Software Engineering Department
Braude College

Capstone Project Phase A – 61998

Authorship Verification using Impostor Projections and Siamese Networks

25-1-R-17

Adir David - 206132029
Eyal Maklada - 206459166

Git Repository

Supervisors:
Zeev Volkovich - vlvolkov@braude.ac.il
Renata Avros - ravros@braude.ac.il

Table of Contents

Abstract

The determination of true authorship is critical in the fields of digital forensics, preserving academic honesty, and analyzing historical texts. Such tools of verifying authorship as hand-crafted features are quite inefficient when facing problems of adversarial mimicking text fragmentation and domain shifts. This study offers a robust, scalable framework that combines Siamese neural networks with BERT embeddings, enhanced with CNN-BiLSTM architectures. In addition, it leverages the Impostor Projection Methodology for adversarial training, while utilizing Dynamic Time Warping (DTW), anomaly detection with Isolation Forest and K-Medoids clustering for stylistic, semantic and mimicry variability. This approach addresses many challenges that traditional methods imposed or failed to manage, indicating great potential for authorship verification across numerous domains.

1. Introduction

Nowadays, in this digital world of interconnection, the concept of authorship has emerged as highly important in digital forensics, academic integrity, and historical text analysis. Authorship verification forms another imperative subfield of authorship recognition, determining whether two given texts were authored by the same individual, while the attribution of authorship identifies who exactly the author is. With the recent rise of machine-generated content and adversarial writing strategies, conventional methods based on handcrafted features have been shown not to be effective. Therefore, quite naturally, the applications of authorship verification include fighting plagiarism and ghostwriting in academia and professional life, analyzing anonymous or adversarial cybercrime content, authenticating historical manuscripts and works of literature. A classic example is the long-drawn controversies over Shakespearean authorship, for which stylistic analysis repeatedly plays a vital role to establish one's claims to authorship.

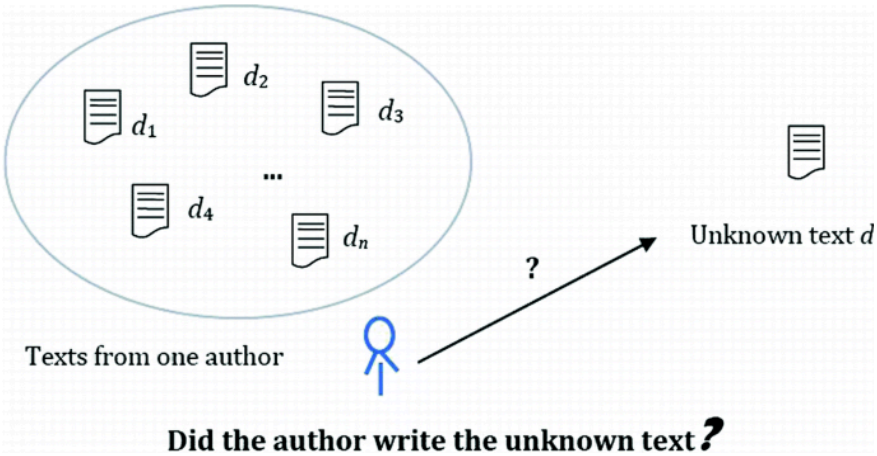


Figure 1. Representation of the authorship verification problem. Source

This proposed research attempts to address some of the important limitations of traditional authorship verification, which is usually dependent on handcrafted stylometric features such as, but not limited to, lexical patterns, syntactic structures, and textual organization. While these methods may prove effective in controlled environments, in real life they often fail. Adversarial texts styled to match another author, fragmented or short texts-such as tweets or even notes-domain variability, all are much significant challenges in which the traditional techniques fail. Moreover, hand-crafted approaches generally have the intrinsic limitation of inability to model such complex semantic and syntactic variations possessed by the natural language.

To address these challenges, this research proposes a robust and scalable authorship verification model that leverages state-of-the-art deep learning techniques. The model integrates Siamese neural networks with pre-trained Bidirectional Encoder Representations from Transformers (BERT), enabling the extraction of rich contextualized embeddings that encapsulate semantic and syntactic features. These embeddings are further refined using CNN-BiLSTM, a hybrid bidirectional long short-term memory and convolutional neural network architecture, capturing sequential dependencies critical for identifying stylistic consistencies or differences.

The proposed model will address some key challenges of authorship verification with the introduction of some new techniques for final prediction and decision-making: Impostors Projection Method, Dynamic Time Warping (DTW), Isolation Forest, and clustering with K-Medoids. These enhance the robustness, handle variable-length texts, and improve accuracy in the detection of impostors and verification and robustness of authorship verification systems.

The rest of the journal is organized as follows: Section 2 presents the related work and theoretical background on authorship verification, discussing in detail the shortcomings of previous approaches and how the proposed techniques overcome these limitations. From here, the sections unfold our expectations from this research–Section 3, the process of research and our approach–Section 4, which is the proposed model. This is further followed by an evaluation plan for Phase B of our capstone project in Section 5. And last but not least, Section 6 concludes this journal.

2. Background and Related Works

The field of authorship verification has evolved significantly, transitioning from traditional handcrafted methods to advanced neural network architectures. This section explores the theoretical foundations and existing literature, focusing on current methods, their limitations, and how the proposed techniques bypass these challenges.

2.1. Siamese Networks

Siamese networks are a type of neural network architecture specifically designed for tasks requiring pairwise comparisons, like authorship verification. The architecture consists of two identical subnetworks, which share weights and parameters, hence providing consistent feature extraction for both inputs.

As intended in their design, Siamese networks learn a similarity metric based directly on comparisons between input pairs and optimize their representations with respect to the similarity or dissimilarity of these inputs.

