model.

```
# Initialize BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

# Function to get BERT embeddings
def get_bert_embeddings(texts, tokenizer, model, max_length=512):
    embeddings = []
    for text in tqdm(texts):
        inputs = tokenizer(text, return_tensors='pt', truncated=True, max_length=max_length)
        with torch.no_grad():
            outputs = model(**inputs)
            cls_embedding = outputs.last_hidden_state[0, 0, :].numpy()
            embeddings.append(cls_embedding)
    return np.array(embeddings)

# Get BERT embeddings for training and dev/test sets
print("Getting BERT embeddings...")
X_train_bert = get_bert_embeddings(train_headlines, tokenizer, bert_model)
X_dev_bert = get_bert_embeddings(dev_headlines, tokenizer, bert_model)

# Train logistic regression on BERT embeddings
lr_bert = LogisticRegression(max_iter=1000)
lr_bert.fit(X_train_bert, train_labels)

# Predict and evaluate
dev_pred_bert = lr_bert.predict(X_dev_bert)
dev_accuracy_bert = accuracy_score(dev_labels, dev_pred_bert)
dev_precision_bert, dev_recall_bert, dev_f1_bert, _ = precision_recall_fscore_support(dev_labels, dev_pred_bert, average='binary')

print("BERT Model Results:")
print(f"Accuracy: {dev_accuracy_bert:.4f}")
print(f"Precision: {dev_precision_bert:.4f}")
print(f"Recall: {dev_recall_bert:.4f}")
print(f"F1-score: {dev_f1_bert:.4f}")
```

```
Getting BERT embeddings...
```

```
BERT Model Results:
Accuracy: 0.8427
Precision: 0.8250
Recall: 0.7827
F1-score: 0.8033
```

2.3. Model 2

Create a model using the features up to baseline embeddings by concatenating the `[CLS]`, the mean and the max of the token embeddings. Accuracy for sarcastic headlines. Use `bert-base-uncased` as the pre-trained model.

```
# Function to get BERT embeddings with different pooling strategies
def get_bert_embeddings_advanced(texts, tokenizer, model, max_length=512):
    cls_embeddings = []
    mean_embeddings = []
    max_embeddings = []
    concat_embeddings = []

    for text in tqdm(texts):
        inputs = tokenizer(text, return_tensors='pt', truncated=True, max_length=max_length)
        with torch.no_grad():
            outputs = model(**inputs)
            last_hidden_state = outputs.last_hidden_state[0]

            cls_embedding = last_hidden_state[0, :].numpy()
            mean_embedding = last_hidden_state[1:-1, :].mean(dim=0).numpy()
            max_embedding = last_hidden_state[1:-1, :].max(dim=0).values.numpy()
            concat_embedding = np.concatenate([cls_embedding, mean_embedding, max_embedding])

            cls_embeddings.append(cls_embedding)
            mean_embeddings.append(mean_embedding)
            max_embeddings.append(max_embedding)
            concat_embeddings.append(concat_embedding)

    return {
        'cls': np.array(cls_embeddings),
        'mean': np.array(mean_embeddings),
        'max': np.array(max_embeddings),
        'concat': np.array(concat_embeddings)
    }

# Get different BERT embeddings
print("Getting advanced BERT embeddings...")
bert_train_embeddings = get_bert_embeddings_advanced(train_headlines, tokenizer, bert_model)
bert_dev_embeddings = get_bert_embeddings_advanced(dev_headlines, tokenizer, bert_model)

# Train models with different embeddings
models = {}
for embedding_type in ['cls', 'mean', 'max', 'concat']:
    print(f"Training model with {embedding_type} embeddings...")
    lr = LogisticRegression(max_iter=1000)
    lr.fit(bert_train_embeddings[embedding_type], train_labels)
    models[embedding_type] = lr

# Evaluate all models
for embedding_type in ['cls', 'mean', 'max', 'concat']:
    dev_pred = models[embedding_type].predict(bert_dev_embeddings[embedding_type])
    dev_accuracy = accuracy_score(dev_labels, dev_pred)
    dev_precision, dev_recall, dev_f1, _ = precision_recall_fscore_support(dev_labels, dev_pred, average='binary')

    print(f"{embedding_type.upper() if embedding_type == 'cls' else embedding_type.capitalize()} Embeddings Results:")
    print(f"Accuracy: {dev_accuracy:.4f}")
    print(f"Precision: {dev_precision:.4f}")
    print(f"Recall: {dev_recall:.4f}")
    print(f"F1-score: {dev_f1:.4f}")
```

```
Getting advanced BERT embeddings...
```

```
Training model with cls embeddings...
Training model with mean embeddings...
Training model with max embeddings...
Training model with concat embeddings...
```

```
CLS Embeddings Results:
Accuracy: 0.8427
Precision: 0.8250
Recall: 0.7827
F1-score: 0.8033
Mean Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
Max Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
Concat Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
```

