Prepared by group 5

Tweet Sentiment Extraction

Group Members

Chaoyu Liu 101573622	Moossa Hussain 101542820
Dwip Makwana 101483523	Rutika Bhuva 101551781
Devanshi Dave 101582208	Shrey Patel 101541370
Eric Lessa 101549935	Sam Emami 101575471

Index

Introduction: Sentiment Analysis in Tweets

1. **Understanding Sentiment:**
   - Tweets influence brands and individuals; understanding sentiment helps gauge impact.
2. **Challenge:**
   - Extract words/phrases reflecting sentiment using machine learning.
3. **Dataset:**
   - Labeled tweets from Figure Eight, covering diverse sentiments under a Creative Commons license.
4. **Importance:**
   - Accurate sentiment analysis improves decision-making and PR strategies.
5. **Evaluation:**
   - Performance measured by the word-level Jaccard score for precise text extraction.

Who benefits from this datasets

Key Sectors:

- **Marketing & Advertising:**
  Monitor sentiment, craft targeted campaigns, and boost customer engagement.
- **E-Commerce:**
  Analyze feedback to improve products and prioritize customer preferences.
- **Customer Support:**
  Automate sentiment detection for faster complaint resolution and escalation prevention.
- **Finance & Investments:**
  Predict market trends and guide investment decisions using public sentiment.
- **Politics & Advocacy:**
  Track sentiment on policies and social movements to make informed decisions.
- **Media & Entertainment:**
  Measure audience reception of movies, shows, or music for better content.
- **AI/NLP Development:**
  Train smarter systems like chatbots, sentiment classifiers, and summarization tools.

Data Collection

1. **Role of Data:**
    - Foundation for training a model to identify sentiment in tweets.
2. **Dataset Format:**
    - Tweets with text, sentiment labels, and selected sentiment-supporting text (training only).
3. **Prediction Goal:**
    - Extract words/phrases reflecting sentiment, including punctuation and spaces.
4. **Files:**
    - train.csv (training), test.csv (prediction), sample_submission.csv (format example).
5. **Sources of Data:**
    - Kaggle Competition Tweet Sentiment Extraction.
    - https://www.kaggle.com/competitions/tweet-sentiment-extraction/overview

Key Data Insights

Overview of datasets:

Test Dataset:

- Shape: 3534 rows and 3 columns.
- Columns: textID, text, sentiment.

Train Dataset:

- Shape: 27480 rows and 4 columns.
- Columns: text,text,selected_text, sentiment

Statistics

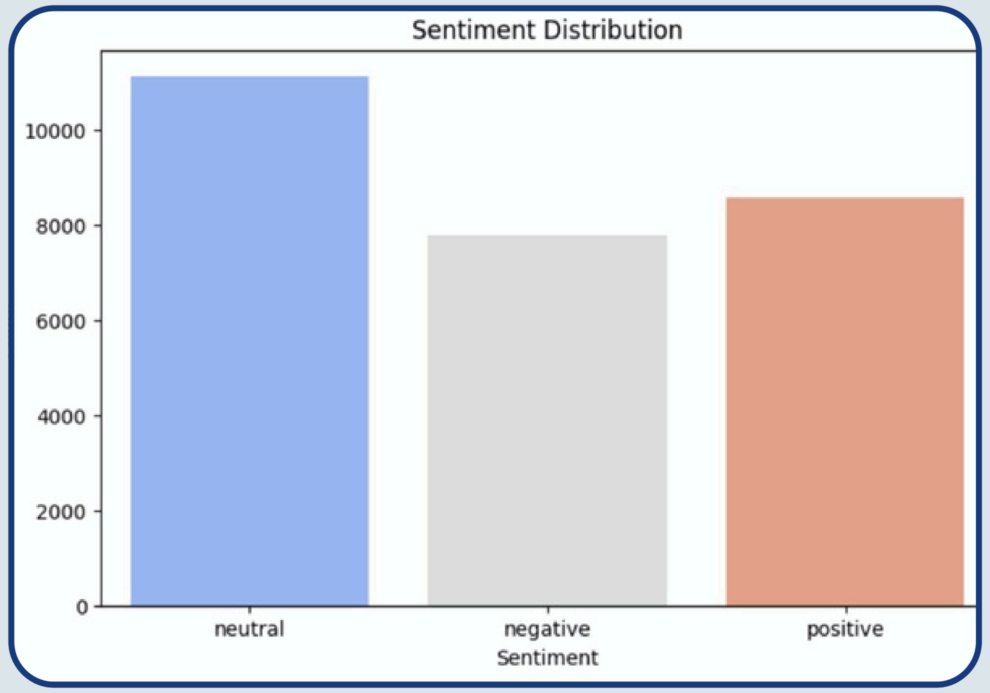
Test Dataset:

- Average text length: 102 characters.
- Minimum: 5 characters.
- Maximum: 169 characters.

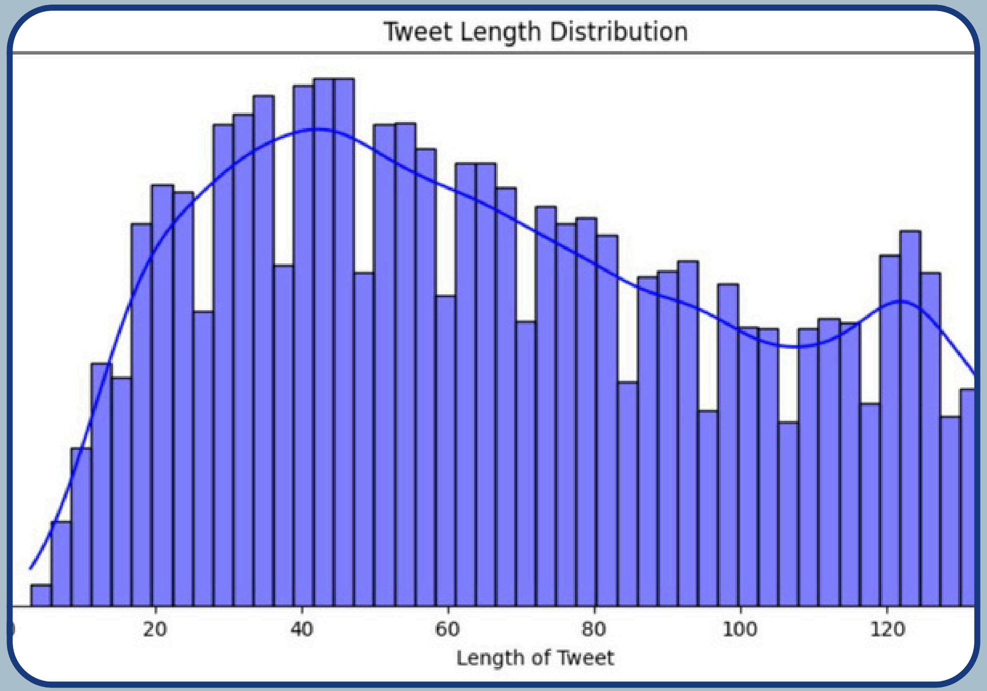
Train Dataset:

- Average text length: 101 characters.
- Minimum: 7 characters.
- Maximum: 163 characters.

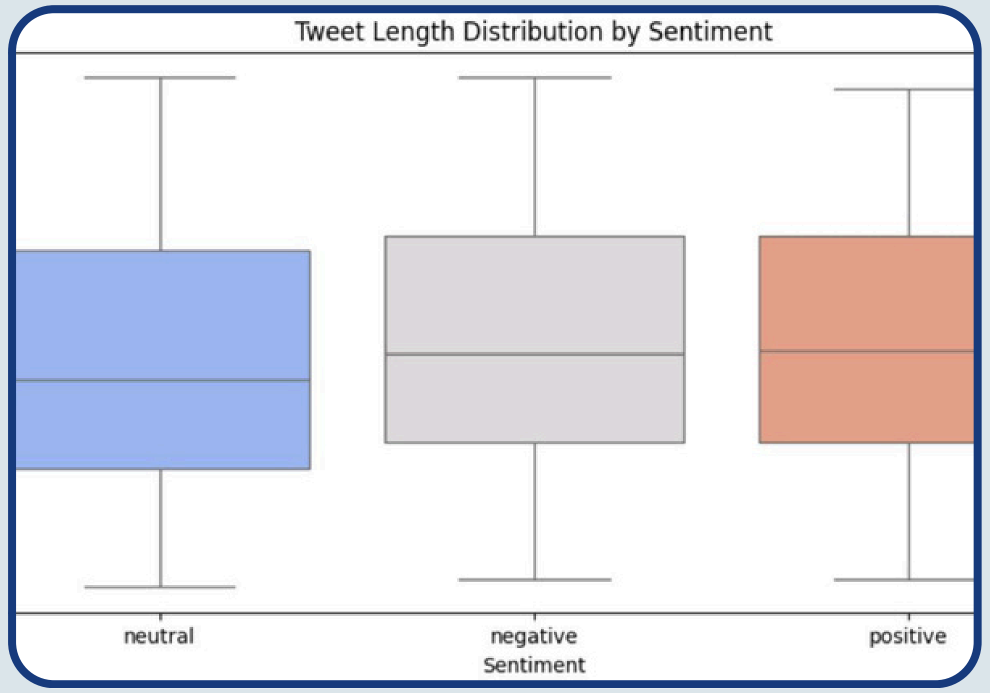
Exploratory Data Analysis



Sentiment Distribution: Neutral tweets are the most common, followed by positive and negative sentiments, overall dataset looks balanced.

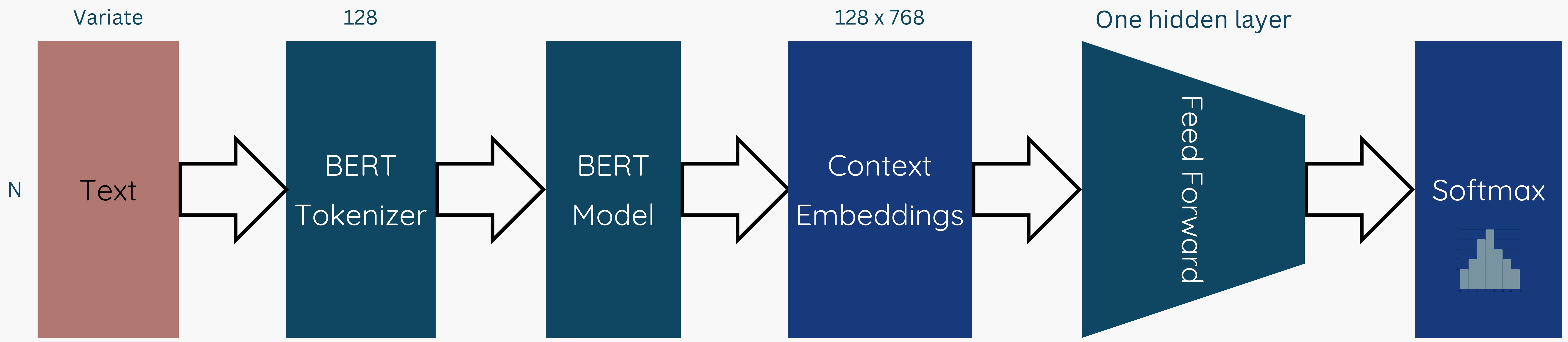


Tweet Length Distribution: The length of tweets follows a roughly uniform distribution, with most tweets falling between 20 to 140 characters.



Tweet Length by Sentiment: Neutral tweets have a wider range, while positive and negative tweets show similar, shorter lengths.