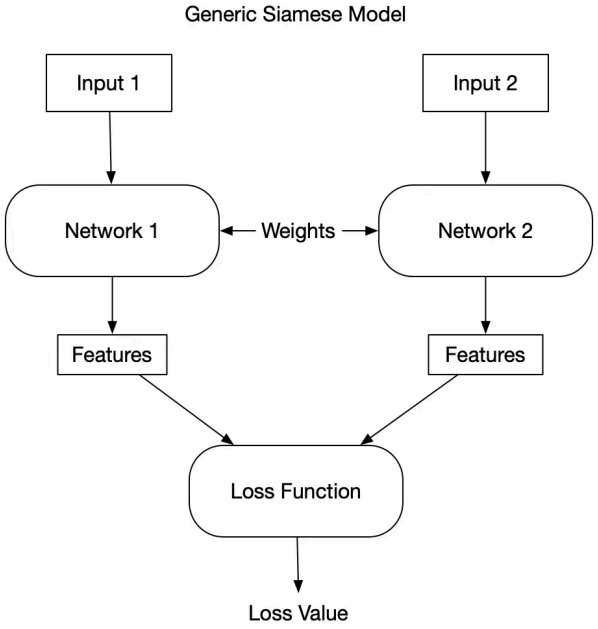


Figure 2. Generic Siamese neural network architecture. Source

Architecture Overview

Each subnetwork within a Siamese network transforms the input (e.g., a text document) into a vector embedding in a shared feature space, and the embeddings are then compared using a similarity or distance metric. The network learns to minimize the distance between embeddings of similar pairs and maximize it for dissimilar pairs. This training process enables the network to generalize to unseen pairs of inputs. It is note-worthy to mention that in most cases, the results of the distance metric calculation are used for further decision-making steps during the inference process, as opposed to the training process where it is used for optimization and loss calculation.

General Mathematical Notations

1. **Shared Weight Subnetworks**: Let X_1 and X_2 be the two input texts. Each input is passed through the shared network $f(\cdot)$:

$$E_1 = f(X_1),\ E_2 = f(X_2),$$

where E_1 and E_2 are the embeddings for the two inputs.

2. **Distance Metric**: Various distance metrics can be employed to achieve the desired comparison. Common distance metrics are:

$$d(E_1, E_2) = \left|\left|E_1 - E_2\right|\right|_2 \text{ (Euclidean distance),}$$

or

$$Sim(E_1, E_2) = \frac{E_1 \cdot E_2}{||E_1|| \cdot ||E_2||} \text{ (Cosine similarity).}$$

3. **Loss Function**: During training, the network optimizes a loss function, such as contrastive loss:

$$L = (1 - Y) \cdot max(0, m - d(E_1, E_2))^2 + Y \cdot d(E_1, E_2)^2$$

where $Y \in \{0, 1\}$ indicates whether the inputs are similar or dissimilar, m is a margin for dissimilar pairs, and $d(E_1, E_2)$ is the distance metric.

In Section 4, we plan to introduce the loss function that we use to train our proposed model.

Traditional methods for authorship verification often rely on explicit feature engineering, such as stylometric analysis, which depends on predefined linguistic features like word frequency or sentence length. However, these approaches struggle with variability in text length, fragmented inputs, and sparse datasets. Siamese networks address these limitations by learning universal similarity functions that generalize well to unseen text pairs.

They are particularly adept at handling challenges like text variability, adversarial mimicry, and sparse datasets by focusing on pairwise relationships rather than individual class labels. This ability of generalization, together with the pairwise training mechanism, makes them the ideal architecture for solving the complex and dynamic problem of authorship verification [1].

2.2. Bidirectional Encoder Representations from Transformers (BERT)

BERT, or **Bidirectional Encoder Representations from Transformers**, is a pre-trained language model that has been used to revolutionize NLP tasks due to its great ability in generating deep contextualized word embeddings. A way to represent words as vectors in the multi-dimensional space where the distance and direction between vectors reflect similarity and relationships among the corresponding words. Unlike the traditional approaches like Word2Vec or GloVe, which rely on static word representations, BERT constructs a bidirectional transformer model, considering both the left and right contexts of every token in the sentence. This implies that BERT captures much richer semantic and syntactic relationships due to this bidirectionality and will provide a holistic understanding of text structure and its meaning.

Contextualized Embeddings

Let $X = \{x_1, x_2, ..., x_n\}$ be an input sequence of tokens. After passing through BERT's transformer layers, the output embeddings are:

$$\{E_1, E_2, ..., E_n\} = BERT(X)$$

where each E_i represents the contextualized embedding for token x_i. These embeddings are subsequently fed into downstream tasks to compute pairwise similarities or, similarly to our proposed model, for further refinement by additional neural layers.

Pre-training and Fine-tuning

Pre-trained BERT models are first trained on large unlabeled corpora with objectives such as masked language modeling and next sentence prediction, allowing the model to learn general language representations. They can then be fine-tuned with comparatively smaller, labeled datasets to adapt these representations to a specific downstream task, such as authorship verification. This fine-tuning process slightly updates the model parameters so that BERT can use its deep understanding of the context of language and, at the same time, focus on task-specific features. It is worthwhile to mention that fine-tuning usually adopts strategies like using a Siamese network architecture, as exemplified in the BiBERT-AV model [3], to further enhance the model's ability in sequential dependencies. Optimization techniques, such as using the AdamW optimizer in the right learning rates for the task at hand, shall guarantee that the fine-tuned model is robust, and overfitting is avoided.

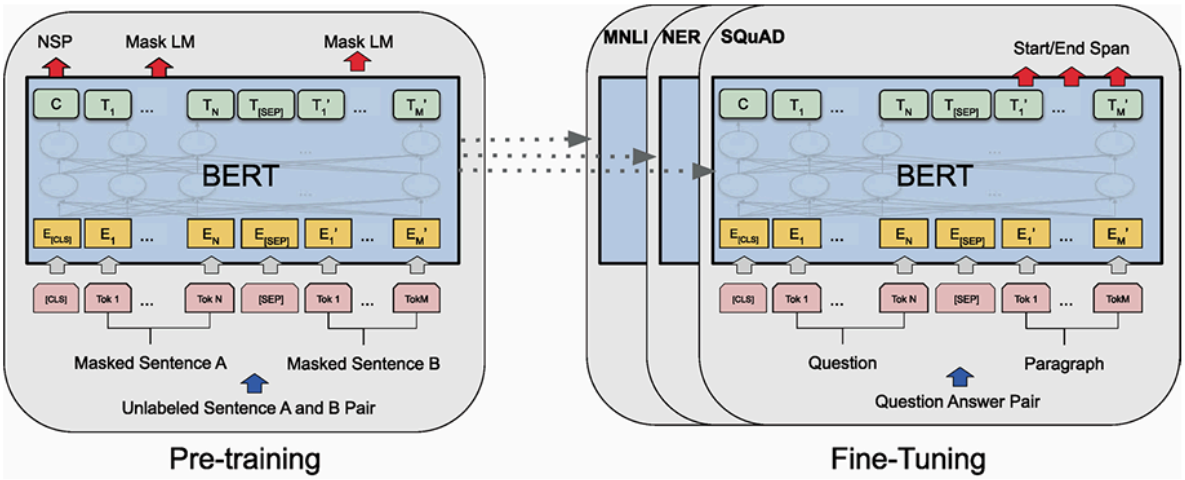


Figure 3. Overall pre-training and fine-tuning procedures for BERT [3]

