**1.INTRODUCTION**

Detecting stylistic boundaries within long texts is a fundamental challenge in computational linguistics and authorship analysis. As texts are increasingly co-authored, edited, or compiled from multiple sources, there is a growing demand for analytical tools that can detect stylistic shifts within a single document. Traditional authorship attribution methods typically assign a single author label to an entire document, which becomes inadequate in the presence of internal stylistic variation. This motivates the need for finer-grained, unsupervised segmentation techniques.

One prominent method addressing this challenge is the Impostors Method, originally proposed by Koppel and Winter (2014) [2]. Rather than using standard supervised classification, this method compares the target text to unrelated “impostor” texts by randomly selecting features and measuring similarity. The target document is said to align more with one author if, across many random feature subsets, it is more similar to that author's documents than to impostors. This approach reduces dependency on overfitted feature sets and provides robustness across domains.

The General Impostors Method, introduced by Seidman (2013) [3], extended this framework by allowing multiple documents per author and incorporating more robust similarity comparisons. Seidman’s method was based on the original Impostors approach initially conceptualized by Koppel and Winter, and later formalized in their 2014 publication. The General Impostors framework proved highly effective, winning multiple PAN authorship verification competitions [4] and becoming a foundation in authorship verification.

Despite its success, the classical impostors framework is limited by its reliance on shallow features, such as character n-grams or term frequencies, which do not capture higher-order syntactic or semantic patterns, particularly in short or stylistically varied segments.

To overcome these limitations, Volkovich and Avros (2025) [1] proposed the Deep Impostors framework, integrating deep learning models such as CNN and BERT into the impostor comparison paradigm. This approach embeds text into high-dimensional vector spaces, enabling the capture of richer linguistic patterns. The model distinguishes between two stylistically unrelated impostor corpora and generates a signal over the target document based on its stylistic alignment.

This project builds directly on the Deep Impostors framework. Our goal is not authorship verification, but the detection of stylistic change points within a text. We segment the text into chunks, classify them using deep neural models trained to distinguish between impostors, and aggregate the results into a stylistic signal. By computing rolling distances over this signal, we can detect stylistic shifts, which often indicate changes in authorial voice or editorial origin.

To further enhance robustness, we apply a voting mechanism based on Spearman correlation [10] across multiple impostor pair signals. This mechanism emphasizes stylistic transitions that are consistently detected by several classifier configurations, increasing segmentation reliability.

The following sections describe the mathematical principles and architectural components of the system, including word embedding, classification, signal generation, and change point analysis.

**2. MATHEMATICAL BACKGROUND**

This section outlines the core mathematical concepts and algorithms underpinning our approach. It covers the techniques employed for word embedding, binary classification, signal construction, and unsupervised change point detection.

**2.1 Word Embedding**

To apply machine learning methods to text, it is necessary to convert words into a numerical format. This is done using word embeddings, dense vector representations that encode semantic and syntactic information about words in a way that machines can interpret.

In essence, an embedding maps each word in the vocabulary to a point in a high-dimensional space, such that similar words are located close together. This allows neural models to detect relationships and patterns that would be invisible in raw text form.

In this project, we use two common embedding approaches: Word2Vec [5] and BERT [6]. Word2Vec, specifically the skip-gram variant, learns word vectors by predicting surrounding context words. Each word is assigned a fixed-length vector (e.g., 100 or 300 dimensions), and words with similar meanings tend to cluster together in this space. BERT generates context-sensitive embeddings, meaning that the same word can have different representations depending on its sentence context. This is achieved using a transformer-based model that captures rich semantic and syntactic dependencies across the entire input sequence.

These embeddings serve as the foundation for downstream classification tasks, allowing the system to learn stylistic features of text in a numerical, model-friendly format.

**2.2 Classification Models**

Following tokenization and embedding, the next stage in our pipeline involves training a binary classifier to detect stylistic patterns within individual batches of text. The goal of this classification step is not to assign explicit author labels, but rather to assess which of two unrelated impostor styles a given batch more closely resembles.

