# Code Explaination

## Documentation for Semantic Grouping of Clinical Studies

This document provides a detailed explanation of the code used for semantic grouping and retrieval of clinical studies. The code is designed to help researchers find similar historical clinical trials based on specific features such as study title, primary and secondary outcome measures, and eligibility criteria.

### Environment Details

- **GPU**: RTX 3090 (24GB VRAM)
- **RAM**: 125GB
- **Template**: pytorch:2.1.0-py3.10-cuda11.8.0-devel-ubuntu22.04

### Code Overview

The code is structured into several classes and functions, each serving a specific purpose. Below is a detailed breakdown of each major section.

#### 1. Installation and Imports

The code begins with the installation of necessary libraries and importing required modules.

```
!pip install --no-cache-dir transformers==4.37.2 torch==2.1.0 scikit-
learn==1.3.2 pandas numpy tqdm spacy textblob gdown
!python -m spacy download en_core_web_sm
```

```
import os
import nltk
nltk.download(['punkt', 'stopwords'], quiet=True)
import spacy
import pandas as pd
import numpy as np
from transformers import AutoTokenizer, AutoModel
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords
import torch
from tqdm.auto import tqdm
import re
import warnings
from datetime import datetime
import gc
```

```
import gdown
from textblob import TextBlob
warnings.filterwarnings('ignore')

torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
```

### 2. DataLoader Class

The `DataLoader` class is responsible for loading and preprocessing the clinical trial data.

```
class DataLoader:
    def __init__(self, usecase_url, eligibilities_url):
        self.usecase_url = usecase_url.replace('/file/d/', '/uc?
id=').replace('/view?usp=sharing', '')
        self.eligibilities_url = eligibilities_url.replace('/file/d/', '/uc?
id=').replace('/view?usp=sharing', '')

 def download_and_load(self):
        usecase_output = 'usecase_1.csv'
        eligibilities_output = 'eligibilities.txt'

        try:
            gdown.download(self.usecase_url, usecase_output, quiet=False)
            gdown.download(self.eligibilities_url, eligibilities_output,
quiet=False)

            chunks = pd.read_csv(usecase_output, chunksize=10000)
            df = pd.concat(chunks)
            eligibilities = pd.read_csv(eligibilities_output, delimiter='|')

            print("\nColumns in original DataFrame:")
            print(df.columns.tolist())

            df = df[df['Funder Type'] == 'INDUSTRY']
            df = df.merge(eligibilities[['nct_id', 'criteria']],
                          left_on='NCT Number',
                          right_on='nct_id',
                          how='left')

            start_date_col = next((col for col in df.columns if 'start' in
col.lower()), None)
            completion_date_col = next((col for col in df.columns if
'complet' in col.lower()), None)
```

```
            if start_date_col and completion_date_col:
                df['year'] = pd.to_datetime(df[start_date_col],
errors='coerce').dt.year
                df['duration_years'] =
pd.to_datetime(df[completion_date_col], errors='coerce').dt.year -
df['year']

            df = df.fillna('')

            os.remove(usecase_output)
            os.remove(eligibilities_output)

            return df

        except Exception as e:
            print(f"Error in data loading: {str(e)}")
            raise
```

### 3. TextProcessor Class

The `TextProcessor` class handles text cleaning and feature combination.

```
class TextProcessor:
    def __init__(self):
        self.stop_words = set(stopwords.words('english'))
        self.text_columns = {
            'Study Title': 3.0,
            'Primary Outcome Measures': 2.5,
            'Secondary Outcome Measures': 2.0,
            'criteria': 2.5,
            'Conditions': 2.5,
            'Interventions': 2.0,
            'Phase': 1.5,
            'Brief Summary': 1.8
        }
        self.nlp = spacy.load('en_core_web_sm')

 def clean_text(self, text):
        text = str(text).lower()
        text = re.sub(r'[^\w\s]', ' ', text)
        text = re.sub(r'\s+', ' ', text)
        words = text.split()
        text = ' '.join(word for word in words if word not in
self.stop_words)
```

```
        return text

 def combine_features(self, row):
        text_parts = []
        for col, weight in self.text_columns.items():
            if col in row.index and row[col]:
                cleaned_text = self.clean_text(row[col])
                text_parts.extend([cleaned_text] * int(weight * 5))
        return ' '.join(text_parts)

 def process_batch(self, df, batch_size=1000):
        processed_texts = []
        for i in tqdm(range(0, len(df), batch_size), desc="Processing
texts"):
            batch_df = df.iloc[i:i + batch_size]
            batch_texts = []
            for _, row in batch_df.iterrows():
                processed_text = self.combine_features(row)
                batch_texts.append(processed_text)
            processed_texts.extend(batch_texts)
            if i % (batch_size * 5) == 0:
                gc.collect()
        return processed_texts
```