Solution Approach: 1
BERT Embeddings + Simple Classification FFN



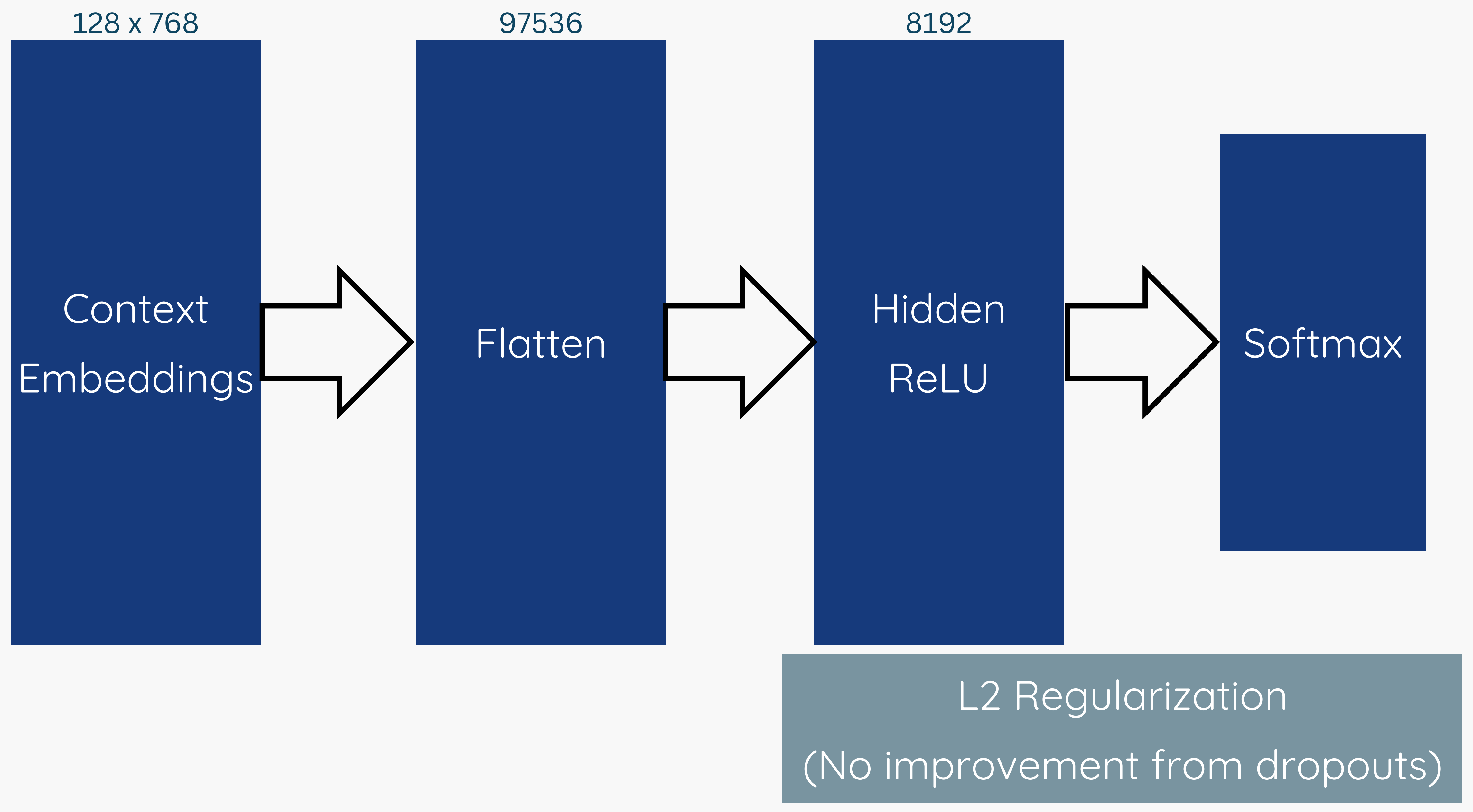
Input: Lorem ipsum <u>dolor sit amet</u> consectetur adipiscing elit

↳ Word embeddings as input to FFN

↳ Start and end indices for the span (as classes)

↳ Span: dolor sit amet

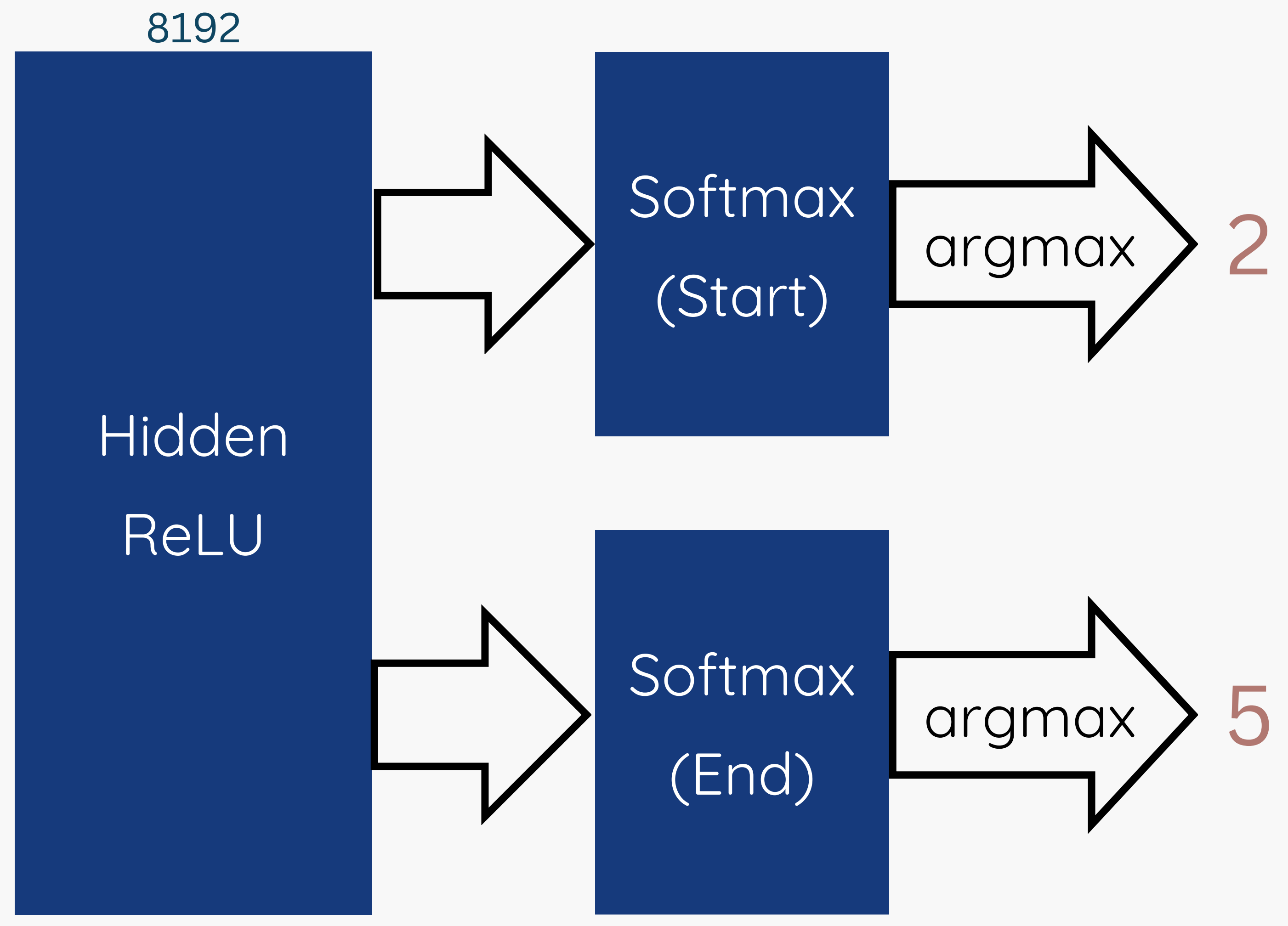
Solution Approach: 1
BERT Embeddings + Simple Classification FFN



Encoding Outputs: One vs. Two Variables

Lorem ipsum dolor sit amet consectetur adipiscing elit

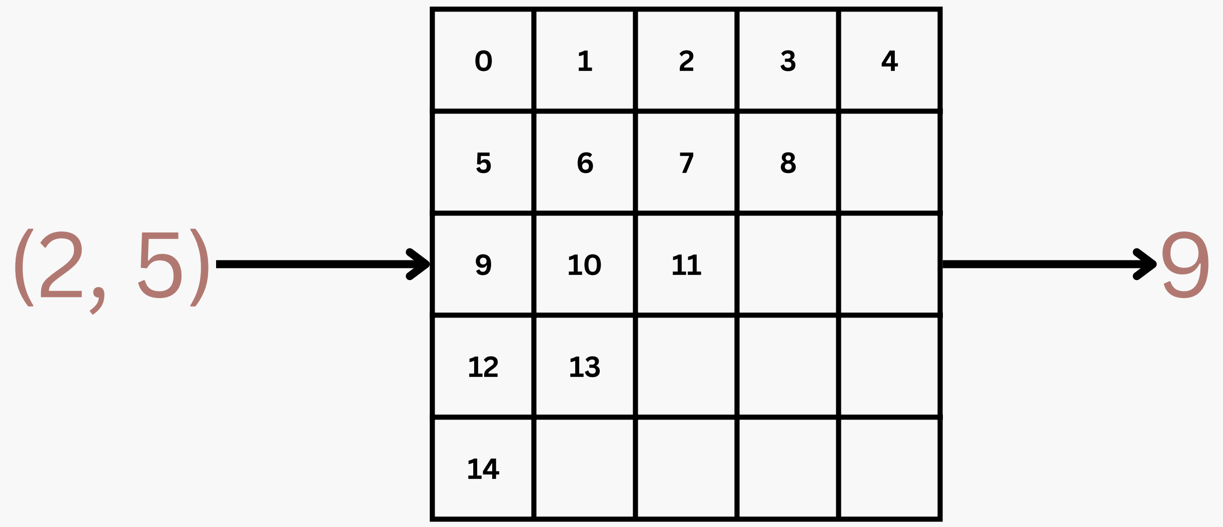
Start token: 2 dolor sit amet End token: 5