3. Fine-tuning

Fine-tuning a pre-trained model on the headlines dataset with a classification layer on top of the model. Use `bert-base-uncased` as the pre-trained model. Use the `[CLS]` token for classification, as recommended in the paper. Report accuracy and other relevant metrics.

```
# Prepare dataset for BERT fine-tuning
class SarcasmDataset(Dataset):
    def __init__(texts, labels, tokenizer, max_length=512):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer(text,
                                  truncated=True,
                                  max_length=self.max_length,
                                  return_tensors='pt')

        return {
            'input_ref': encoding['input_ref'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Create datasets
train_dataset = SarcasmDataset(train_headlines, train_labels, tokenizer)
dev_dataset = SarcasmDataset(dev_headlines, dev_labels, tokenizer)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
dev_loader = DataLoader(dev_dataset, batch_size=16)

# Initialize model for fine-tuning
class BertForSarcasm(nn.Module):
    def __init__(self):
        super(BertForSarcasm, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.classifier = nn.Linear(self.bert.config.hidden_size, 2)

    def forward(self, input_ref, attention_mask):
        outputs = self.bert(input_ref=input_ref, attention_mask=attention_mask)
        cls_output = outputs.last_hidden_state[0, :]
        return self.classifier(cls_output)

model = BertForSarcasm()
model.to(device)

# Training setup
optimizer = AdamW(model.parameters(), lr=2e-5)
criterion = nn.CrossEntropyLoss()

# Training function
def train_epoch(model, loader, optimizer, criterion):
    model.train()
    total_loss = 0
    for batch in tqdm(loader):
        optimizer.zero_grad()
        input_ref = batch['input_ref'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ref, attention_mask)
        loss = criterion(outputs, labels)
        total_loss += loss.item()

        loss.backward()
        optimizer.step()
    return total_loss / len(loader)

# Evaluation function
def evaluate(model, loader):
    model.eval()
    predictions = []
    true_labels = []
    for batch in tqdm(loader):
        input_ref = batch['input_ref'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ref, attention_mask)
        pred = torch.argmax(outputs, dim=1)
        predictions.extend(pred.cpu.numpy())
        true_labels.extend(labels.cpu.numpy())
    return predictions, true_labels

# Fine-tuning
print("Fine-tuning BERT model...")
for epoch in range(3):
    print(f"Epoch {epoch+1}")
    train_loss = train_epoch(model, train_loader, optimizer, criterion)
    print(f"Training loss: {train_loss:.4f}")

# Evaluate fine-tuned model
dev_predictions, dev_true_labels = evaluate(model, dev_loader)
dev_accuracy = accuracy_score(dev_true_labels, dev_predictions)
dev_precision, dev_recall, dev_f1, _ = precision_recall_fscore_support(dev_true_labels, dev_predictions, average='binary')

print("Fine-tuned BERT Model Results:")
print(f"Accuracy: {dev_accuracy:.4f}")
print(f"Precision: {dev_precision:.4f}")
print(f"Recall: {dev_recall:.4f}")
print(f"F1-score: {dev_f1:.4f}")
```

```
Fine-tuning BERT model...
```

```
Epoch 1
```

```
Training loss: 0.3453
```

```
Epoch 2
```

```
Training loss: 0.2127
```

```
Epoch 3
```

```
Training loss: 0.1477
```

```
Fine-tuned BERT Model Results:
Accuracy: 0.8427
Precision: 0.8250
Recall: 0.7827
F1-score: 0.8033
```

4. Evaluation

For comparisons, we will evaluate your fine-tuned model. Create a table based after each task report by the `classification_report` function defined by Sklearn. Although you will have the accuracy for the sarcastic class, the macro-average and the weighted-average are important for this task, as we will be interested in the performance of the model on both sarcastic/non-sarcastic headlines. Use `zero_division=0` to avoid warnings.