Each classifier assigns a binary score (0 or 1) to every batch, indicating whether it is more stylistically aligned with impostor A or impostor B. These per-batch predictions are subsequently aggregated into fixed-size chunks to construct a one-dimensional **stylistic signal** over the entire document. This signal reflects stylistic continuity or fluctuation and forms the basis for identifying potential change points between different authors or sources.

In this project, we evaluate two classification architectures. The first is a convolutional model augmented with a BiLSTM layer, designed to extract local stylistic features and capture longer-range dependencies. The second utilizes the BERT language model, which encodes each batch using a transformer-based structure that accounts for deep contextual relationships. Both classifiers produce soft scalar predictions (values in the range [0,1]), which are later used to construct the chunk-level stylistic signal.

**2.2.1 CNN + LSTM Classifier**

**Convolutional Neural Networks (CNNs)** [9] are widely used in text classification due to their ability to capture local sequential patterns. In our framework, CNNs are employed to process embedded batches of text and produce scalar outputs representing stylistic similarity to a given impostor class.

Each input batch is represented as a matrix of shape , where is the number of tokens in the batch (e.g., 50 words) and is the embedding dimension (e.g., 100 or 300 for Word2Vec). One-dimensional convolutional filters with various kernel sizes (e.g., 3, 6, 12) are applied across the input matrix to extract n-gram-like stylistic features. Each filter produces a corresponding feature map that emphasizes local stylistic cues.

The output of the CNN is then passed to a Bidirectional Long Short-Term Memory (BiLSTM) layer, which processes the feature maps in both forward and backward directions. This allows the model to capture longer-range dependencies and temporal context, which is particularly valuable in authorship analysis where stylistic indicators may span across multiple tokens [8].

The BiLSTM outputs are fed into a fully connected dense layer (e.g., 1024 units) with ReLU activation, followed by a dropout layer (e.g., rate = 0.25) for regularization. Finally, a sigmoid-activated output unit produces a probability in the range [0,1], indicating the batch’s alignment with one of the two impostors.

This CNN–BiLSTM architecture offers a balance between local feature detection and global sequence modeling. The CNN extracts short-range stylistic elements (e.g., lexical phrases), while the BiLSTM captures document-level flow. Together, they produce a rich stylistic signal that improves the sensitivity and accuracy of change point detection.

**2.2.2 BERT Classifier**

Bidirectional Encoder Representations from Transformers (BERT) [6] is a transformer-based language model that has significantly advanced natural language processing (NLP). Unlike CNNs, which rely on fixed-size filters to extract local features, BERT captures rich contextual dependencies by attending to all tokens in both forward and backward directions. This enables the model to learn nuanced syntactic and semantic patterns within each input batch.

In our framework, we use BERT to classify embedded batches of text according to their stylistic similarity to one of two impostor corpora. Each batch, typically consisting of 50 tokens, is processed using a pre-trained BERT model (e.g., bert-base-uncased) with an added classification layer. During training, the model is fine-tuned on batches labeled as either 0 or 1, based on their source impostor set. This process updates both the classification head and the internal weights of BERT, allowing it to produce binary stylistic predictions.

The architecture includes the BERT transformer encoder followed by a dense output layer with sigmoid activation. The final output is a scalar in the range [0,1], representing the likelihood that the batch aligns with one impostor’s stylistic profile. As with the CNN-based model, these predictions are later used to construct the document-level stylistic signal.

Compared to CNN–BiLSTM models, BERT offers superior performance in capturing long-range dependencies and contextually sensitive features. However, this benefit comes at the cost of increased computational complexity. Still, its integration into our system, based on the Deep Impostors framework [1], significantly improves the quality and stability of the stylistic signal, especially in cases where subtle stylistic distinctions exist across textual segments.