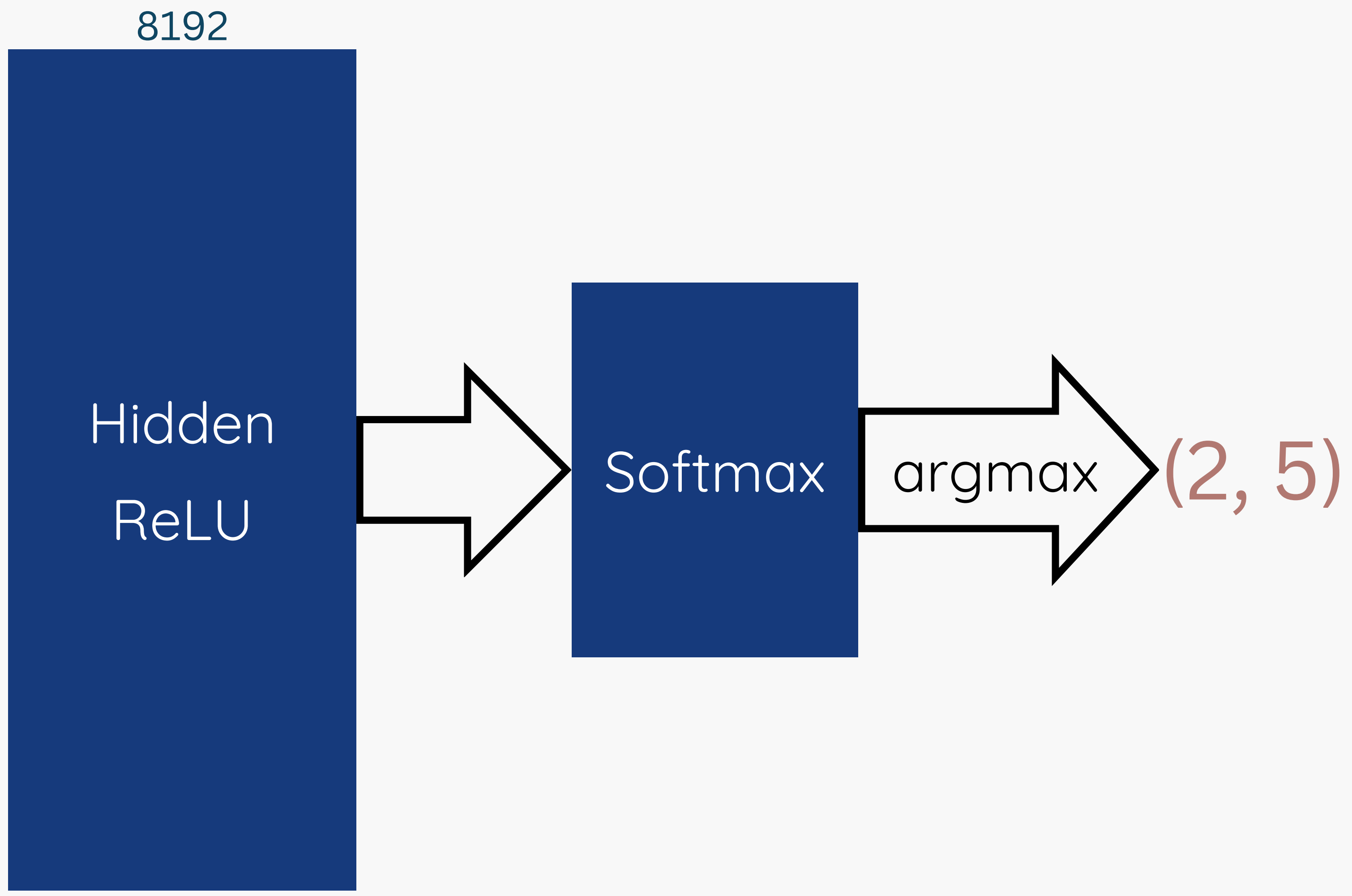


Encoding Outputs: One vs. Two Variables

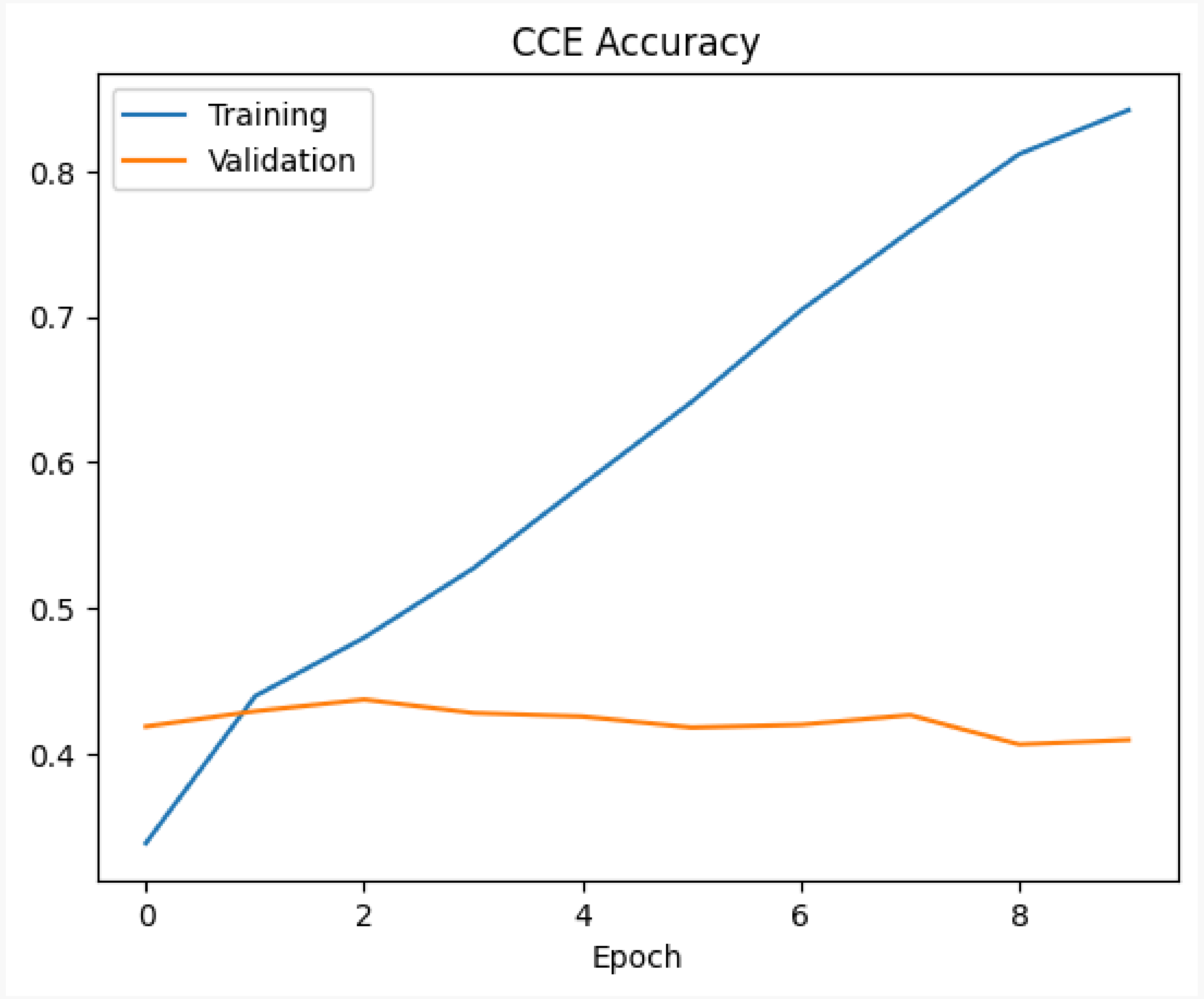
Lorem ipsum dolor sit amet consectetur adipiscing elit

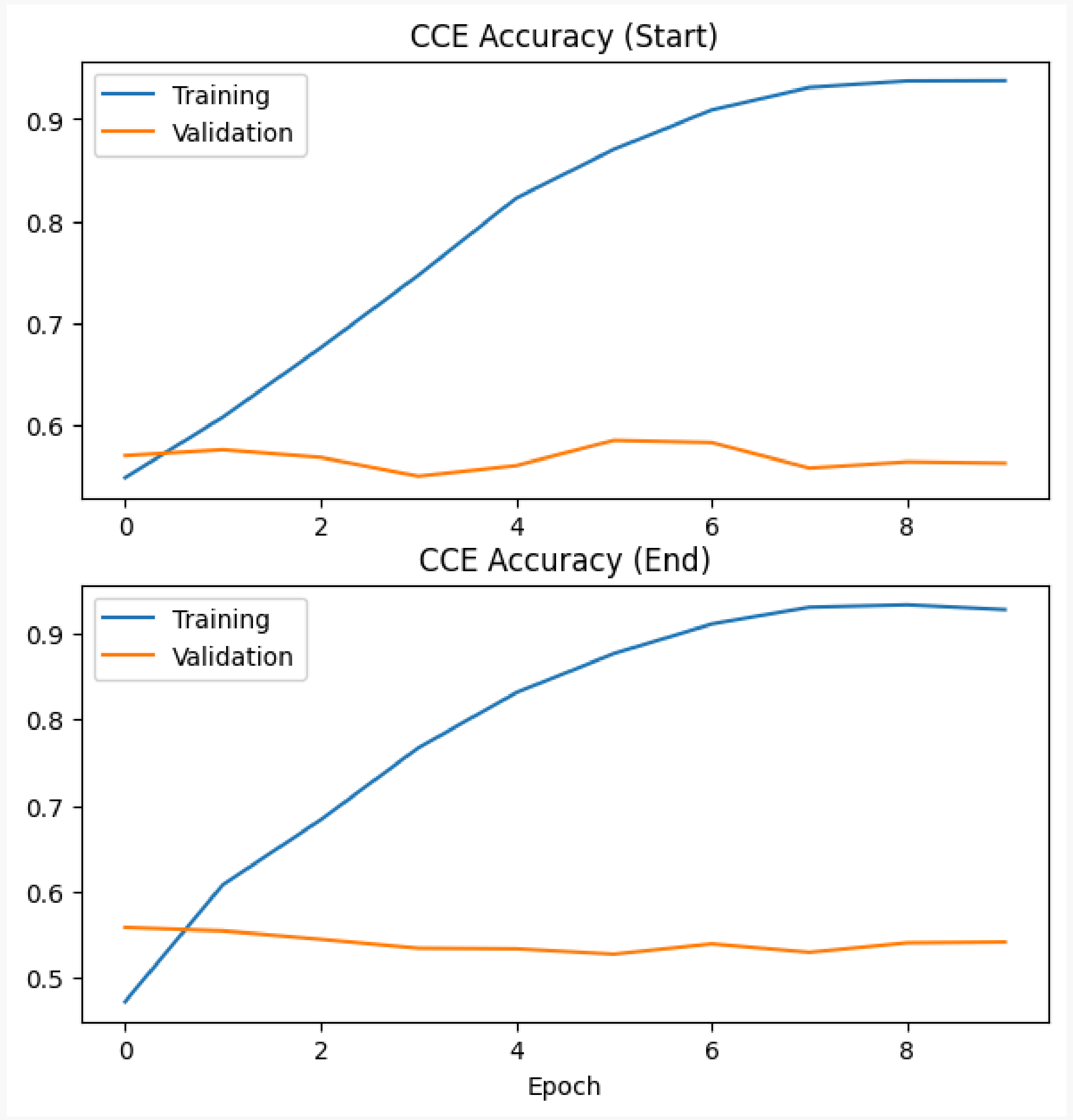
Start token: 2 dolor sit amet End token: 5





Encoding Outputs: One vs. Two Variables

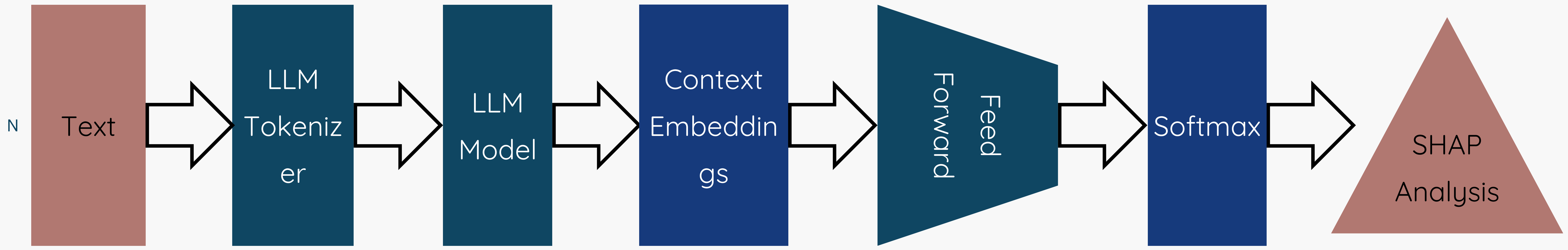




What Went Wrong?

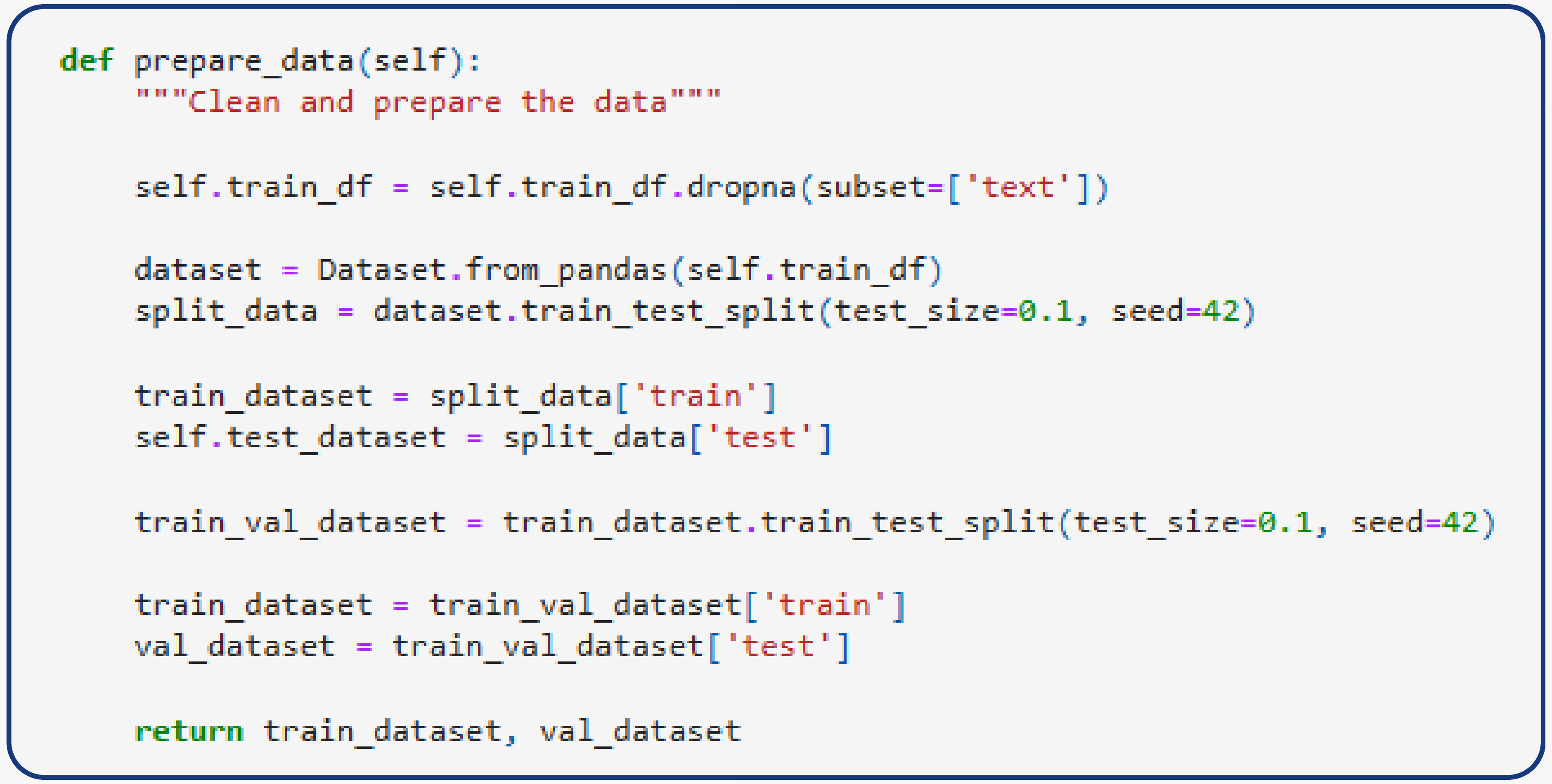
- One vs. two variable output
  - Using single variable (instead of two parallel softmax layers) led to worse accuracy, possibly due to overfitting. Eg. output encoding scheme described earlier favour values closer to zero
- Reasonable training accuracies, but low validation accuracies (less than 0.6)
  - Poor problem formulation (naive assumption as a classification problem)
  - Not enough layers (width and depth)