With its layered attention mechanisms, BERT can capture both semantic and syntactic elements. Thus, fine-tuned embeddings from BERT are strong and flexible, requiring very little extra training data. Thus, it maintains its versatility and performance across many application domains.

Previous approaches to authorship verification were plagued by static embeddings, which could not consider relationships between words in their specific contexts and therefore were losing much stylistic and semantic richness. BERT overcomes these flaws by using layers of attention mechanisms to weigh the importance of words in context so that the embedding reflects relationships between tokens at different positions.

In this regard, BERT acts as the backbone in authorship verification by providing high-quality embeddings that encode subtle stylistic variations between texts [2]. What makes it even more powerful is its capability in handling variable-length inputs, quite fitting for fragmented, noisy, or incomplete data—common issues found in real-world datasets. This enables leveraging pre-training on large-scale corpora and generalizing well across domains, thus adapting to diverse tasks in authorship verification. Moreover, BERT forms a basic building block for state-of-the-art authorship verification systems by engineering a strong input representation that encodes both global and local contextual dependencies, thereby enabling valid pairwise comparisons between text samples [1].

2.3. CNN-BiLSTM

Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (BiLSTM)

Convolutional networks are a type of feed forward neural network developed to learn spatial hierarchies of features from the input data in fully automatic and adaptive

manners, which originated from the field of computer vision, and have been successfully generalized to other Natural Language Processing (NLP) tasks. They capture input features by employing kernels. Kernels are automatic learning filters that scan the input sequence for n-gram features, integrating pattern recognition without manual intervention [2].

Mathematically, for an input sequence $X \in R^{n \times d}$, a filter $W \in R^{k \times d}$ performs a convolution to produce a feature map $c \in R^{n-k+1}$, where each element c_i is given by $c_i = f(W \cdot X_{i:i+k} + b)$, f being a non-linear activation function and k the kernel size. Stacking and pooling these feature maps yields representations that extract progressively more higher-level features, while several pooling techniques (max or average pooling) takes these extracted features and create a more compact and invariant representation.

In the use case of a vector embedding input, CNN filters then capture local stylistic patterns within these representations, enhancing their expressiveness by highlighting subtle stylistic cues [1]. After the CNN layers generate a richer representation of features, one could use a BiLSTM layer to incorporate these localized features with contextual dependencies from both forward and backward directions, thus obtaining more comprehensive and context-sensitive embeddings [3]. Therefore, the CNN-BiLSTM pipeline turns static embeddings into dynamic enriched sequences, which better reflect complex stylistic constructs of authorship.