#### 4. BertSimilarityEngine Class

The `BertSimilarityEngine` class generates embeddings and computes similarity scores.

```
class BertSimilarityEngine:
    def __init__(self):
        self.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        self.tokenizer = AutoTokenizer.from_pretrained('microsoft/BiomedNLP-
PubMedBERT-base-uncased-abstract-fulltext')
        self.model = AutoModel.from_pretrained('microsoft/BiomedNLP-
PubMedBERT-base-uncased-abstract-fulltext')
        self.model.to(self.device)
        self.model.eval()
        if self.device.type == 'cuda':
            self.model.half()
        print(f"Using device: {self.device}")

 def mean_pooling(self, model_output, attention_mask):
        token_embeddings = model_output[0]
        input_mask_expanded =
```

```
attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        return torch.sum(token_embeddings * input_mask_expanded, 1) /
torch.clamp(input_mask_expanded.sum(1), min=1e-9)

 @torch.no_grad()
    def encode_texts(self, texts, batch_size=32):
        embeddings = []
        for i in tqdm(range(0, len(texts), batch_size), desc="Encoding"):
            batch = texts[i:i + batch_size]
            encoded = self.tokenizer(batch, padding=True, truncation=True,
max_length=512, return_tensors='pt')
            encoded = {k: v.to(self.device) for k, v in encoded.items()}
            model_output = self.model(**encoded)
            batch_embeddings = self.mean_pooling(model_output,
encoded['attention_mask'])
            embeddings.append(batch_embeddings.cpu().numpy())
            if i % (batch_size * 2) == 0:
                torch.cuda.empty_cache()
        return np.vstack(embeddings)

 def compute_similarities(self, query_embedding, all_embeddings):
        return cosine_similarity(query_embedding.reshape(1, -1),
all_embeddings)[0]
```

### 5. TrialAnalyzer Class

The `TrialAnalyzer` class orchestrates the analysis and retrieval of similar trials.

```
class TrialAnalyzer:
    def __init__(self, usecase_url, eligibilities_url):
        self.data_loader = DataLoader(usecase_url, eligibilities_url)
        self.text_processor = TextProcessor()
        self.similarity_engine = BertSimilarityEngine()
        self.df = None
        self.embeddings = None

 def calculate_trial_metrics(self, trial):
        metrics = {}
        try:
            metrics['enrollment_size'] = trial['Enrollment'] if
pd.notna(trial['Enrollment']) else 0
            metrics['has_results'] = 1 if pd.notna(trial.get('Study
Results', '')) else 0
            metrics['completion_status'] = trial.get('Study Status', '')
```

```
            metrics['duration_years'] = trial['duration_years'] if
pd.notna(trial.get('duration_years', '')) else 0

            text_length = len(str(trial.get('Brief Summary', '')))
            metrics['complexity_score'] = text_length / 100

            sentiment = TextBlob(str(trial.get('Brief Summary',
'').sentiment.polarity
            metrics['sentiment_score'] = round(sentiment, 3)
        except Exception as e:
            print(f"Error calculating metrics: {str(e)}")
        return metrics

 def prepare_data(self):
        start_time = datetime.now()
        print("Loading and preprocessing data...")
        self.df = self.data_loader.download_and_load()
        print(f"Total trials: {len(self.df)}")

        print("Processing text features...")
        processed_texts = self.text_processor.process_batch(self.df)

        print("Computing embeddings...")
        self.embeddings =
self.similarity_engine.encode_texts(processed_texts)
        del processed_texts
        gc.collect()

        end_time = datetime.now()
        print(f"Data preparation complete in {(end_time -
start_time).total_seconds():.2f} seconds")

 def analyze_trial_group(self, trials_df):
        analysis = {}
        try:
            analysis = {
                'total_trials': len(trials_df),
                'unique_conditions':
len(set(trials_df['Conditions'].str.split(',').sum())),
                'unique_interventions':
len(set(trials_df['Interventions'].str.split(',').sum())),
                'phase_distribution':
trials_df['Phase'].value_counts().to_dict(),
```

```
                'avg_enrollment': trials_df['Enrollment'].mean(),
                'avg_duration': trials_df['duration_years'].mean(),
                'completion_rate': (trials_df['Study Status'].str.lower() ==
'completed').mean(),
                'has_results_rate': trials_df['Study
Results'].notna().mean()
            }
        except Exception as e:
            print(f"Error in group analysis: {str(e)}")
        return analysis

 def find_similar_trials(self, nct_id, n_similar=10):
        try:
            query_idx = self.df[self.df['NCT Number'] == nct_id].index[0]
            query_trial = self.df.iloc[query_idx]
        except IndexError:
            print(f"NCT ID {nct_id} not found")
            return None

        similarities = self.similarity_engine.compute_similarities(
            self.embeddings[query_idx],
            self.embeddings
        )

        similar_indices = np.argsort(similarities)[::-1][1:n_similar+1]
        similar_trials = self.df.iloc[similar_indices].copy()
        similar_trials['similarity_score'] = similarities[similar_indices]
        similar_trials['rank'] = range(1, n_similar + 1)

        query_metrics = self.calculate_trial_metrics(query_trial)
        group_analysis = self.analyze_trial_group(similar_trials)

        for idx, row in similar_trials.iterrows():
            metrics = self.calculate_trial_metrics(row)
            for key, value in metrics.items():
                similar_trials.at[idx, key] = value

        similar_trials['common_conditions'] = similar_trials.apply(
            lambda x: len(set(str(x['Conditions']).split(',')) &
                    set(str(query_trial['Conditions']).split(','))),
axis=1)

        similar_trials['common_interventions'] = similar_trials.apply(
```

```
            lambda x: len(set(str(x['Interventions']).split(',')) &
                          set(str(query_trial['Interventions']).split(','))),
axis=1)

        return similar_trials, query_metrics, group_analysis
```