Final Solution : Using Multi-Model Training with LLMs and SHAP ANALYSIS



Data Preprocess:

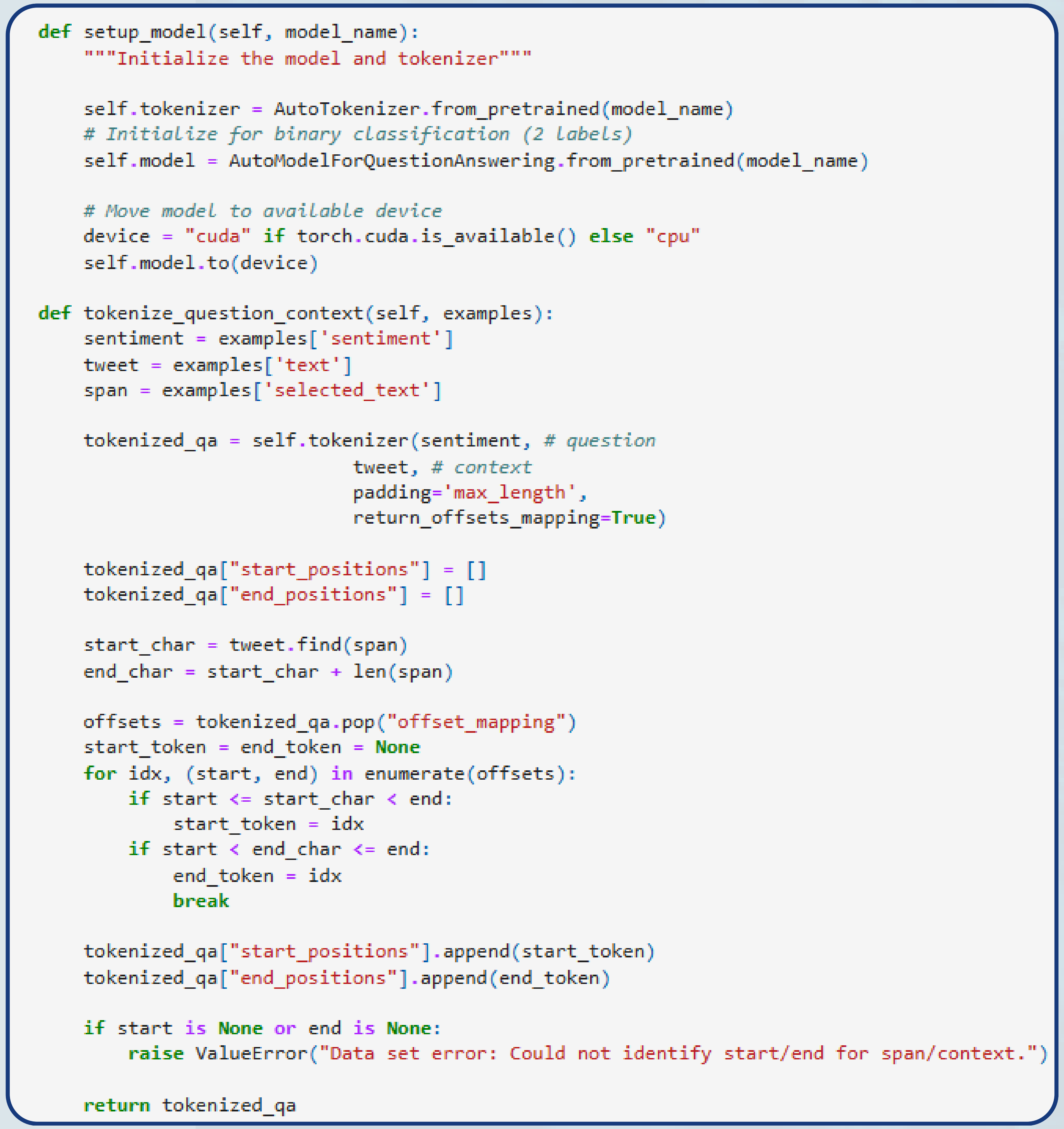
1. **Data Cleaning & Preparation:**
   - Removes rows with missing 'text' values and converts the DataFrame into a dataset format.
2. **Dataset Splitting:**
   - Splits the data into training (80%), validation (10%), and test (10%) sets for model evaluation.
3. **Reproducibility:**
   - Uses a fixed seed (42) to ensure consistent train-test splits across runs.



```python
def prepare_data(self):
    """Clean and prepare the data"""

    self.train_df = self.train_df.dropna(subset=['text'])

    dataset = Dataset.from_pandas(self.train_df)
    split_data = dataset.train_test_split(test_size=0.1, seed=42)

    train_dataset = split_data['train']
    self.test_dataset = split_data['test']

    train_val_dataset = train_dataset.train_test_split(test_size=0.1, seed=42)

    train_dataset = train_val_dataset['train']
    val_dataset = train_val_dataset['test']

    return train_dataset, val_dataset
```

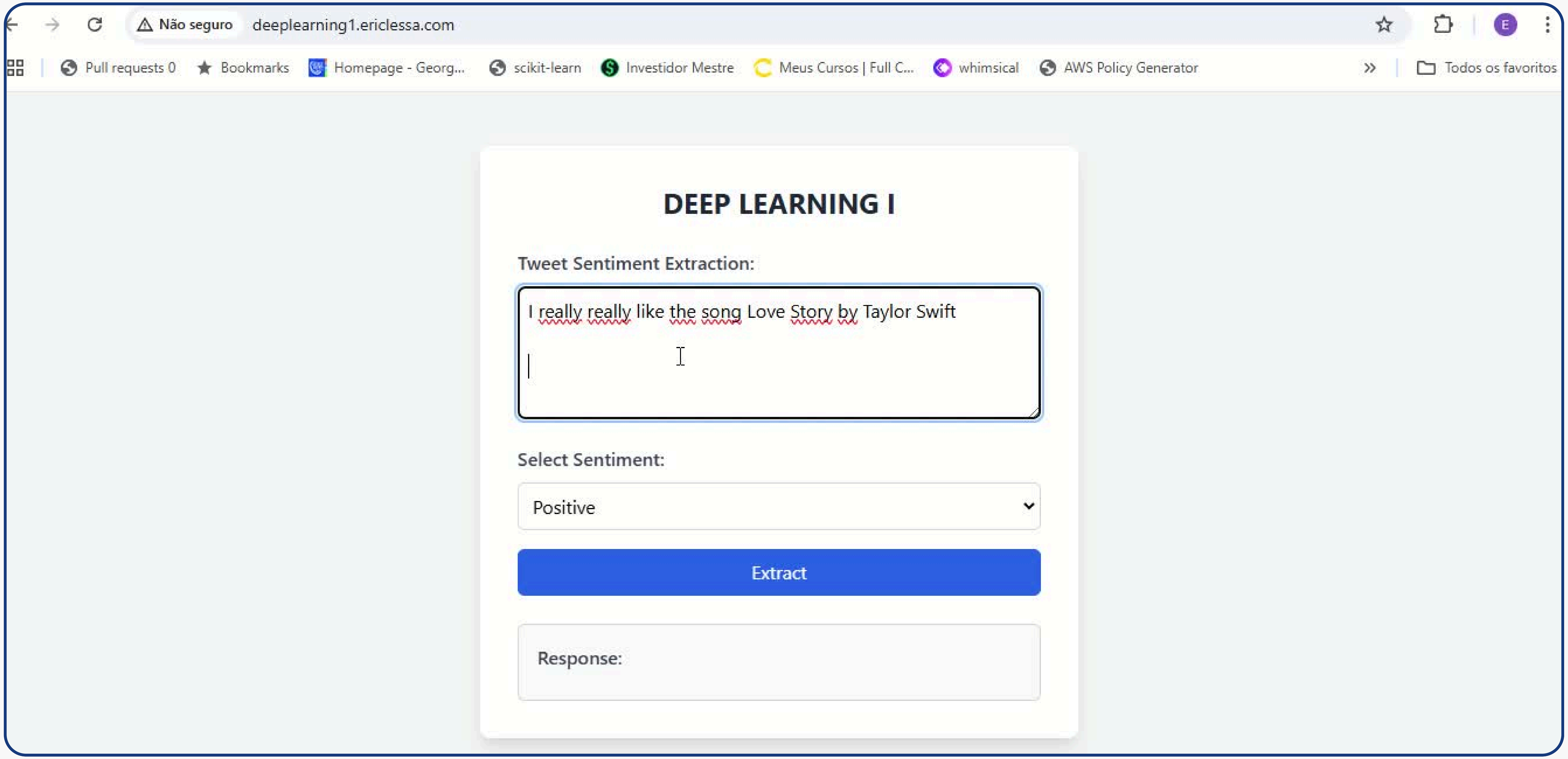
Model and tokenizer

1. **Model Setup:**
   - Loads a pre-trained question-answering model and tokenizer, assigning them to the available device (GPU/CPU).
2. **Tokenization & Span Mapping:**
   - Tokenizes sentiment (question) and tweet (context), mapping the selected text's start and end token positions.
3. **Error Handling:**
   - Ensures valid span identification, raising an error if start or end positions are not found.



```python
def setup_model(self, model_name):
    """Initialize the model and tokenizer"""

    self.tokenizer = AutoTokenizer.from_pretrained(model_name)
    # Initialize for binary classification (2 labels)
    self.model = AutoModelForQuestionAnswering.from_pretrained(model_name)

    # Move model to available device
    device = "cuda" if torch.cuda.is_available() else "cpu"
    self.model.to(device)

def tokenize_question_context(self, examples):
    sentiment = examples['sentiment']
    tweet = examples['text']
    span = examples['selected_text']

    tokenized_qa = self.tokenizer(sentiment, # question
                                  tweet, # context
                                  padding='max_length',
                                  return_offsets_mapping=True)

    tokenized_qa["start_positions"] = []
    tokenized_qa["end_positions"] = []

    start_char = tweet.find(span)
    end_char = start_char + len(span)

    offsets = tokenized_qa.pop("offset_mapping")
    start_token = end_token = None
    for idx, (start, end) in enumerate(offsets):
        if start <= start_char < end:
            start_token = idx
        if start < end_char <= end:
            end_token = idx
            break

    tokenized_qa["start_positions"].append(start_token)
    tokenized_qa["end_positions"].append(end_token)

    if start is None or end is None:
        raise ValueError("Data set error: Could not identify start/end for span/context.")

    return tokenized_qa
```

App - Demonstration



Não seguro deeplearning1.ericlessa.com

Pull requests 0 · Bookmarks · Homepage - Georg... · scikit-learn · Investidor Mestre · Meus Cursos | Full C... · whimsical · AWS Policy Generator · Todos os favoritos

DEEP LEARNING I

Tweet Sentiment Extraction:

I really really like the song Love Story by Taylor Swift

Select Sentiment:

Positive

Extract

Response:

Tokenization Example

“So many tests todayyy I don`t feel confident about any.”

Token IDs: tensor([[101, 4997, 102, 2061, 2116, 5852, 2651, 2100, 2100, 1045, 2123, 1036, 1056, 2514, 9657, 2055, 2151, 2100, 1012, 102]])

Tokens: ['[CLS]', 'negative', '[SEP]', 'so', 'many', 'tests', 'today', '##y', '##y', 'i', 'don', '`', 't', 'feel', 'confident', 'about', 'any', '##y', '.', '[SEP]']

question tokenized: 'negative'

context tokenized: 'so', 'many', 'tests', 'today', '##y', '##y', 'i', 'don', '`', 't', 'feel', 'confident', 'about', 'any', '##y', '.', '

special Tokens: [CLS] [SEP] -> Separate question from the context, start and end.