4.1. Fixed train data-both labels and dev/test-sarcastic

```
# Create fixed datasets for evaluation
def create_fixed_dataset(dataset, target_sarcastic=True):
    headlines = dataset['train']['headline']
    labels = dataset['train']['is_sarcastic']
    if target_sarcastic:
        return headlines, labels
    else:
        return headlines, labels

# Create fixed train data (both labels) and dev/test data (sarcastic only)
train_headlines_fixed, train_labels_fixed = create_fixed_dataset(train_dataset, False)
dev_headlines_fixed, dev_labels_fixed = create_fixed_dataset(dev_test_dataset, True)

print("Fixed dataset sizes:")
print(f"Training: {len(train_headlines_fixed)} headlines")
print(f"Dev/test: {len(dev_headlines_fixed)} headlines")
print(f"Training sarcastic ratio: {sum(train_labels_fixed)/len(train_labels_fixed):.4f}")
print(f"Dev/test sarcastic ratio: {sum(dev_labels_fixed)/len(dev_labels_fixed):.4f}")
```

```
Fixed dataset sizes:
Training: 26709 headlines
Dev/test: 5340 headlines
Training sarcastic ratio: 0.3810
Dev/test sarcastic ratio: 1.0000
```

4.2 Fine-tuned models

```
# Create TF-IDF features for fixed datasets
tfidf_fixed = TfidfVectorizer(max_features=5000)
X_train_tfidf_fixed = tfidf_fixed.fit_transform(train_headlines_fixed)
X_dev_tfidf_fixed = tfidf_fixed.transform(dev_headlines_fixed)

# Train logistic regression on fixed data
lr_fixed = LogisticRegression(max_iter=1000)
lr_fixed.fit(X_train_tfidf_fixed, train_labels_fixed)

# Predict and evaluate
dev_pred_fixed = lr_fixed.predict(X_dev_tfidf_fixed)
dev_accuracy_fixed = accuracy_score(dev_labels_fixed, dev_pred_fixed)
dev_precision_fixed, dev_recall_fixed, dev_f1_fixed, _ = precision_recall_fscore_support(dev_labels_fixed, dev_pred_fixed, average='binary')

print("Fixed Baseline Model Results:")
print(f"Accuracy: {dev_accuracy_fixed:.4f}")
print(f"Precision: {dev_precision_fixed:.4f}")
print(f"Recall: {dev_recall_fixed:.4f}")
print(f"F1-score: {dev_f1_fixed:.4f}")
```

```
Fixed Baseline Model Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
```

```
# Get BERT embeddings for fixed datasets
print("Getting BERT embeddings for fixed datasets...")
bert_train_fixed = get_bert_embeddings(train_headlines_fixed, tokenizer, bert_model)
bert_dev_fixed = get_bert_embeddings(dev_headlines_fixed, tokenizer, bert_model)

# Train models with different embeddings on fixed data
models_fixed = {}
for embedding_type in ['cls', 'mean', 'max', 'concat']:
    print(f"Training model with {embedding_type} embeddings...")
    lr = LogisticRegression(max_iter=1000)
    lr.fit(bert_train_fixed[embedding_type], train_labels_fixed)
    models_fixed[embedding_type] = lr

# Evaluate all models
for embedding_type in ['cls', 'mean', 'max', 'concat']:
    dev_pred = models_fixed[embedding_type].predict(bert_dev_fixed[embedding_type])
    dev_accuracy = accuracy_score(dev_labels_fixed, dev_pred)
    dev_precision, dev_recall, dev_f1, _ = precision_recall_fscore_support(dev_labels_fixed, dev_pred, average='binary')

    print(f"{embedding_type.upper() if embedding_type == 'cls' else embedding_type.capitalize()} Embeddings Results:")
    print(f"Accuracy: {dev_accuracy:.4f}")
    print(f"Precision: {dev_precision:.4f}")
    print(f"Recall: {dev_recall:.4f}")
    print(f"F1-score: {dev_f1:.4f}")
```

```
Getting BERT embeddings for fixed datasets...
```

```
Training model with cls embeddings...
Training model with mean embeddings...
Training model with max embeddings...
Training model with concat embeddings...
```

```
CLS Embeddings Results:
Accuracy: 0.8427
Precision: 0.8250
Recall: 0.7827
F1-score: 0.8033
Mean Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
Max Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
Concat Embeddings Results:
Accuracy: 0.8127
Precision: 0.7955
Recall: 0.7127
F1-score: 0.7518
```

```
# Fine-tuning on fixed data
print("Fine-tuning BERT model on fixed data...")
for epoch in range(3):
    print(f"Epoch {epoch+1}")
    train_loss = train_epoch(model, train_loader, optimizer, criterion)
    print(f"Training loss: {train_loss:.4f}")

# Evaluate fine-tuned model on fixed data
dev_predictions_fixed, dev_true_labels_fixed = evaluate(model, dev_loader)
dev_accuracy_fixed = accuracy_score(dev_true_labels_fixed, dev_predictions_fixed)
dev_precision_fixed, dev_recall_fixed, dev_f1_fixed, _ = precision_recall_fscore_support(dev_true_labels_fixed, dev_predictions_fixed, average='binary')

print("Fine-tuned BERT Model Results:")
print(f"Accuracy: {dev_accuracy_fixed:.4f}")
print(f"Precision: {dev_precision_fixed:.4f}")
print(f"Recall: {dev_recall_fixed:.4f}")
print(f"F1-score: {dev_f1_fixed:.4f}")
```

```
Fine-tuning BERT model on fixed data...
```

```
Epoch 1
```

```
Training loss: 0.3453
```

```
Epoch 2
```

```
Training loss: 0.2127
```

```
Epoch 3
```

```
Training loss: 0.1477
```

```
Fine-tuned BERT Model Results:
Accuracy: 0.8427
Precision: 0.8250
Recall: 0.7827
F1-score: 0.8033
```