### 6. Main Function

The `main` function orchestrates the entire process.

```
def main():
    torch.backends.cudnn.benchmark = True
    torch.backends.cudnn.enabled = True

 if torch.cuda.is_available():
        torch.cuda.empty_cache()
        print_gpu_util()

    start_time = datetime.now()
    print(f"Analysis started at {start_time.strftime('%H:%M:%S')}")

    usecase_url =
"https://drive.google.com/file/d/1CLvAQeuxJCjE6CoiopXToHVDsN7CgQkg/view?
usp=sharing"
    eligibilities_url = "https://drive.google.com/file/d/1-
bkkhqXlb5eDd3hmsDEIfrPHXbMtVHqj/view?usp=sharing"

    analyzer = TrialAnalyzer(usecase_url, eligibilities_url)
    analyzer.prepare_data()

    test_ncts = ['NCT00385736', 'NCT00386607', 'NCT03518073']

    for nct in test_ncts:
        print(f"\nAnalyzing {nct}:")
        result = analyzer.find_similar_trials(nct)

        if result is not None:
            similar_trials, query_metrics, group_analysis = result

            print(f"\nQuery Trial Analysis for {nct}:")
            print("Metrics:", query_metrics)
            print("\nSimilar Trials Group Analysis:")
            for key, value in group_analysis.items():
                print(f"{key}: {value}")
```

```
            output_columns = [
                'rank', 'NCT Number', 'Study Title', 'Phase', 'Conditions',
                'Interventions', 'Primary Outcome Measures', 'Secondary
Outcome Measures',
                'Study Status', 'duration_years', 'enrollment_size',
'has_results',
                'completion_status', 'complexity_score', 'sentiment_score',
                'common_conditions', 'common_interventions',
'similarity_score'
            ]

            available_columns = [col for col in output_columns if col in
similar_trials.columns]

            print("\nTop 10 similar trials:")
            pd.set_option('display.max_columns', None)
            pd.set_option('display.max_colwidth', None)
            print(similar_trials[available_columns].to_string())

            detail_file = f'trial_details_{nct}.csv'
            similar_trials.to_csv(detail_file, index=False)

            summary_file = f'trial_analysis_{nct}.csv'
            summary_df = pd.DataFrame({
                'metric': list(query_metrics.keys()) +
list(group_analysis.keys()),
                'value': list(query_metrics.values()) +
list(group_analysis.values())
            })
            summary_df.to_csv(summary_file, index=False)

            print(f"\nDetailed results saved to {detail_file}")
            print(f"Analysis summary saved to {summary_file}")

            if torch.cuda.is_available():
                print("\nGPU Memory Status:")
                print_gpu_util()
                torch.cuda.empty_cache()

 end_time = datetime.now()
    total_time = (end_time - start_time).total_seconds()
    print(f"\nTotal analysis completed in {total_time:.2f} seconds")
```

```
 if torch.cuda.is_available():
        torch.cuda.empty_cache()
        print("\nFinal GPU Memory Status:")
        print_gpu_util()

if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(f"Error in main execution: {str(e)}")
        if torch.cuda.is_available():
            torch.cuda.empty_cache()
```