Tokenization Example

tweet: “So many tests todayyy I don`t feel confident about any.”
prediction: “I don`t feel confident”

Token IDs: tensor([[101, 4997, 102, 2061, 2116, 5852, 2651, 2100, 2100, 1045, 2123, 1036, 1056, 2514, 9657, 2055, 2151, 2100, 1012, 102]])

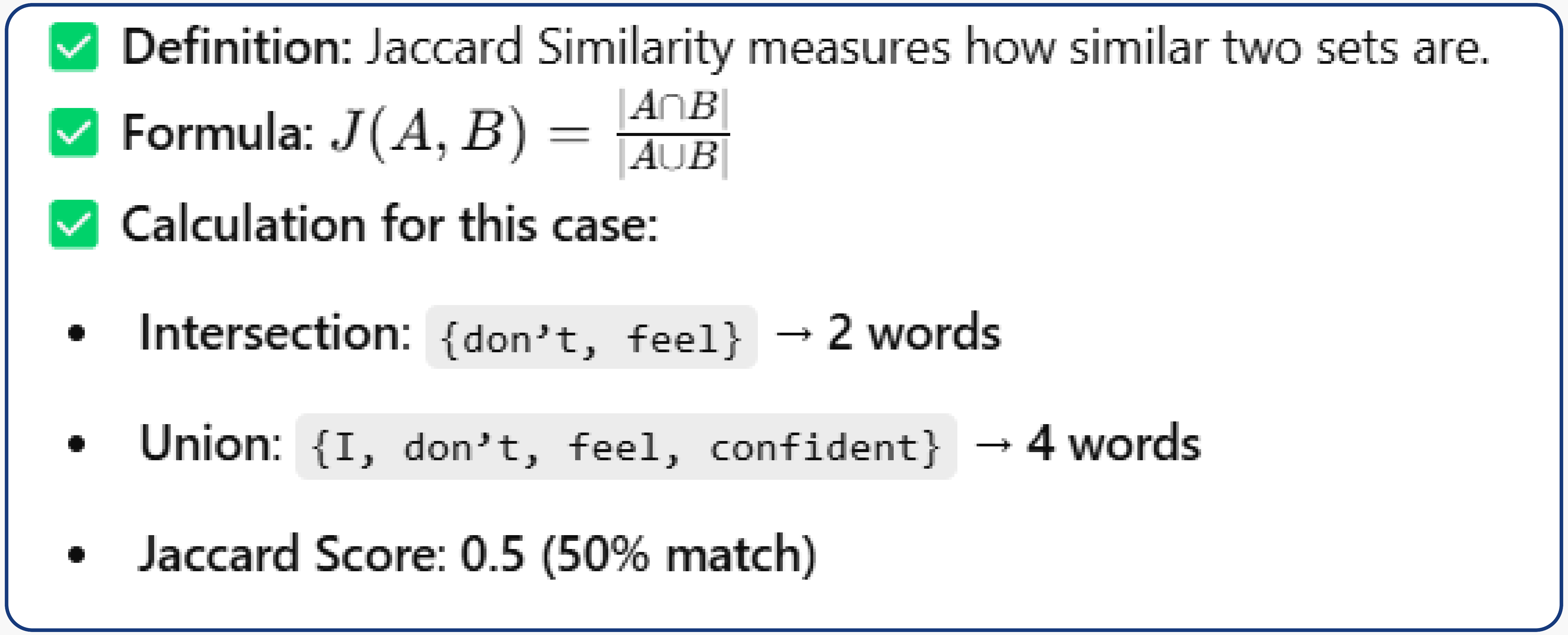
Start token: 'i'
Start token id: 1045

End token: ‘confident’
End token id: 9657

span tokens: [1045, 2123, 1036, 1056, 2514, 9657]
result: 'i', 'don', '`', 't', 'feel', 'confident'

Evaluation - Jaccard Similarity

tweet: So many tests todayyy I don`t feel confident about anyy.
target/span expected: I don`t feel confident
prediction: don`t feel



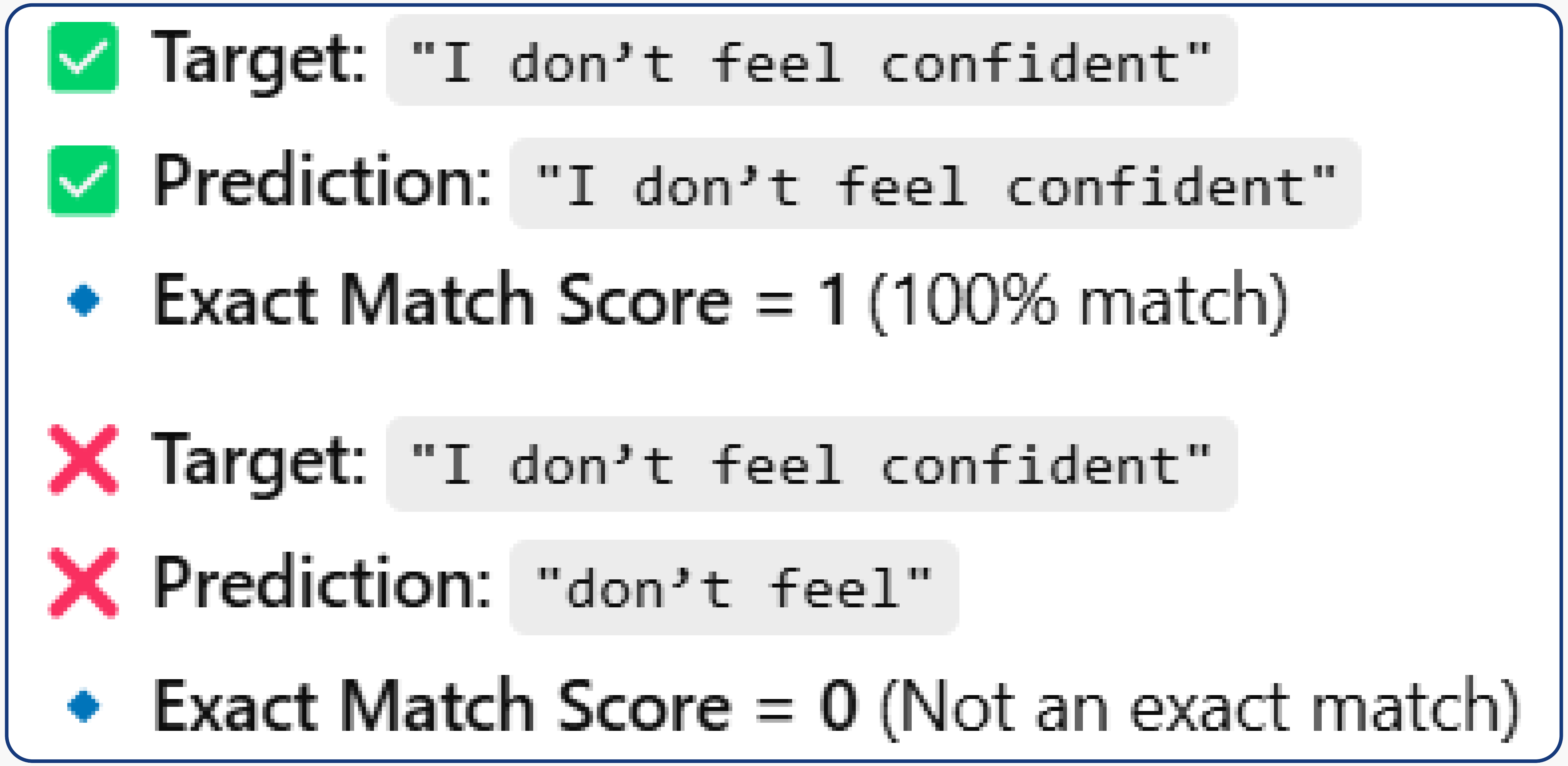
✅ **Definition:** Jaccard Similarity measures how similar two sets are.

✅ **Formula:** $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$

✅ **Calculation for this case:**

- **Intersection:** `{don’t, feel}` → **2 words**
- **Union:** `{I, don’t, feel, confident}` → **4 words**
- **Jaccard Score: 0.5 (50% match)**

Evaluation - exact match



✅ Target: `"I don’t feel confident"`

✅ Prediction: `"I don’t feel confident"`

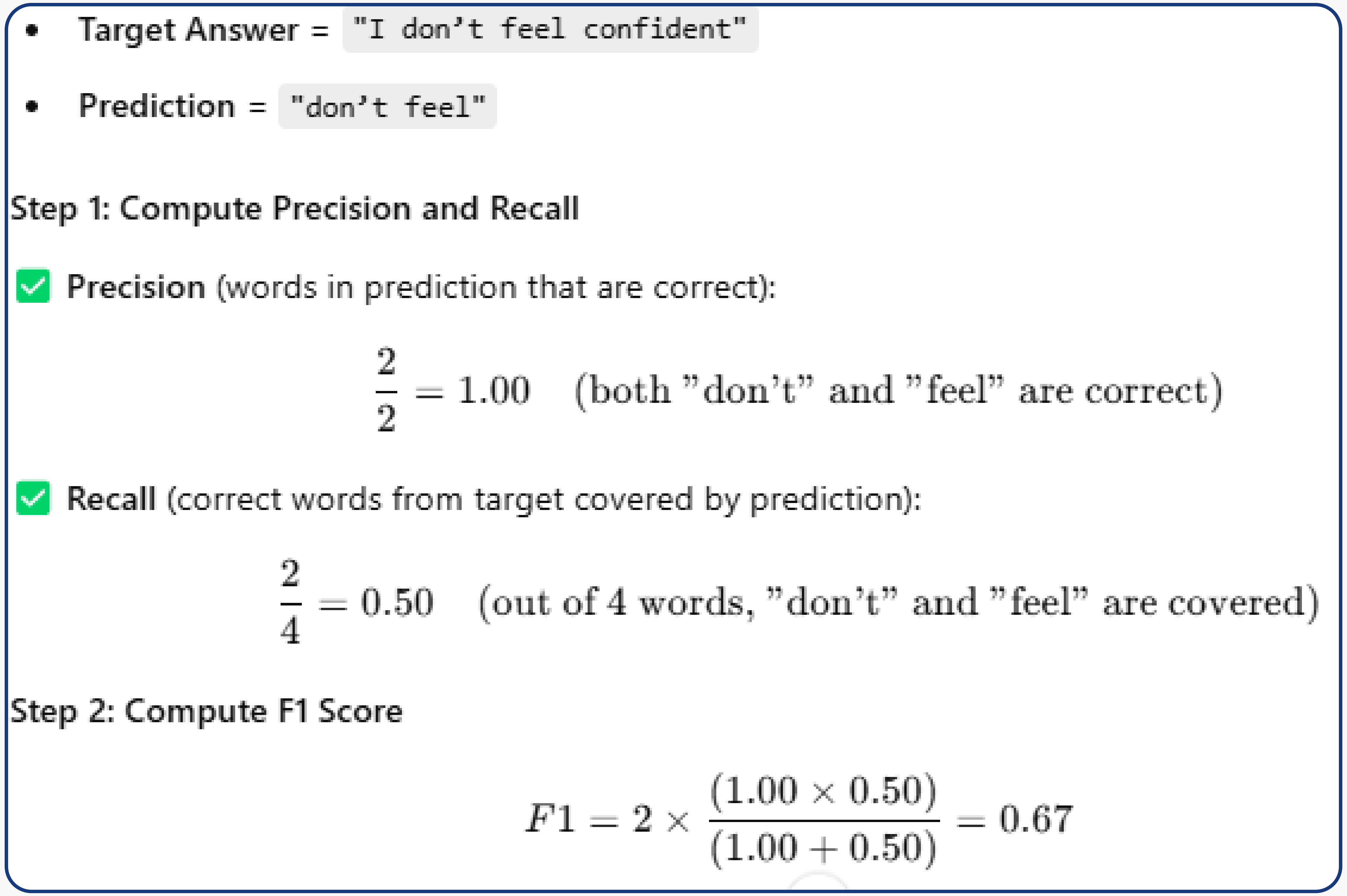
🔹 Exact Match Score = 1 (100% match)

❌ Target: `"I don’t feel confident"`

❌ Prediction: `"don’t feel"`

🔹 Exact Match Score = 0 (Not an exact match)

Evaluation - f1 score



- Target Answer = `"I don’t feel confident"`
- Prediction = `"don’t feel"`

Step 1: Compute Precision and Recall

✅ **Precision** (words in prediction that are correct):

$$\frac{2}{2} = 1.00 \quad \text{(both "don't" and "feel" are correct)}$$

✅ **Recall** (correct words from target covered by prediction):

$$\frac{2}{4} = 0.50 \quad \text{(out of 4 words, "don't" and "feel" are covered)}$$