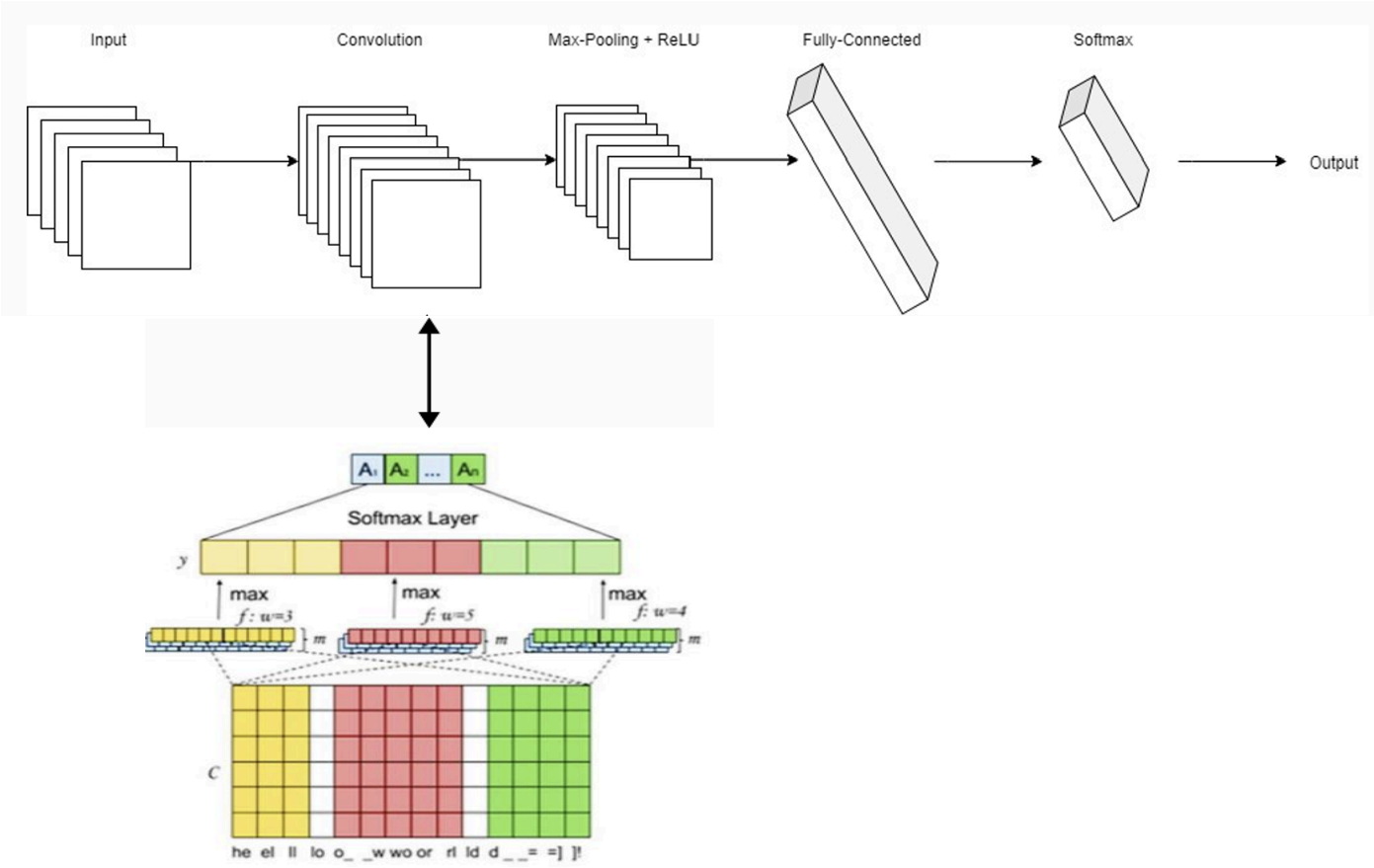


Figure 4. A Convolutional neural network architecture [12].

In order to understand BiLSTM layers, first we introduce Long Short-Term Memory (LSTMs):
LSTMs are part of a big family known as recurrent neural networks, used predominantly to learn, process and classify sequential data because they may learn long-term dependencies between time steps of data. Formally, at time t, given an input x_t, the LSTM updates its hidden state h_t and cell state c_t as presented in the following figure:

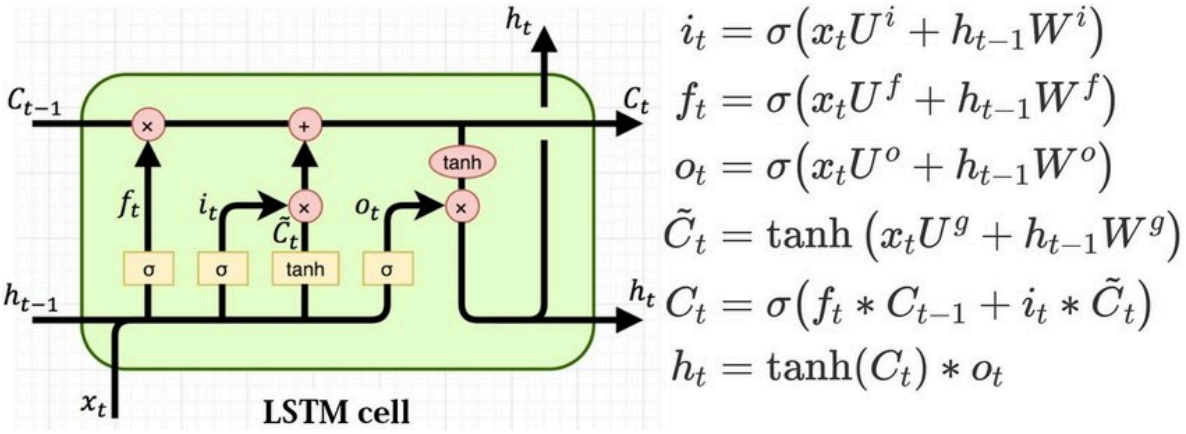


Figure 5. Structure of a LSTM Cell. Source

Here, f_t, i_t, o_t are forget, input, and output gates respectively and σ is the sigmoid activation. These mechanisms allow the LSTM to selectively remember or forget information, thus effectively capturing long-term dependencies in sequential data.

BiLSTM extends the unidirectional LSTM by introducing a forward LSTM and a backward LSTM that process the sequence in opposite directions. Mathematically, if $h_t^{\rightarrow}$ and $h_t^{\leftarrow}$ are the hidden states from the forward and backward LSTMs, respectively, then the output at time step t often looks like the concatenation: $h_t = \left[h_t^{\rightarrow} \cdot h_t^{\leftarrow}\right]$. This layout allows access to both past and future context at the same time: past-to-future, future-to-past; in representations which would give out sensitivities in the order of a sequence from both ways, enabling the model to preserve the global flow of textual structure and find out broader stylistic relationships between sentences and documents. In turn, the resulting embeddings are much better for such tasks as authorship verification, whose stylistic cues often rely on contextual patterns which may appear anywhere in the text.

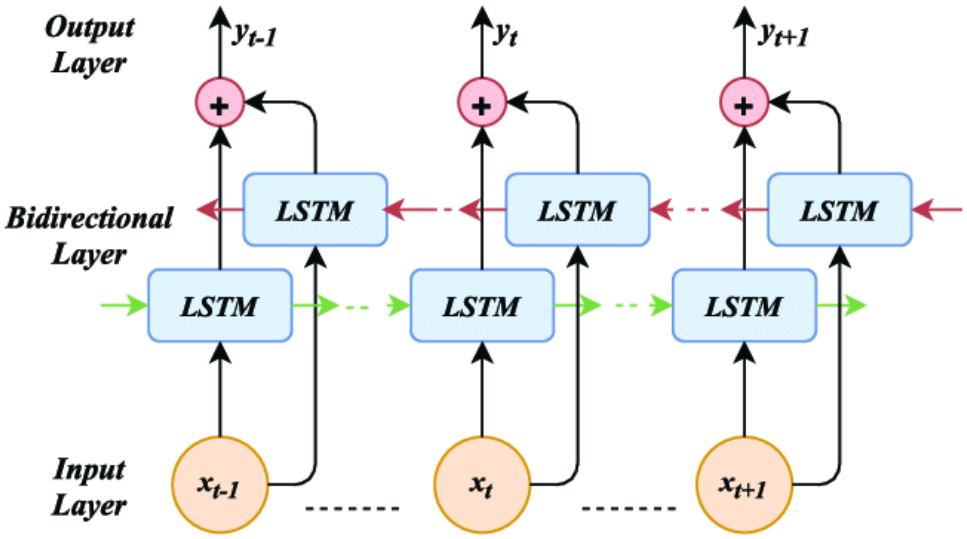


Figure 6. BiLSTM model showing the input and output layers. The red arrows represent the backward sequence track and green the forward sequence track. Source

The integration of CNN and BiLSTM layers brings synergy between localized feature extraction and global sequential modeling. The CNN extracts meaningful stylistic local and spatial sub-patterns from embedding-for example, lexical choices, usage-that identify stylistic features such as short sequences that are particularly useful in fragmented or short texts. BiLSTMs then track how these patterns evolve and interact throughout the text, capturing broader stylistic and structural dependencies. This hybrid approach has been explored for state-of-the-art authorship verification frameworks in order to capture both the fine-grained and holistic characteristics of an author's writing style [1][3].

