The first step in the pipeline is to run **clean_text.py** on the text files located in the **essays** and **special_essays** folders. This script processes the text by expanding contractions (e.g., *"don't"* → *"do not"*), removing unnecessary punctuation while keeping sentence boundaries intact, and normalizing the text to ensure that only relevant characters remain. The cleaned text files are then saved into **cleaned_essays** and **cleaned_special_essays** folders.

Once the text has been cleaned, the next step is to use **create_data_set.py** on the **cleaned_essays** and **cleaned_special_essays** folders. This script splits the cleaned text into phrases based on punctuation, ensuring that meaningful sentence fragments are retained. Each phrase is then transformed into two versions: one where spaces are preserved (*Separated*) and another where spaces are removed (*Concatenated*). The processed data is stored as CSV files in the **processed_essays** and **processed_special_essays** directories.

Following this, the **create_data_set.py** script processes these CSV files, ensuring that the *Concatenated* and *Separated* text data pairs are properly structured for segmentation tasks.

Next, **create_data_set_with_random_words.py** is used after **create_data_set.py** to generate a dataset where random words are inserted between phrases with a certain probability. This creates additional variability in the data, making the models more robust to different input formats. The resulting dataset is saved as CSV files in the **processed_essays** and **processed_special_essays** directories.

Then, **create_labels_for_bert.py** is used to generate labeled segmentation data from the processed CSV files. This script takes each *Concatenated* phrase and its corresponding *Separated* version and assigns a binary segmentation label to each character. The resulting labeled data is stored as JSON files in the **segmentation_jsons** folder, which can later be used for training models like BERT and RoBERTa.

However, before proceeding with training, the dataset must be refined. **delete_big_labels.py** is used to sift through the labeled segmentation data and remove instances where labels exceed the 512-character limit. Since models like BERT and RoBERTa have a maximum input length, this step ensures that only valid training examples are retained.

The **custom_character_transformer.py** script is responsible for training a character-level segmentation model using a transformer-based architecture. It loads a dataset of segmented and unsegmented phrases, tokenizes them at the character level, and trains a model to predict segmentation labels. The model is trained using a binary classification approach for each character in the phrase, with padding to handle variable-length sequences. The training process includes logging batch-wise losses and evaluating accuracy on a test set.

The **custom_character_transformer_inference.py** script is designed for inference using the trained transformer model. It loads the trained model and the character-to-index mapping, then allows the user to input phrases for segmentation. The model predicts binary labels indicating where segmentations should occur. The script runs interactively, continuously accepting user input until the user decides to exit.

The **fill_essay_with_characters.py** script is a data augmentation tool that modifies text files by randomly inserting special characters and additional letters into words. This introduces variability in the dataset, which can help train models to be more robust. The script processes all text files in a specified directory and saves modified versions of them, ensuring they retain their readability while adding noise.

The **ninja_inference.py** script utilizes the wordninja library to segment concatenated words into meaningful components. This is useful for breaking down words that have been combined without spaces, such as those found in certain datasets or user-generated content. The script demonstrates its functionality by segmenting a predefined string.

The **roberta.py** script trains a RoBERTa-based token classification model to perform character-level segmentation. It loads preprocessed data, tokenizes it at the character level using RoBERTa's tokenizer, and trains a model to predict segmentation points. The model's performance is evaluated using accuracy metrics, and the trained model is saved for later use.

The **bert.py** script follows a similar approach to **roberta.py** but uses a BERT-based model instead. It includes a custom character-level tokenizer that maps ASCII characters to unique indices and processes input sequences accordingly. The training loop involves optimizing the model with AdamW and evaluating its segmentation accuracy on a validation dataset.

The **bert_inference.py** script provides an inference pipeline for the trained BERT segmentation model. It loads the trained model and character-level tokenizer, accepts user input for segmentation, and outputs predicted labels. The script ensures that input sequences are properly tokenized, padded, and processed before being fed into the model.

The **roberta_inference.py** script is designed for performing inference using a trained RoBERTa-based token classification model for text segmentation. It begins by loading the trained RoBERTa model and tokenizer, ensuring they are set to evaluation mode. The script allows users to input phrases interactively, where each phrase is tokenized at the character level and converted into a format suitable for the model. The model then predicts segmentation labels, assigning each character a classification that indicates where breaks should occur. The output consists of binary labels corresponding to the input sequence, which can be used to determine where spaces should be reintroduced. The script runs in a loop, continuously accepting user input until explicitly exited.