**2.3 Signal Generation**

After classifying each batch using either the CNN or BERT model, we obtain a sequence of binary outputs, where each batch receives a value of 0 or 1, indicating its stylistic alignment with one of the two impostors. These predictions are then aggregated into a continuous **stylistic signal** over the entire document.

To construct the signal, classified batches are grouped into fixed-size chunks. Each chunk typically contains consecutive batches. For each chunk, we compute the average of its batch-level predictions. Since the individual predictions are binary, the chunk score is a real number in the interval , representing the proportion of batches classified as belonging to one of the two styles.

Formally, for a chunk containing predictions , the signal value is calculated as:

This yields a one-dimensional signal vector that captures stylistic variation across the text. Signal values near 0 indicate similarity to one impostor style, while values near 1 indicate similarity to the other. Stable regions in the signal reflect stylistic consistency, while sharp fluctuations suggest potential change points, possibly corresponding to different authors or sources.

This signal generation process is adapted from the methodology proposed by Volkovich and Avros [1], and serves as the foundation for our subsequent change point detection algorithm.

**2.4 Change Point Detection via Rolling Distance**

Once the chunk-level stylistic signal has been generated, the next step is to identify points of significant stylistic change within the text. This is achieved using a change point detection technique based on rolling distance calculations, which quantify how much the current signal deviates from recent stylistic behavior.

We define a rolling distance function that computes the average squared difference between the signal value at chunk , denoted ​, and the values of the preceding chunks:

where is the size of the sliding window. This function is only computed for chunks where , as it requires at least prior values for comparison. The resulting sequence of values forms a new signal that captures stylistic fluctuation. Peaks in this signal correspond to points of high stylistic deviation and are interpreted as potential change points.

This method is simple, computationally efficient, and easy to interpret. It enables unsupervised detection of stylistic boundaries without the need for labeled training data. Compared to more sophisticated techniques like Dynamic Time Warping (DTW) [7], this squared-difference approach performs well under the assumption of discrete stylistic shifts and is well suited to chunk-based signal representations.

Our implementation is based on the approach described in the Deep Impostors framework [1], where abrupt changes in the stylistic signal are used to infer transitions between authors or sources.

**2.5 Alternative Signal Similarity: Spearman Correlation**

In addition to rolling distance, we optionally compute Spearman rank correlation between stylistic signals derived from different impostor pairs. This technique allows us to compare the shape or trend of the signals rather than their absolute values, thereby assessing the consistency of classifier behavior.

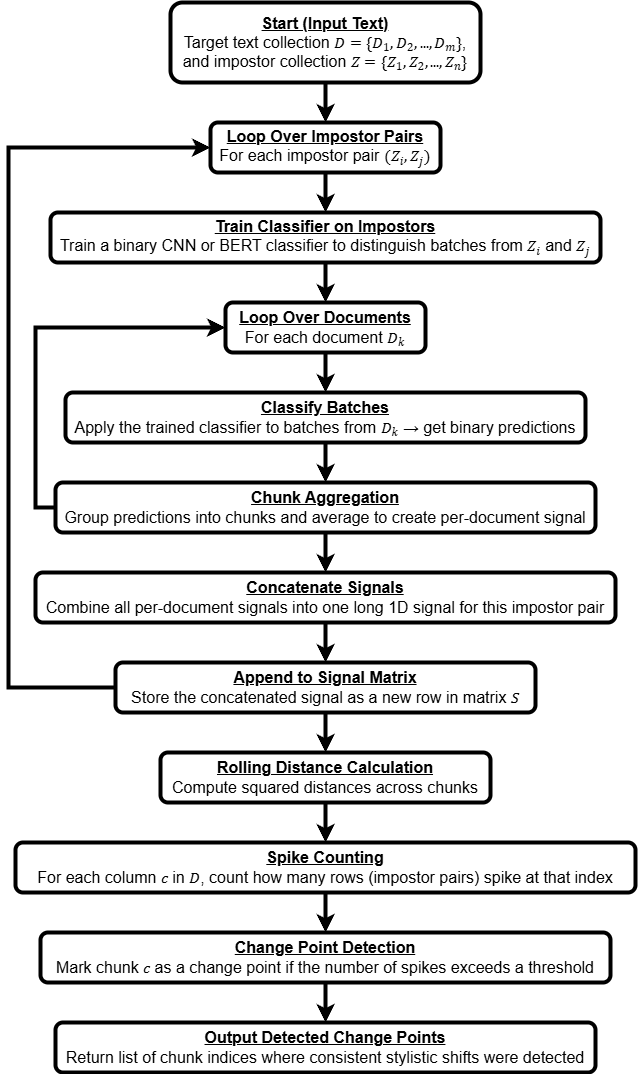
Let ​ and ​ denote two such signals (e.g., rows from a signal matrix). The Spearman correlation [10] is calculated as the Pearson correlation between the rank-transformed values of ​ and ​:

This correlation measure is particularly useful for:

* Detecting outlier impostor pairs whose signals diverge significantly from others,
* Identifying highly correlated classifiers that may indicate stable stylistic patterns,
* Optionally weighting or filtering signals before applying a voting scheme.

By computing a Spearman correlation matrix over all signal pairs, we can visualize inter-signal agreement using a heatmap or leverage this information to enhance the final consensus segmentation process.

**3. MODEL**

This section presents the full processing pipeline developed in this project, illustrating how the mathematical techniques described earlier are integrated to perform stylistic change point detection. The model progresses through a sequence of structured stages, transforming raw textual input into a final set of identified stylistic boundaries.

**Step-by-Step Process Overview**

1. **Input Preparation**  
   The process begins with a set of target documents and a pool of unrelated impostor texts . The objective is to detect stylistic change points within by evaluating internal stylistic variation relative to contrasting impostor pairs.
2. **Iteration Over Impostor Pairs**  
   The model iterates over all unordered pairs of impostors , where each pair defines a binary stylistic contrast. For each pair, a classifier is trained once, and the entire target corpus is processed using this fixed contrast. This design ensures that all documents are consistently classified with respect to the same stylistic reference before proceeding to the next impostor pair.
3. **Classifier Training**  
   A binary classifier either a CNN–BiLSTM model or a fine-tuned BERT is trained to distinguish between batches sampled from the two impostor corpora ​ and ​. This training process establishes a learned stylistic boundary between the two sources and is performed independently of the target documents.
4. **Application to Target Texts**  
   The trained classifier for the current impostor pair is applied to all documents in the target corpus. Each document ​ is tokenized and divided into fixed-length batches (e.g., 50 words per batch). These batches are transformed into embeddings using either Word2Vec or BERT, and then passed through the classifier to generate a sequence of binary predictions, each indicating stylistic alignment with one of the two impostors.
5. **Chunk-Level Aggregation**  
   The binary predictions for each document are grouped into fixed-size chunks (e.g., 8 consecutive batches). For each chunk, the average of its binary predictions is computed, yielding a real-valued score in the range . A score near 0 indicates alignment with one impostor, while a score near 1 suggests alignment with the other. This process results in a one-dimensional stylistic signal for the document, capturing how its style evolves with respect to the current impostor pair.
6. **Signal Concatenation**  
   For each impostor pair, the chunk-level signals from all target documents are concatenated into a single continuous vector. This aggregated signal simulates an unsegmented stream, allowing for the unsupervised detection of stylistic transitions across document boundaries.
7. **Append to Signal Matrix**  
   The complete signal for each impostor pair is stored as a row in the signal matrix , where is the number of impostor pairs and is the total number of chunks across the entire target corpus. Each row in represents the stylistic alignment of the corpus with respect to a specific impostor contrast.
8. **Computation of Rolling Distances**  
   A rolling window of size is applied to each row of the signal matrix , calculating the average squared deviation between each chunk and its preceding values. This process produces the rolling distance matrix , where local peaks correspond to points of sharp stylistic deviation.
9. **Peak Detection in Distance Signals**  
   Local maxima are identified for each row in the rolling distance matrix . These peaks correspond to positions in the chunk sequence where the stylistic flow, relative to that impostor pair, undergoes significant change.
10. **Consensus-Based Spike Counting**  
    For each column index in the matrix , the number of impostor pairs exhibiting a peak at that position is counted. This aggregation step captures global agreement across multiple stylistic perspectives regarding the location of potential shifts.
11. **Change Point Determination**  
    A chunk index is designated as a change point if the number of impostor pairs exhibiting a peak at that position exceeds a predefined threshold (e.g., 80%). This voting-based mechanism mitigates false positives by requiring broad consensus across multiple classifiers.
12. **Final Output of Change Points**  
    The model outputs a list of chunk indices where consistent and robust stylistic change points have been detected. These indices are interpreted as likely transitions between different authors, documents, or stylistically distinct sections within the text.