Step 2: Compute F1 Score

$$F1 = 2 \times \frac{(1.00 \times 0.50)}{(1.00 + 0.50)} = 0.67$$

Evaluation

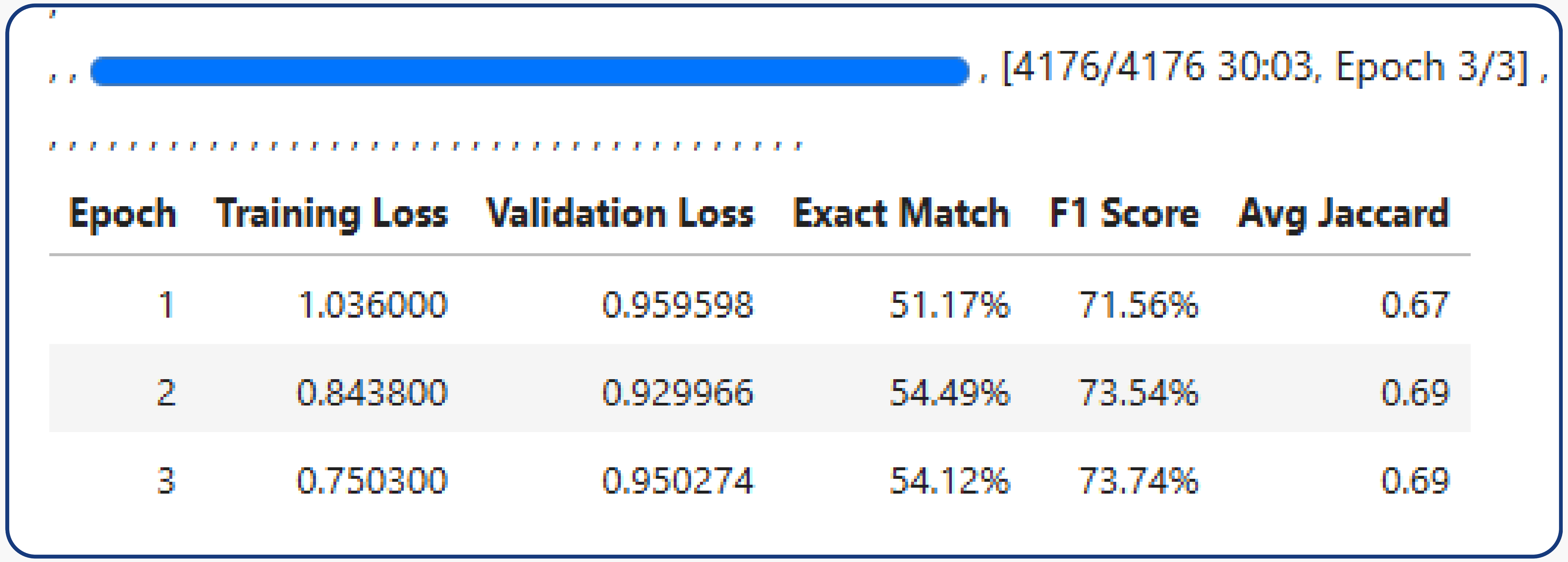
BASELINE -> BERT execution without fine tuning:

exact_match: 0.38%

f1_score: 9.44%

avg_jaccard: 7%

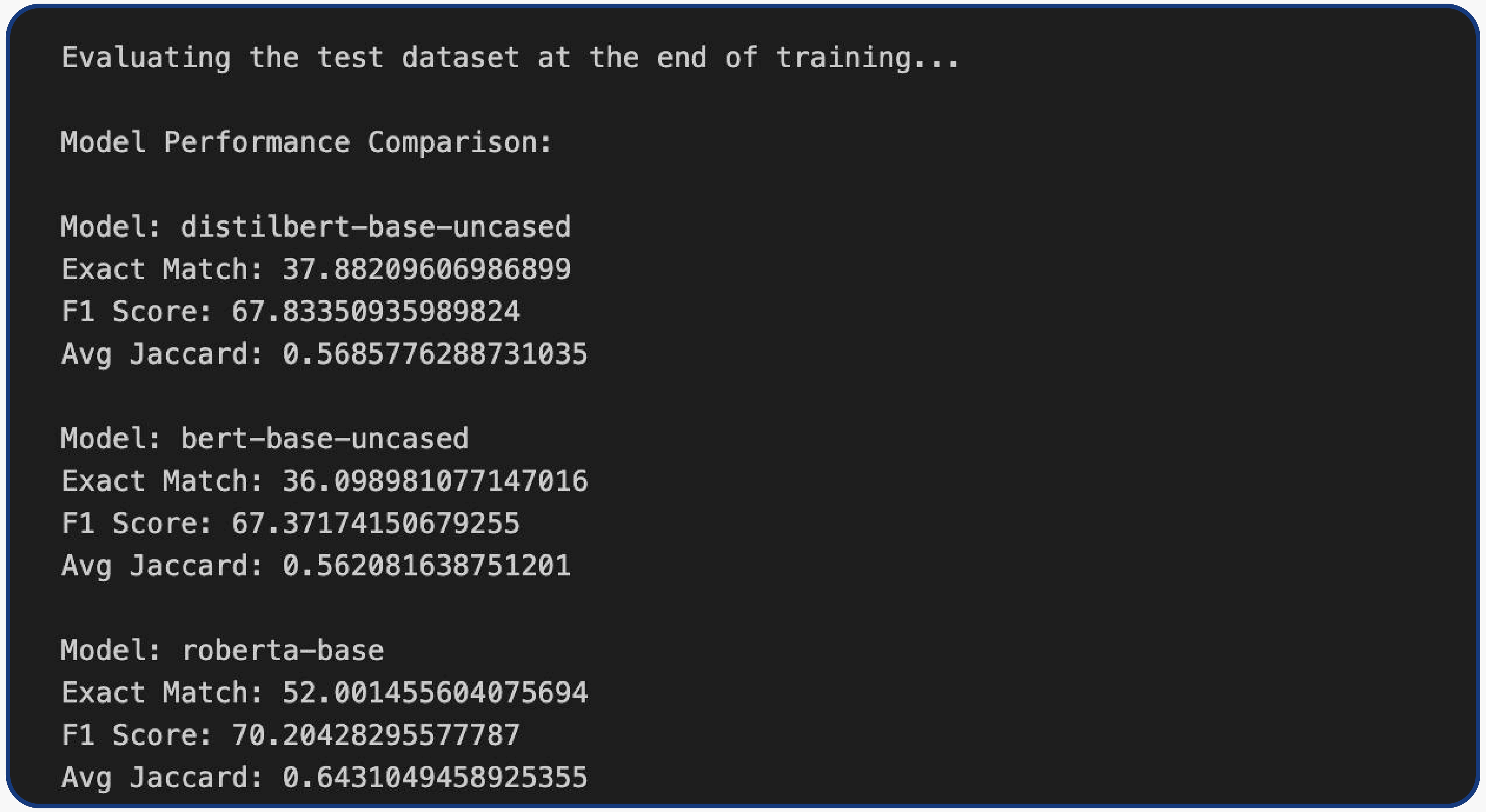
After fine-tuning with distilbert-base-uncased



```
[4176/4176 30:03, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Exact Match	F1 Score	Avg Jaccard
1	1.036000	0.959598	51.17%	71.56%	0.67
2	0.843800	0.929966	54.49%	73.54%	0.69
3	0.750300	0.950274	54.12%	73.74%	0.69

Metrics:

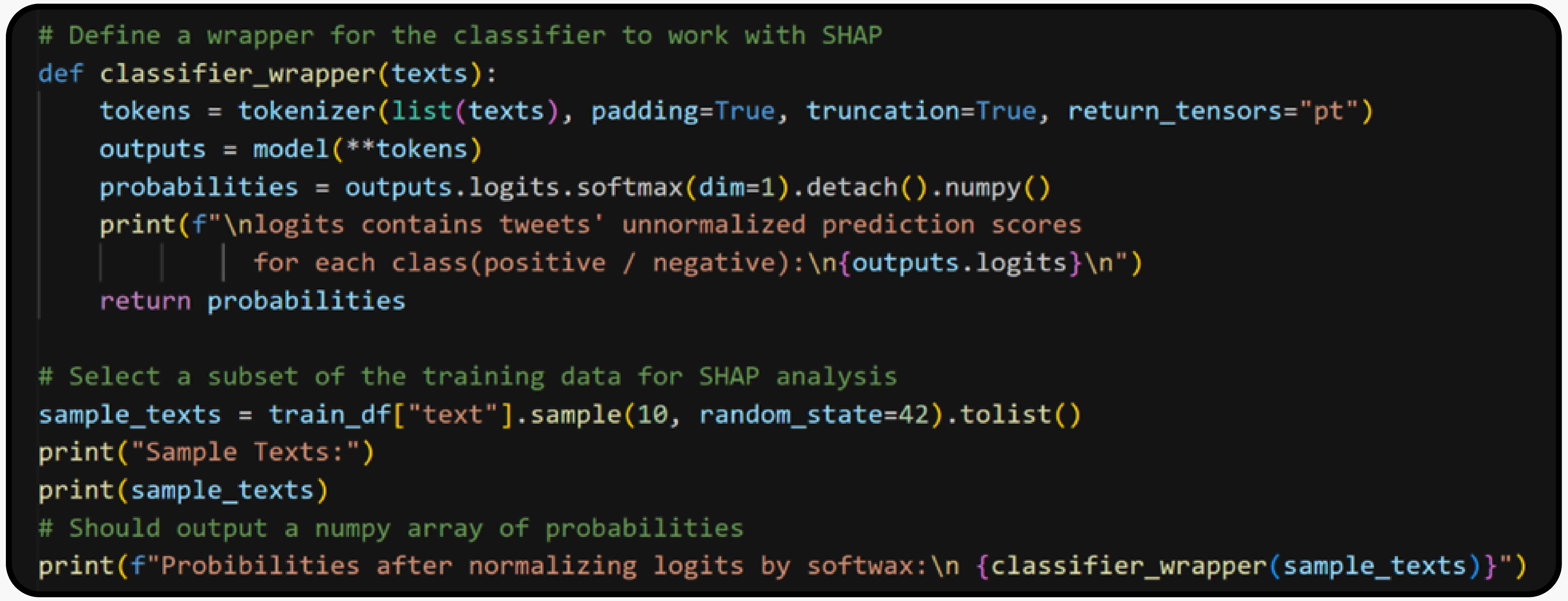


```
Evaluating the test dataset at the end of training...

Model Performance Comparison:

Model: distilbert-base-uncased
Exact Match: 37.88209606986899
F1 Score: 67.83350935989824
Avg Jaccard: 0.5685776288731035

Model: bert-base-uncased
Exact Match: 36.098981077147016
F1 Score: 67.37174150679255
Avg Jaccard: 0.562081638751201