This arrangement—CNN followed by BiLSTM—is ideal for authorship verification because:

- CNNs are very good at extracting fine-grained stylistic cues, such as lexical patterns, which may appear within short fragments of text. These features often form the core stylistic patterns that characterize an author.
- Enriched features can be processed by BiLSTMs to point out how these patterns will evolve across the text and capture long-range dependencies with the assurance of preserving global stylistic structure.

Changing the order of this-for instance, using the BiLSTM before CNN-would make the global sequential patterns before the capture of localized cues. This could overlook some important stylistic details, often carrying much weight in small text patterns, such as phrase structures or specific word choices in authorship tasks. First isolating local features with CNNs and then using a BiLSTM for global contextualization helps in ensuring a comprehensive and hierarchical feature representation.

This is justified because state-of-the-art frameworks of authorship verification are already employing this approach, such as the work by Volkovich, Zeev. (2020) [10], which emphasized how CNN contributes to finding short text patterns and how BiLSTM is complementing it by handling sequence-based dependencies.

Most classic approaches remain deficient in embedding both features; instead, they rely on the static word embedding provided by, for instance, Word2Vec and GloVe, together with isolated feature extraction regardless of complex patterns of texts. This further limits their representational power of fine-grained stylistic dependencies, along with consideration to context variability.

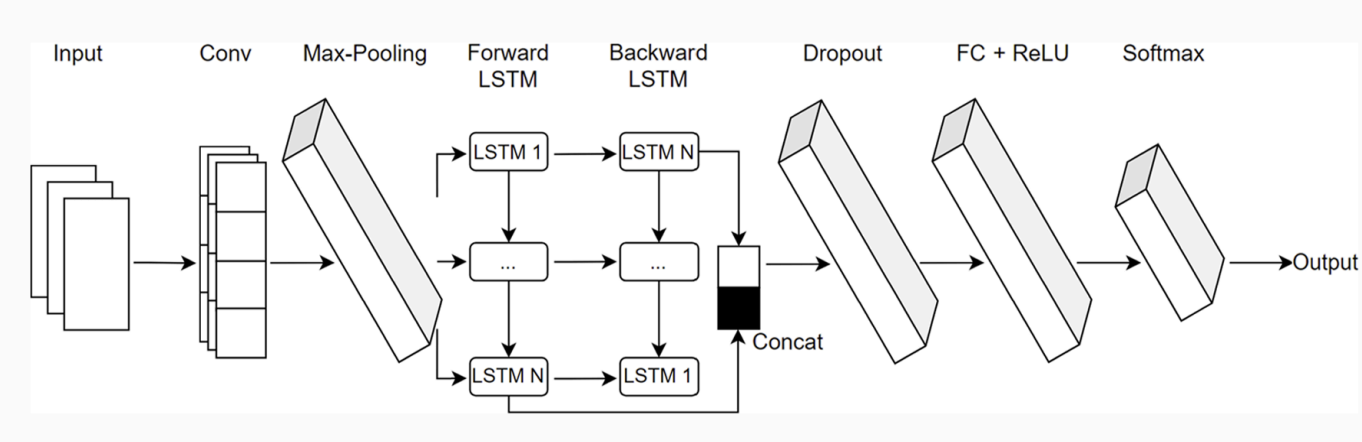


Figure 7. The proposed CNN-BiLSTM hybrid network architecture. [12]

By putting together these components, the architecture of CNN-BiLSTM ensures robust text representation and overcomes these limitations in the way CNNs capture spatial patterns with BiLSTMs' strength regarding temporal modeling. CNN considers fine-grained stylistic details, and BiLSTM integrates this into a wide contextual framework. The fusion overcomes the limitations that the traditional method has been reeling from, including how it does not handle either localized information or sequential quite effectively. This hybrid model turns out to be very good in practice for finding stylistic trends across fragmented or short texts, which is really of great value in authorship verification [3], providing a robust solution for authorship verification tasks.

2.4. Impostors Projection Methodology

The **Impostors Projection Methodology** is a robust approach in authorship verification that enhances model effectiveness of a model by introducing "impostor" pairs, namely text samples that are intentionally taken from different authors in order to simulate adversarial situations. By exposing the model to these challenging negative examples, the methodology compels the model to learn subtle stylistic differences and deeper semantic and syntactic distinctions between genuine authorial text pairs and deceptive ones. The method is based on labeling sequential small fractions of texts in the tested collection by a deep network trained on a pair of impostors.

This method was introduced in the literature for addressing issues in dealing with limited training data, which include external baselines as impostor documents to compare. Early works such as Seidman (2013) [4] introduced the general Impostors Method to compare document similarities against impostor texts, while subsequent refinements such as Potha, et al. (2017) [5] provided enhanced selection of impostor documents and more sophisticated techniques of comparison.

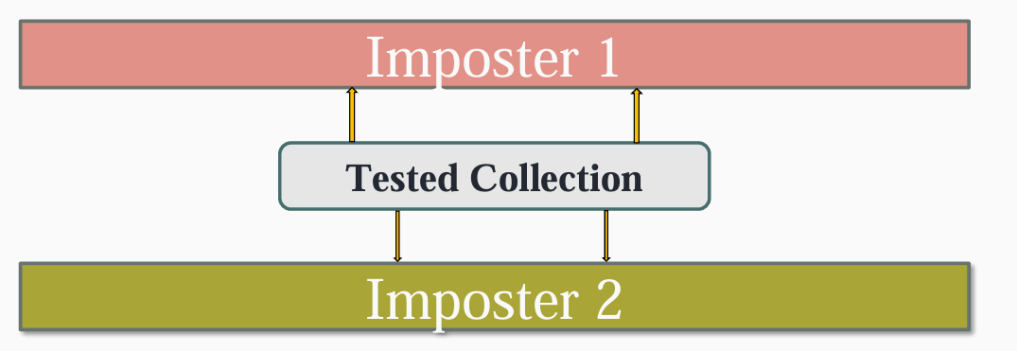


Figure 8. Texts from the tested collection are processed and labeled using the networks trained on impostor pairs. [12]

Impostor Projection has been adopted in concert with advanced neural network architectures, such as Siamese networks, through embedding both genuine and impostor text pairs to a shared latent space using deep encoders like BERT. This method has come to be known as the “Deep Impostors Projection Method”.