Input:

- Target text collection D = {D1, D2, ..., Dm}

- Impostor collection Z = {Z1, Z2, ..., Zn}

Output:

- Detected change points in the concatenated text

1. Initialize empty matrix S to store per-pair signals

2. For each impostor pair (Zi, Zj) in Z:

a. Train a binary classifier on batches from Zi vs. Zj

b. Initialize empty list signal\_parts

c. For each target document Di in D:

i. Apply the trained classifier to batches of Di

ii. Group batch predictions into chunks

iii. Average predictions in each chunk to create a 1D signal

iv. Append the signal to signal\_parts

d. Concatenate all signals in signal\_parts into a single 1D vector

e. Append the concatenated signal as a new row in matrix S

3. S now has shape [#impostor pairs × total\_chunks]

4. Initialize empty matrix D to store rolling distances

5. For each row r in S (i.e., each impostor pair signal):

a. Compute rolling squared difference using a sliding window of size w

b. Store the result as row r in distance matrix D

6. D now has shape [#impostor pairs × (total\_chunks - w)]

7. For each column index c in D:

a. Count how many rows (i.e., impostor pairs) have a spike at column c

b. Mark column c as a potential change point if the number of spikes exceeds a defined threshold

Return:

- List of column indices corresponding to detected change points

**4. Evaluation Test**

This section outlines the planned implementation and evaluation strategy for the proposed method. It includes both a code-level description and component-wise verification of the pipeline, as well as the intended evaluation procedure using a labeled test set.

**4.1 Implementation Overview**

This subsection describes the planned implementation workflow used to operationalize the Deep Impostors pipeline on the target dataset. The system is implemented in Python, and consists of the following components:

* **Preprocessing and batching**: Each document is tokenized and divided into fixed-size batches, which are then grouped into chunks for downstream signal generation.
* **Classifier training**: For each impostor pair , a binary classifier is trained to distinguish between batches sampled from the two sources.
* **Prediction and signal generation**: Trained classifiers are applied to all target documents to produce binary predictions, which are aggregated per chunk to yield continuous stylistic signals.
* **Signal matrix construction**: One signal is generated per impostor pair and stored as a row in the matrix .
* **Rolling distance computation**: A rolling squared difference is computed for each row in , resulting in the distance matrix .
* **Peak detection**: Local maxima are identified in each row of using scipy.signal.find\_peaks.
* **Voting mechanism** A consensus-based threshold is applied across all rows in to determine final change point predictions.

The pipeline is implemented in Python 3.10.8 using NumPy, SciPy, and Matplotlib. Classifier training is conducted using TensorFlow and scikit-learn.