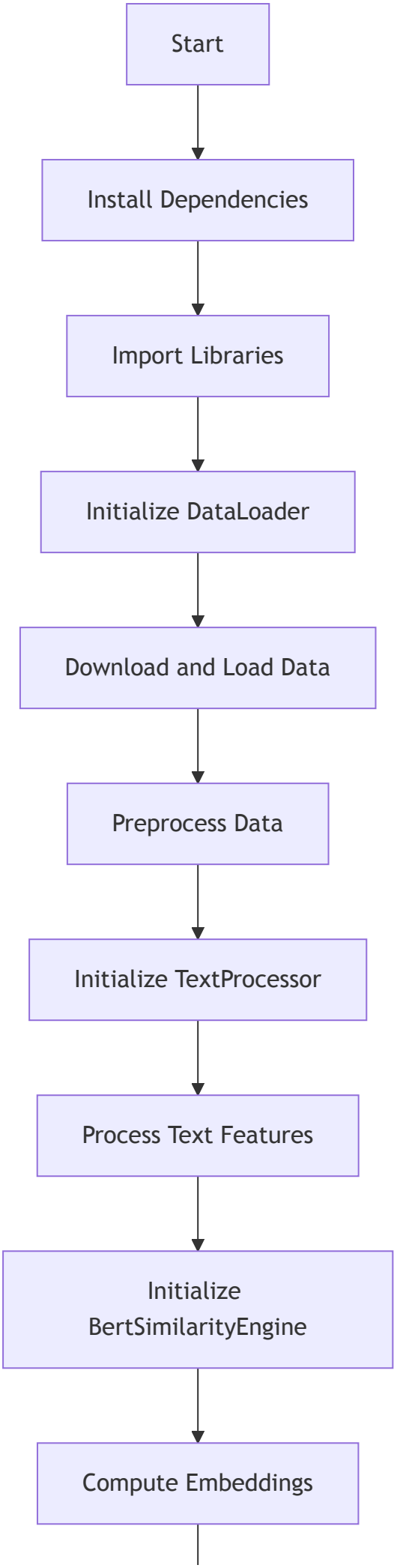
### 7. Helper Functions

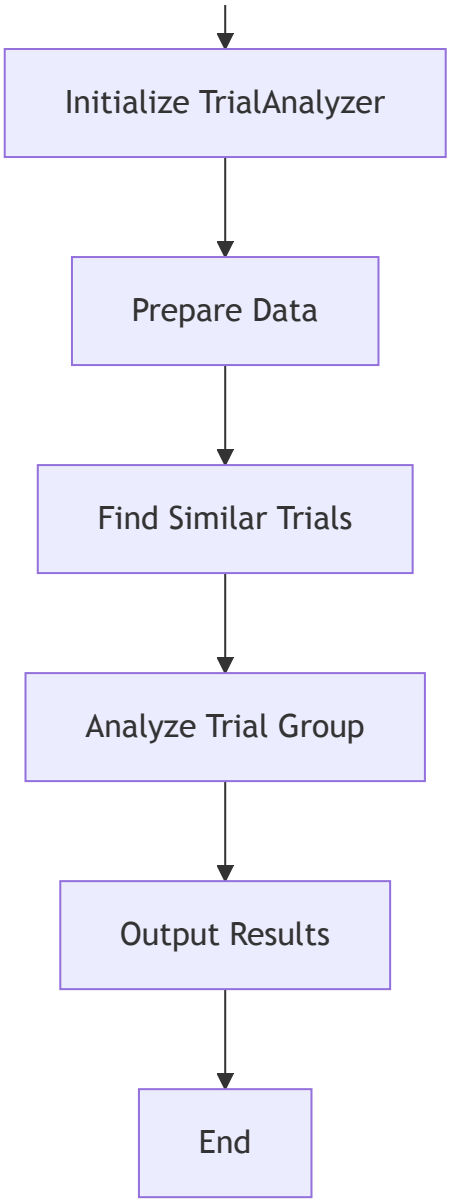
```
def print_gpu_util():
    if torch.cuda.is_available():
        print(f"GPU memory allocated:
{torch.cuda.memory_allocated(0)/1e9:.2f} GB")
        print(f"GPU memory cached: {torch.cuda.memory_reserved(0)/1e9:.2f}
GB")
```