Impostor pairs are constructed by deliberately selecting texts from different authors. These serve as negative examples, simulating adversarial conditions where the stylistic similarities of the texts could deceive a weaker model. For the genuine text pairs, and for impostor pairs from different authors, the goal is to embed genuine pairs closer together in the latent space while maximizing the separation of impostor pairs, thus achieving the goal of exposing the system to realistic adversarial conditions.

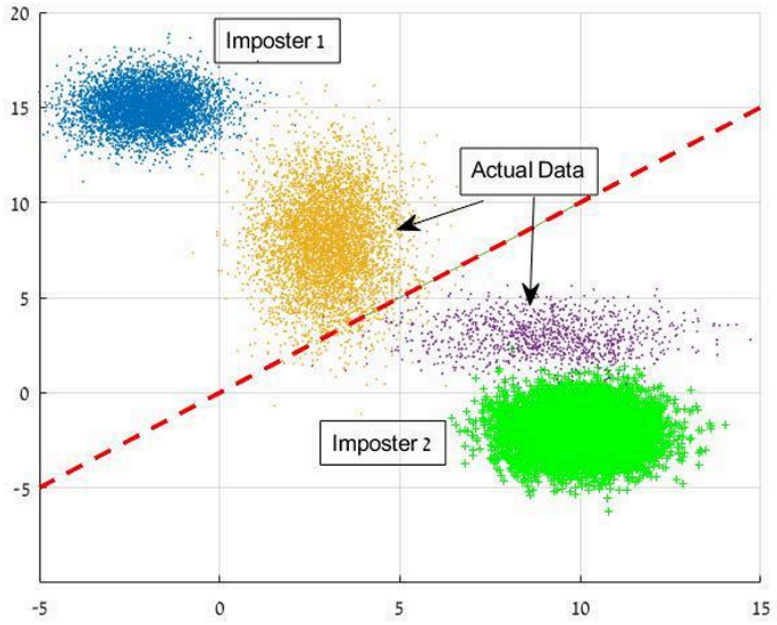


Figure 9. Method exemplification of a trained deep network providing a separating manifold. A text pair under consideration is classified according to the splitting rule generated by the trained network. [12]

Earlier approaches have relied heavily on the assumption that all training data were either positive or negative examples. However, applications in the real world are adversarial, where boundaries are blurred due to stylistic mimicry. Impostor Projection creates more challenging training scenarios, therefore forcing models to become more robust against adversarial inputs.

This strategy keeps the model from becoming overly confident in superficial features such as word frequency or sentence length, instead emphasizing deeper stylistic and semantic patterns that are harder to fake. This approach addresses key challenges within state-of-the-art authorship verification tasks: mimicry and stylistic variability.

2.5. Dynamic Time Warping

Dynamic Time Warping (DTW) is a widely-used sequence alignment algorithm, originally developed for applications like speech recognition and time-series analysis [5]. It has since been adapted for text analysis tasks, including authorship verification, where it excels in aligning sequences of varying lengths or styles.

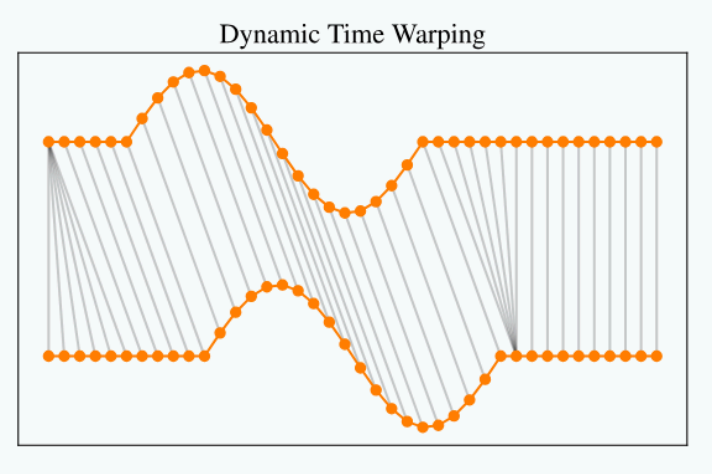


Figure 10. Distance between two time-series using DTW**.** Source

Mathematical Notations and Algorithm Let $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ be two sequences. We define a local distance function $d(x_i, y_j)$ to quantify how dissimilar the elements x_i and y_j are (by default, Euclidean distance in an embedding space).

DTW constructs a cost matrix D of size $(M + 1) \times (N + 1)$ using a dynamic programming approach:

Initialization:

$$D[0, 0] = 0$$

$D[0, j] = \infty$; for all j. $D[i, 0] = \infty$; for all i.

1. **Recurrence:**
   $D[i, j] = d(x_i, y_j) + min(D[i - 1, j], D[i, j - 1], D[i - 1, j - 1])$ for $i = 1 \ldots n$ and $j = 1 \ldots m$

2. **Optimal Warping Path:** The final DTW cost is $D[n, m]$. An optimal alignment path can then be found by backtracking from (n, m) to $(0, 0)$, revealing how the two sequences align non-linearly in "time."

In the proposed model, we are solely interested in the distance matrix of the two time-series, calculated as the first step in the DTW algorithm. We intend to elaborate on this interest, in the Proposed Model section–Section 4.2.a.