**4.2 Unit Testing and Component Verification**

In addition to system-level evaluation, we plan to conduct unit testing and modular verification to ensure the correctness of key components within the pipeline. The following elements will be tested:

* **Tokenizer and Batching Module**:  
  We will verify that tokenization is consistent across documents and that batch boundaries align with the expected chunking configuration.
* **Classifier Prediction Logic**:  
  We will check that the classifier produces binary or probabilistic outputs within the expected range, and validate predictions using controlled inputs (e.g., all- batches should yield all-zero predictions).
* **Chunk Aggregation and Signal Construction**:  
  We will confirm that batch-level predictions are correctly grouped into chunks and that averaging is accurate under both hard and soft label conditions.
* **Rolling Distance Calculation**:  
  We will test the rolling distance method using synthetic input signals (e.g., flat, stepped, or noisy) to ensure expected peak behavior and robustness under null-change conditions.
* **Spike Detection and Voting Mechanism**:  
  We will verify the identification of local maxima and ensure that the voting threshold is applied consistently across all signal rows.

All unit tests will be implemented in Python using the unittest framework. Planned coverage includes both edge cases (e.g., first chunk, empty batch) and randomized inputs for stress testing and robustness.

**4.3 Results Evaluation**

To evaluate the performance of the proposed method, we plan to apply the complete pipeline to a controlled dataset constructed by concatenating several stylistically distinct texts into a single document. Ground truth change points will be manually annotated at the boundaries between these source texts.

**The evaluation procedure will use the following configuration parameters:**

* **Chunk size**: Defines the number of tokenized batches grouped together to form a chunk, the primary unit of stylistic analysis.
* **Rolling window size** : Specifies how many preceding chunks are used when computing the rolling stylistic distance at each position.
* **Change point detection threshold:** A numerical threshold that determines whether a peak in the distance signal is considered a valid stylistic transition.
* **Voting threshold:** The minimum proportion of impostor classifiers that must agree on a peak's location for it to be classified as a change point..

After executing the pipeline, predicted change points will be compared to the ground truth annotations. Evaluation will allow for a positional tolerance (e.g., ±1 **chunk**) to account for minor alignment differences.

The following metrics will be computed to assess performance:

* **Precision**: The proportion of predicted change points that match ground truth.
* **Recall**: The proportion of ground truth change points that are successfully detected.
* **F1 Score**: The harmonic mean of precision and recall.

Additionally, a visualization of the rolling distance signal will be produced, with both predicted and true change points overlaid. This will provide an intuitive summary of the method’s behavior and alignment accuracy.

This evaluation framework is designed to assess the model’s accuracy, robustness, and stylistic segmentation consistency across varying stylistic boundaries.

**REFERENCES**

1. Volkovich, Z., & Avros, R. (2025). Comprehension of the Shakespeare Authorship Question Through Deep Impostors Approach. Digital Scholarship in the Humanities. <https://www.popularmechanics.com/science/a64366995/shakespeare-authorship-ai-test/>
2. Koppel, M., and Winter, Y. (2014) ‘Determining if Two Documents are Written by the Same Author’, Journal of the Association for Information Science and Technology, 65: 178–87.
3. Seidman, S. (2013). Authorship verification using the impostors method. In CLEF 2013 Evaluation Labs and Workshop – Working Notes Papers.
4. Potthast, M., Stein, B., Gollub, T., & Hagen, B. (2013). Overview of the 2013 Author Identification Task. In CLEF 2013 Evaluation Labs and Workshop – Working Notes Papers.
5. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.
6. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT, 4171–4186.
7. Sakoe, H., & Chiba, S. (1978). Dynamic Programming Algorithm Optimization for Spoken Word Recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 43–49.
8. Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research, 57, 345–420.
9. Shrestha, P., Sierra, S., González, F. A., and Rosso, P. (2017). Convolutional Neural Networks for Authorship Attribution of Short Texts. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 669–674.
10. Spearman, C. (1904). *The proof and measurement of association between two things*. American Journal of Psychology, 15(1), 72–101.