Model: roberta-base
Exact Match: 52.001455604075694
F1 Score: 70.20428295577787
Avg Jaccard: 0.6431049458925355
```

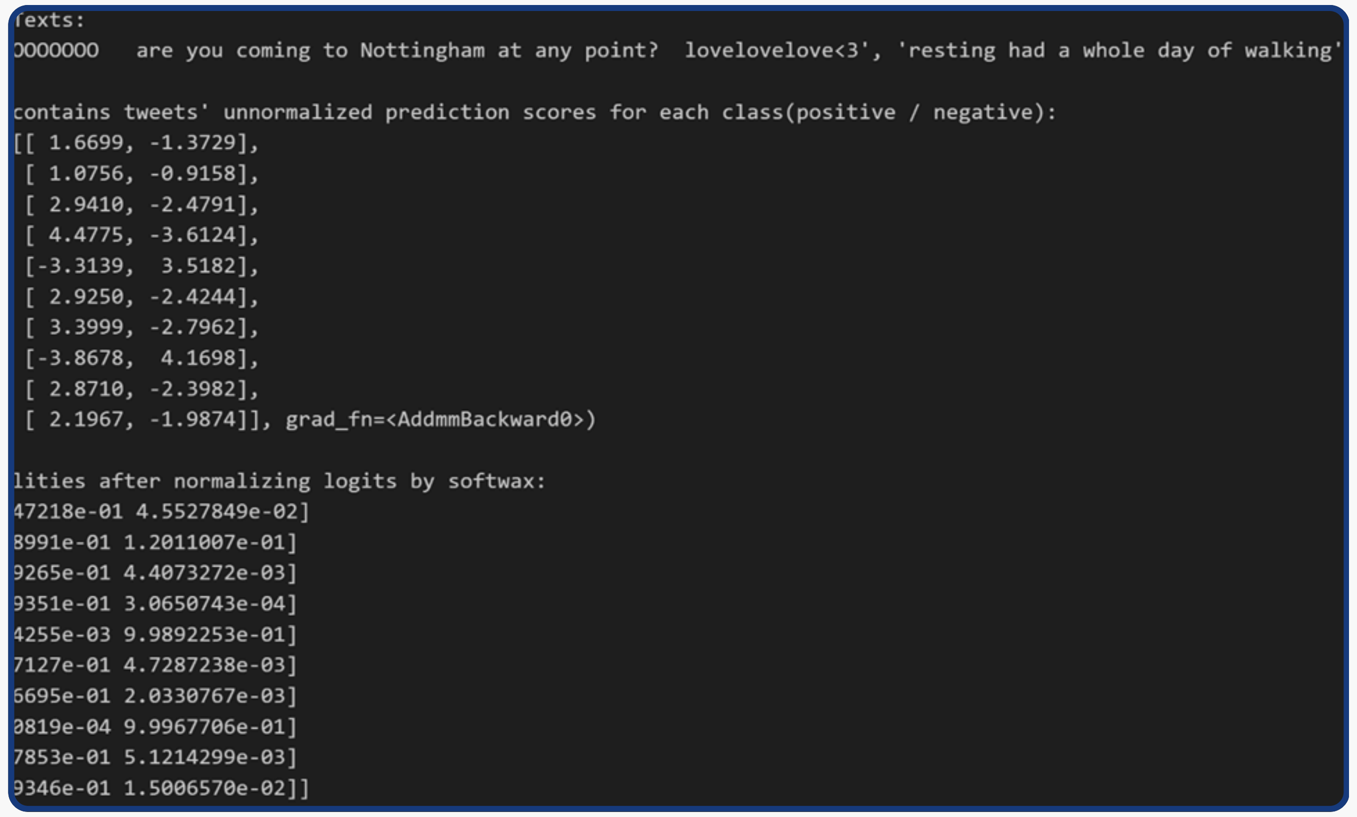
- Model Comparison: Among the tested models, RoBERTa-base outperforms DistilBERT and BERT in all metrics, achieving the highest Exact Match (52.00%), F1 Score (70.20%), and Average Jaccard (0.64), indicating its superior ability to extract accurate sentiment spans.

LLMs:

1. ·Pre-trained models such as BERT, LLama, GPTT,etc., and BERT-based models (like distilbert-base-uncased-finetuned-sst-2-english) are general-purpose transformer models which do not include an activation function at their output layer by default.
2. ·Instead, they output logits, which are raw, unnormalized scores for each class. The activation function, if needed, is applied externally depending on the specific task and requirements such as Multi-class classification (e.g., softmax), Binary classification (e.g., sigmoid), Regression tasks (no activation function).
3. ·We use detach() since we don't want to compute gradients for the tensor. Since probabilities are used only for prediction (not training), there's no need to track gradients, so detach() avoids unnecessary computational overhead.



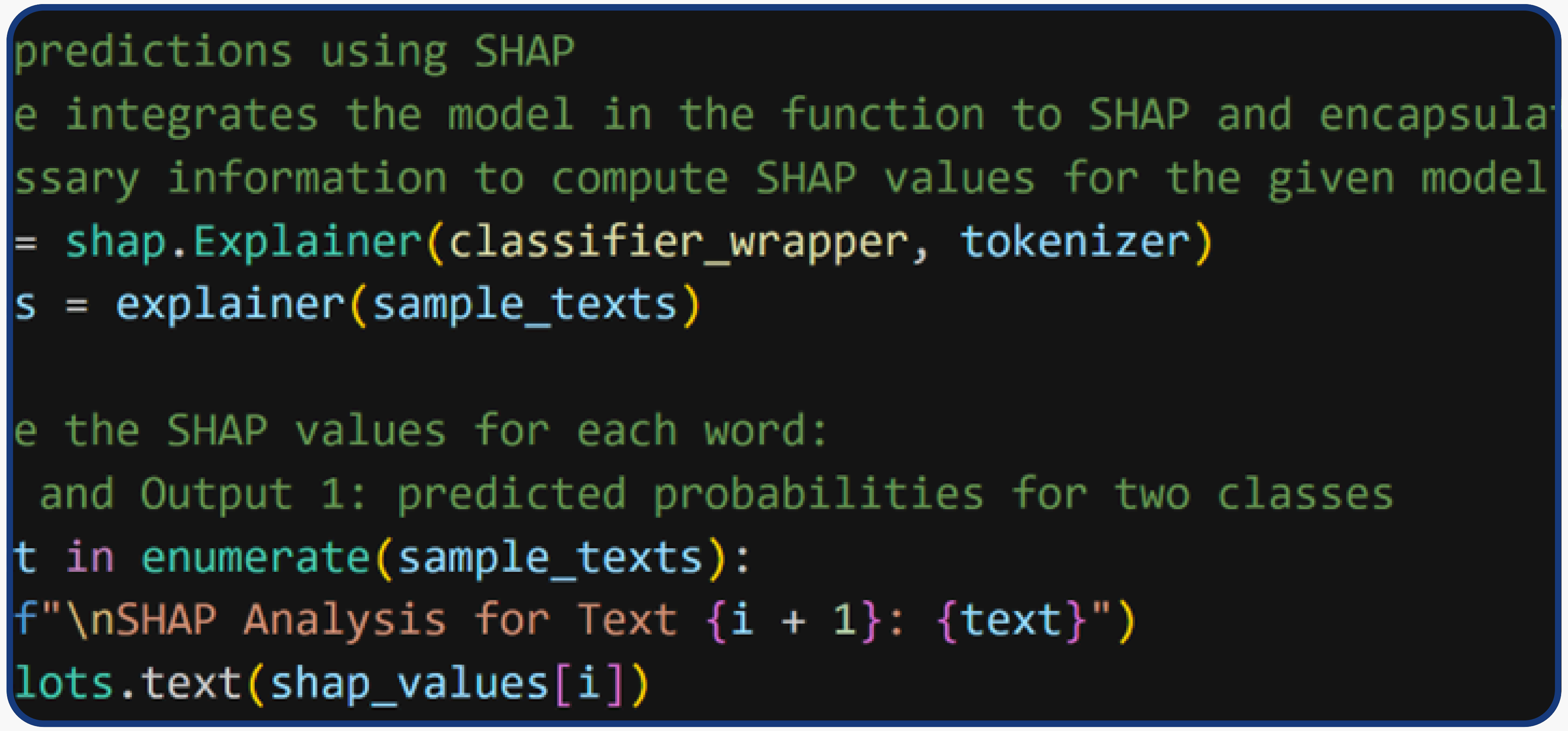
```python
# Define a wrapper for the classifier to work with SHAP
def classifier_wrapper(texts):
    tokens = tokenizer(list(texts), padding=True, truncation=True, return_tensors="pt")
    outputs = model(**tokens)
    probabilities = outputs.logits.softmax(dim=1).detach().numpy()
    print(f"\nlogits contains tweets' unnormalized prediction scores
            for each class(positive / negative):\n{outputs.logits}\n")
    return probabilities

# Select a subset of the training data for SHAP analysis
sample_texts = train_df["text"].sample(10, random_state=42).tolist()
print("Sample Texts:")
print(sample_texts)
# Should output a numpy array of probabilities
print(f"Probibilities after normalizing logits by softwax:\n {classifier_wrapper(sample_texts)}")
```



```
exts:
0000000   are you coming to Nottingham at any point?  lovelovelove<3', 'resting had a whole day of walking'

contains tweets' unnormalized prediction scores for each class(positive / negative):
[[ 1.6699, -1.3729],
 [ 1.0756, -0.9158],
 [ 2.9410, -2.4791],
 [ 4.4775, -3.6124],
 [-3.3139,  3.5182],
 [ 2.9250, -2.4244],
 [ 3.3999, -2.7962],
 [-3.8678,  4.1698],
 [ 2.8710, -2.3982],
 [ 2.1967, -1.9874]], grad_fn=<AddmmBackward0>)

lities after normalizing logits by softwax:
47218e-01 4.5527849e-02]
8991e-01 1.2011007e-01]
9265e-01 4.4073272e-03]
9351e-01 3.0650743e-04]
4255e-03 9.9892253e-01]
7127e-01 4.7287238e-03]
6695e-01 2.0330767e-03]
0819e-04 9.9967706e-01]
7853e-01 5.1214299e-03]
9346e-01 1.5006570e-02]]
```

SHAP ANALYSIS:

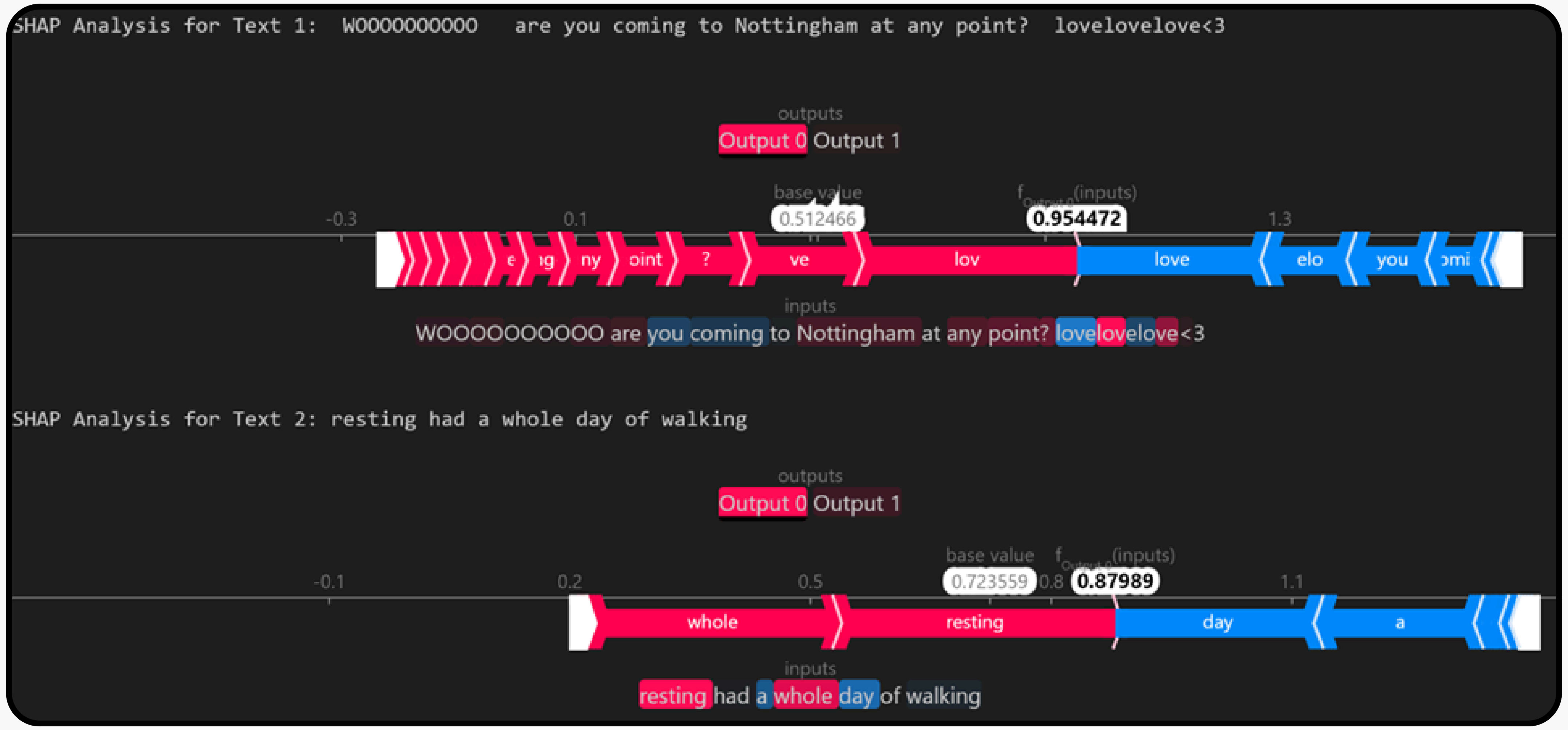
1. ·When classifier_wrapper is run by SHAP, there are six or less probabilities for the number of the sample_texts. It is because the tokenizer applies padding and truncation to the input. If multiple texts are identical, the tokenizer may return the same tokens for duplicates. If sample_texts contains duplicate or very similar tweets, their logits will be identical, and we might get fewer output tensors.