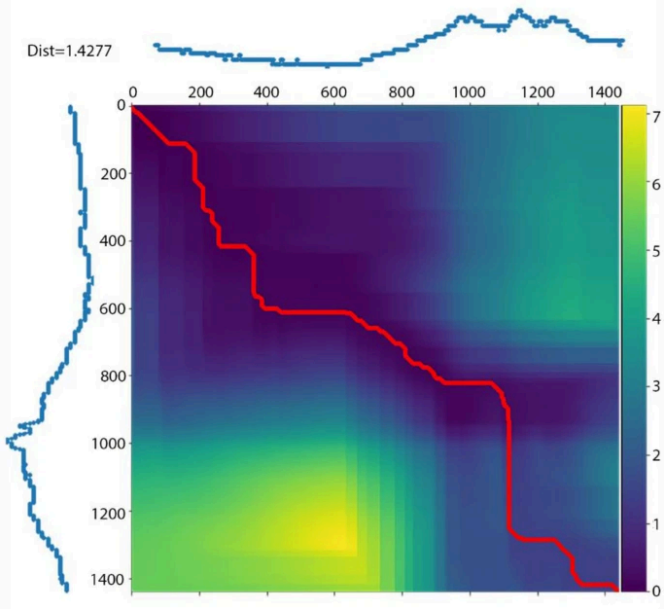


Figure 11. Distance matrix between two time-series calculated using DTW**.** Blue indicates the shortest distances, and the red line shows the shortest "optimal warping" path. **Source**

DTW captures both the local and global patterns that sometimes get lost in more rigid alignment approaches by allowing for flexible stretching or compressing of the sequences.

Traditional aggregation methods often fail to capture temporal or structural variability in texts, oversimplifying their representation. DTW overcomes these limitations by dynamically adjusting alignment paths, which dynamically stretches or compresses sequences to achieve optimal alignment that preserves both local and global patterns, thus capturing nuanced stylistic patterns that other methods might miss.

Recent enhancements, such as Derivative Dynamic Time Warping (Keogh, et al. 2001) [7], favor the alignment process by utilizing information for derivatives, making DTW more effective in identifying subtle temporal patterns [5]. Similarly, Salvador & Chan (2007) proposed some enhancements that reduce DTW's computational complexity, enabling its application to large-scale datasets. With such improvements, DTW widens its applicability and reinforces its position as a cornerstone in sequence alignment tasks.

One of DTW's strengths is that it acts in a very robust manner with noisy or fragmented data such as informal texts or tweets, making it particularly effective when dealing with challenging datasets. It aligns signals along different temporal scales, allowing both local and global stylistic congruence to be analyzed in depth. Style divergence or similarity structured representations can be created.

By integrating these enhancements, DTW has become a flexible tool. Its flexibility and adaptability ensure its continued relevance within modern text analysis frameworks.

2.6. Anomaly Detection and Clustering

Anomaly detection and clustering are foundational techniques in authorship verification, addressing challenges such as noisy data, adversarial inputs, and stylistic variability.

Anomaly Detection focuses on finding outliers - data points that are very different from the general stylistic patterns in a dataset. In the case of authorship verification, these anomalies may represent either impostor texts or adversarial crafted inputs designed to mimic another author's style. Several techniques, such as the Isolation Forest (Liu, et al. 2008) [9], are widely used for outlier detection in high-dimensional text data, mostly in unsupervised settings.

Isolation Forest Algorithm

Isolation Forest performs the isolation of the observations by randomly choosing a feature and then randomly choosing a split value between the maximum and minimum values of that feature. Every data point is given an anomaly score that reflects how isolated it is within the random partitioning structure developed by the model. More precisely, the algorithm builds multiple, randomly generated decision trees, with the isolation being performed by recursively splitting the feature space. Indeed, points that are easier to isolate—which require fewer splits—tend to be anomalous and thus have higher anomaly scores.

Formally, the anomaly score $s(x)$ for a point x is derived by:

Path Length: Calculating the path length $h(x)$ from the root to the terminating node in each tree.

Average Path Length: Averaging $h(x)$ across T trees to obtain $\underline{h}(x)$.

Normalization: Converting $\underline{h}(x)$ into a score between [0, 1] by comparing it to the expected path length of a random observation, often approximated by a function $c(n)$ dependent on the dataset size n.

$$s(x) = 2^{-\frac{\underline{h}(x)}{c(n)}}$$

A score closer to 1 means that this instance is more likely to be an outlier, and the values closer to 0 are indicating that the point is more "normal." By isolating observations through random partitioning of the feature space, Isolation Forest is able to spot outliers with high efficiency in large datasets - a common scenario in authorship verification - guaranteeing that noisy or malicious data will not affect the result of the verification.

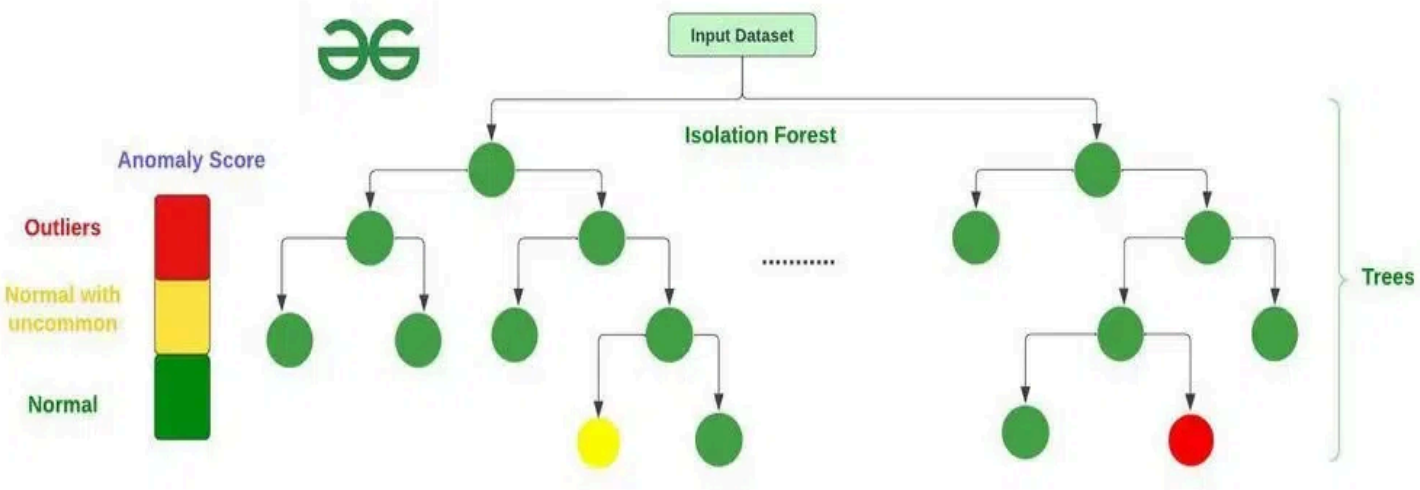


Figure 12. Shivi. (2023, January 12). What is Isolation Forest? GeeksforGeeks. Source

This technique is especially suited for high-dimensional data, which is common in authorship verification, ensuring that noisy or deceptive data does not interfere with the verification process.