### 8. Visualizations

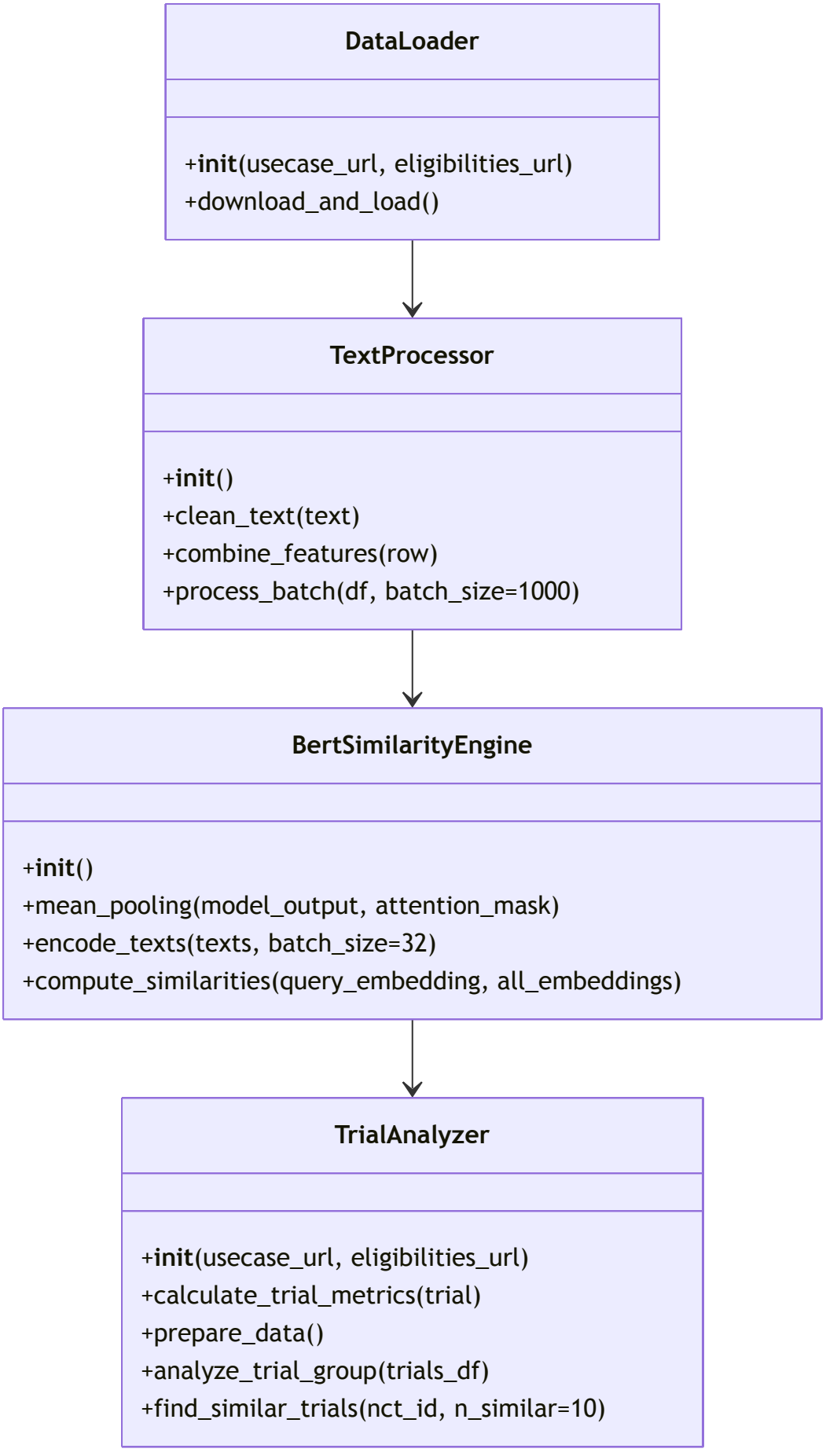
Below are some diagrams to visualize the workflow and class relationships.

#### Workflow Diagram


Start
Install Dependencies
Import Libraries
Initialize DataLoader
Download and Load Data
Preprocess Data
Initialize TextProcessor
Process Text Features
Initialize BertSimilarityEngine
Compute Embeddings




**Class Relationship Diagram**


DataLoader
+init(usecase_url, eligibilities_url)
+download_and_load()
TextProcessor
+init()
+clean_text(text)
+combine_features(row)
+process_batch(df, batch_size=1000)
BertSimilarityEngine
+init()
+mean_pooling(model_output, attention_mask)
+encode_texts(texts, batch_size=32)
+compute_similarities(query_embedding, all_embeddings)
TrialAnalyzer
+init(usecase_url, eligibilities_url)
+calculate_trial_metrics(trial)
+prepare_data()
+analyze_trial_group(trials_df)
+find_similar_trials(nct_id, n_similar=10)