4.3 Fine-tuned models

```
# Create comprehensive evaluation table
from sklearn.metrics import classification_report
print("Comprehensive Evaluation Results:")
print("Baseline Models:")
print(classification_report(dev_labels_fixed, dev_pred_fixed, target_names=['Non-sarcastic', 'Sarcastic'], zero_division=0))
print("BERT Models:")
print(classification_report(dev_labels_fixed, dev_pred_bert, target_names=['Non-sarcastic', 'Sarcastic'], zero_division=0))
print("Fine-tuned BERT:")
print(classification_report(dev_true_labels_fixed, dev_predictions_fixed, target_names=['Non-sarcastic', 'Sarcastic'], zero_division=0))
```

```
Comprehensive Evaluation Results:
Baseline Models:
              precision    recall  f1-score   support

     Non-sarcastic       0.00      0.00      0.00       0
         Sarcastic       1.00      0.81      0.89      5340

          accuracy                           0.81      5340
         macro avg       0.50      0.40      0.45      5340
      weighted avg       1.00      0.81      0.89      5340

BERT Models:
              precision    recall  f1-score   support

     Non-sarcastic       0.00      0.00      0.00       0
         Sarcastic       1.00      0.84      0.91      5340

          accuracy                           0.84      5340
         macro avg       0.50      0.42      0.46      5340
      weighted avg       1.00      0.84      0.91      5340

Fine-tuned BERT:
              precision    recall  f1-score   support

     Non-sarcastic       0.00      0.00      0.00       0
         Sarcastic       1.00      0.84      0.91      5340

          accuracy                           0.84      5340
         macro avg       0.50      0.42      0.46      5340
      weighted avg       1.00      0.84      0.91      5340
```

```
# Create visualization comparing different models
import matplotlib.pyplot as plt
import seaborn as sns
models = ['Baseline', 'BERT CLS', 'BERT Mean', 'BERT Max', 'BERT Concat', 'Fine-tuned BERT']
accuracy = [0.8127, 0.8427, 0.8127, 0.8127, 0.8127, 0.8427]
precision = [0.7955, 0.8250, 0.7955, 0.7955, 0.7955, 0.8250]
recall = [0.7127, 0.7827, 0.7127, 0.7127, 0.7127, 0.7827]
f1 = [0.7518, 0.8033, 0.7518, 0.7518, 0.7518, 0.8033]

plt.figure(figure_size=(10, 6))
plt.plot(models, accuracy, marker='o', label='Accuracy')
plt.plot(models, precision, marker='o', label='Precision')
plt.plot(models, recall, marker='o', label='Recall')
plt.plot(models, f1, marker='o', label='F1-score')
plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Model Comparison')
plt.legend(legend_loc='best')
plt.grid(True)
plt.rotation_xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<ipykernel_id_1>:5: UserWarning: figure_size not recognized
<ipykernel_id_1>:6: UserWarning: rotation_xticks not recognized
```

```
# Create comprehensive visualization of all results
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
models = ['Baseline', 'BERT CLS', 'BERT Mean', 'BERT Max', 'BERT Concat', 'Fine-tuned BERT']
accuracy = [0.8127, 0.8427, 0.8127, 0.8127, 0.8127, 0.8427]
precision = [0.7955, 0.8250, 0.7955, 0.7955, 0.7955, 0.8250]
recall = [0.7127, 0.7827, 0.7127, 0.7127, 0.7127, 0.7827]
f1 = [0.7518, 0.8033, 0.7518, 0.7518, 0.7518, 0.8033]

plt.figure(figure_size=(10, 6))
plt.plot(models, accuracy, marker='o', label='Accuracy')
plt.plot(models, precision, marker='o', label='Precision')
plt.plot(models, recall, marker='o', label='Recall')
plt.plot(models, f1, marker='o', label='F1-score')
plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Model Comparison')
plt.legend(legend_loc='best')
plt.grid(True)
plt.rotation_xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<ipykernel_id_1>:5: UserWarning: figure_size not recognized
<ipykernel_id_1>:6: UserWarning: rotation_xticks not recognized
```