```
predictions using SHAP
e integrates the model in the function to SHAP and encapsula
ssary information to compute SHAP values for the given model
= shap.Explainer(classifier_wrapper, tokenizer)
s = explainer(sample_texts)

e the SHAP values for each word:
 and Output 1: predicted probabilities for two classes
t in enumerate(sample_texts):
f"\nSHAP Analysis for Text {i + 1}: {text}")
lots.text(shap_values[i])
```

SHAP ANALYSIS:

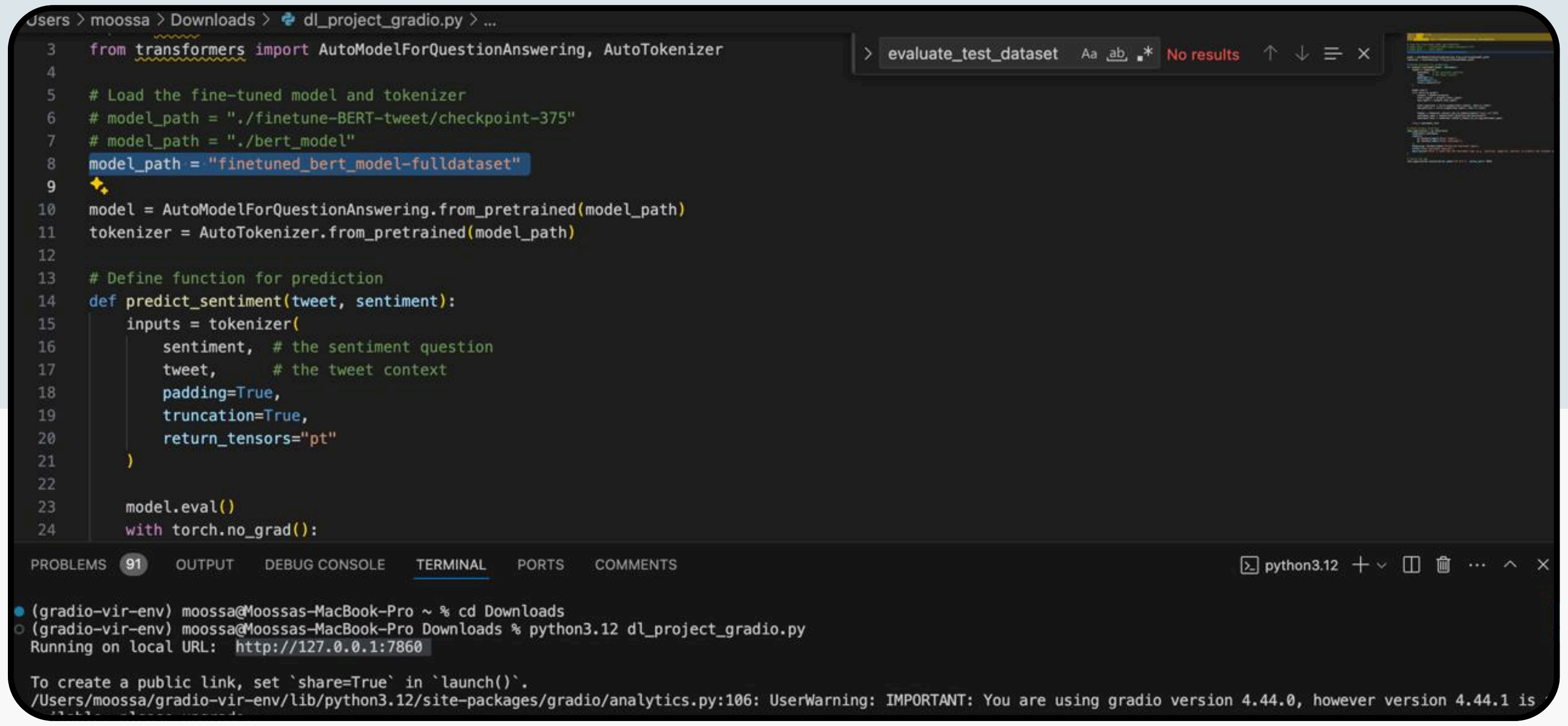
1. ·SHAP Analysis: SHAP is based on Shapley Values which is a concept from Game Theory. Shapley Values are assigned to each feature (player) based on contribution of each feature (in this case, words or tokens) to the model's prediction or different feature combinations.
2. ·The explainer Is the core class in the SHAP library. It encapsulates the model and the method for calculating SHAP Values.



```
SHAP Analysis for Text 1:  WOOOOOOOOOO   are you coming to Nottingham at any point?  lovelovelove<3
```

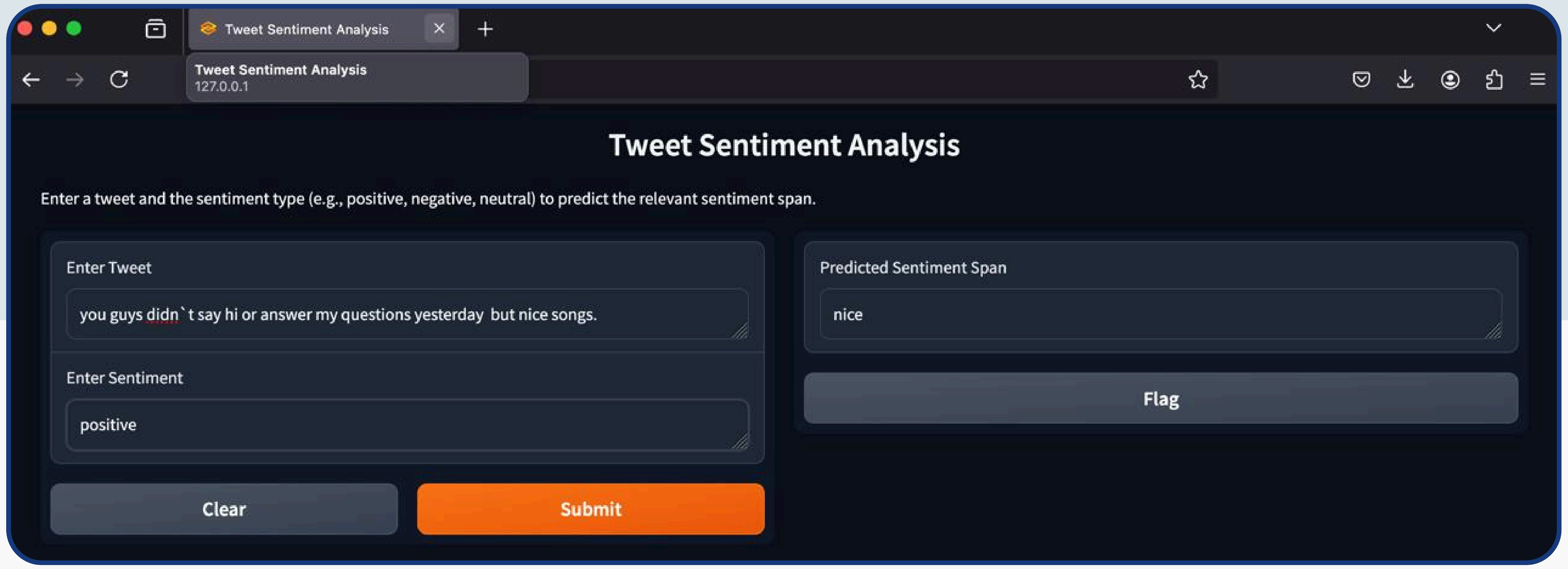
```
SHAP Analysis for Text 2: resting had a whole day of walking
```

Front-End Interface:



```
Users > moossa > Downloads > dl_project_gradio.py > ...
from transformers import AutoModelForQuestionAnswering, AutoTokenizer

# Load the fine-tuned model and tokenizer
# model_path = "./finetune-BERT-tweet/checkpoint-375"
# model_path = "./bert_model"
model_path = "finetuned_bert_model-fulldataset"

model = AutoModelForQuestionAnswering.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Define function for prediction
def predict_sentiment(tweet, sentiment):
    inputs = tokenizer(
        sentiment,  # the sentiment question
        tweet,      # the tweet context
        padding=True,
        truncation=True,
        return_tensors="pt"
    )

    model.eval()
    with torch.no_grad():
```

```
(gradio-vir-env) moossa@Moossas-MacBook-Pro ~ % cd Downloads
(gradio-vir-env) moossa@Moossas-MacBook-Pro Downloads % python3.12 dl_project_gradio.py
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
/Users/moossa/gradio-vir-env/lib/python3.12/site-packages/gradio/analytics.py:106: UserWarning: IMPORTANT: You are using gradio version 4.44.0, however version 4.44.1 is
```



Tweet Sentiment Analysis

Enter a tweet and the sentiment type (e.g., positive, negative, neutral) to predict the relevant sentiment span.

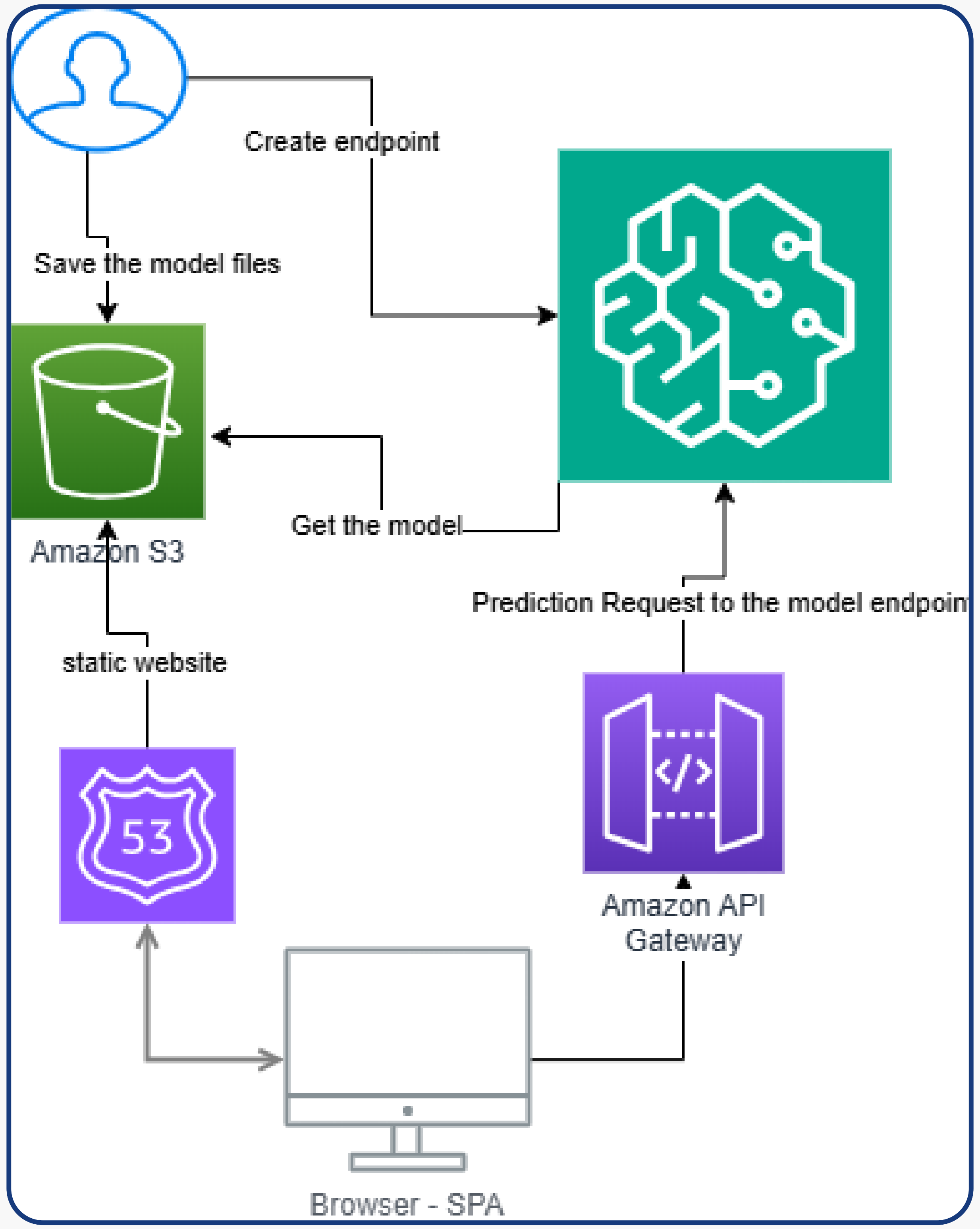
Enter Tweet: you guys didn`t say hi or answer my questions yesterday but nice songs.

Enter Sentiment: positive

Predicted Sentiment Span: nice

1. **Sentiment Span Prediction:**
    - **Utilizes a fine-tuned BERT-based model to extract sentiment spans from tweets by identifying start and end tokens using logits.**
2. **Interactive Gradio Interface:**
    - **Provides a user-friendly web interface for real-time sentiment analysis, where users input a tweet and sentiment to receive the predicted sentiment span.**

AWS Deployment SPA-Based App:



1. **AWS Sage Maker AI**
    - Managed environment for model training and deployment
    - Containerized environment for efficient model deployment and scaling.
    - Endpoint provided for model inference and predictions.
2. **S3**
    - Object storage for model files and parameters.
3. **API Gateway**
    - Exposes a secure endpoint to access the service over the internet.
4. **SPA (Single Page Application)**
    - Web architecture for seamless user interaction.
5. **S3 Static Website**
    - Hosting static web content via S3 for the frontend interface.

Conclusion

- The code successfully fine-tunes the Roberta model for sentiment span extraction, achieving improved performance metrics (e.g., F1 score of 70.20% and Avg Jaccard of 0.64 on the test dataset).
- It integrates SHAP analysis to interpret model predictions, providing valuable insights into token-level contributions to sentiment classification.
- The model's performance improves significantly over epochs, indicating effective learning, with a clear increase in F1 score and Jaccard similarity.

Future Scope:

- Performance Enhancement:
    - Train with a larger dataset or explore ensemble methods for better generalization.
- Explainability:
    - Extend SHAP analysis to include batch visualizations and explore other interpretability techniques like LIME.
- Improved Baseline:
    - Compare fine-tuning results with advanced pre-trained models or other architectures.
- Deployment:
    - Package the model for real-world applications, such as customer sentiment analysis tools.

Thank you