Clustering

As we have been trying to highlight throughout this journal so far, adversarially designed inputs and noisy datasets are among the leading limitations of traditional methods. This holds true for traditional clustering techniques as well, despite being effective in the grouping of data points, integrating robust anomaly detection techniques like Isolation Forest to mitigate these limitations and ensure cleaner and more accurate clustering results.

Similarly, clustering reinforces anomaly detection by organizing the texts according to stylistic coherence. Together, these techniques enhance the system's robustness and interpretability, making them essential components of a modern authorship verification pipeline.

K-Medoids Clustering

The K-medoids algorithm is a robust clustering technique with the capability to deal with noise and outliers in high-dimensional data. Unlike K-means, which makes use of centroids as the representative of the cluster, which is sensitive to outliers, the K-medoids algorithm selects actual data points (medoids) as the cluster centers to minimize the sum of pairwise dissimilarities between points and their assigned medoids.

This is more robust to the influence of extreme values, which aligns well with the nature of anomaly scores.

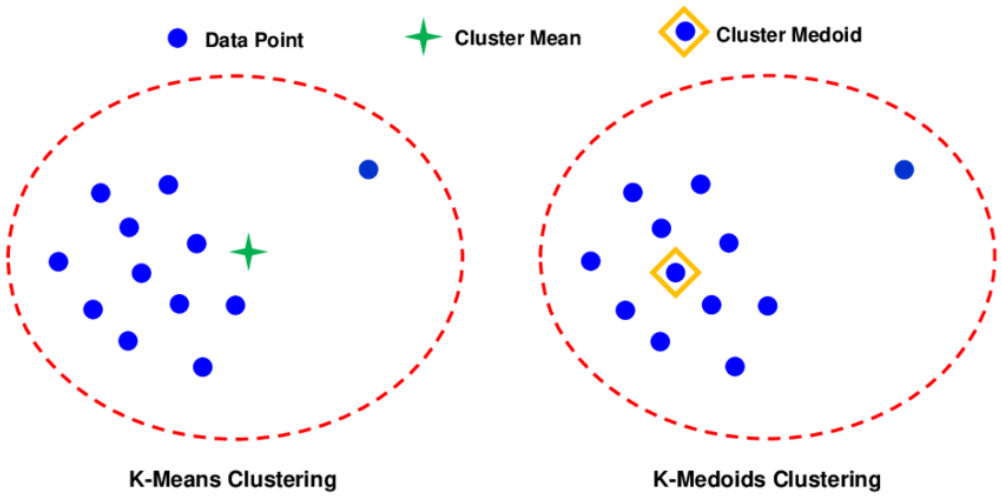


Figure 13. The graphical representation of the difference between the k-means and k-medoids clustering methods. Source

K-Medoids Algorithm

1. **Initialization**: Randomly select k points from the dataset as initial medoids.
2. **Assignment**: Assign each data point to the cluster represented by the nearest medoid, based on a dissimilarity measure.
3. **Update**: For each cluster, calculate the total dissimilarity for each point in the cluster to all other points in the cluster. Next, replace the medoid with the point that minimizes the total dissimilarity if such a replacement reduces the overall clustering cost.
4. **Iteration**: Repeat the assignment and update steps until the medoids stabilize or the clustering cost converges.
5. **Output**: The final clusters and their medoids.

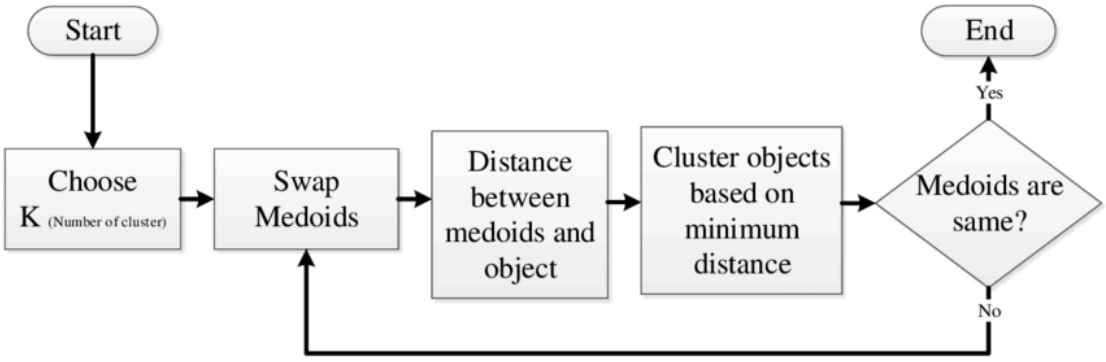


Figure 14. Flowchart of the K-medoids algorithm. Source

Application to Isolation Forest Anomaly Scores

In the context of authorship verification, anomaly scores generated by the Isolation Forest represent a single-dimensional dataset, where each score indicates the likelihood of a text being an outlier. K-medoids clusters these scores into stylistically coherent groups, effectively separating between the “normal” texts-which would have low anomaly score-from probably adversarial or outlier texts that would present high anomaly scores. K-medoids ensures that that the clustering is reliable even in the presence of noisy or deceptive inputs by aiming at minimizing dissimilarity around robust medoids.

3. Expected Achievements

Our capstone project is focused on improving the existing methodologies and addressing their limitations in authorship verification. The project seeks to develop a strong and sophisticated framework for authorship verification by integrating the Impostors Method into a Siamese network architecture with complementary pre-trained BERT embeddings, further refined through a CNN-BiLSTM hybrid network. Key limitations in the current traditional methods will be addressed.

The primary goal of this project is to provide a final solution to the Shakespeare Apocrypha controversy. The new architecture is expected to improve the capabilities of identifying stylistic patterns within longer texts with different complexities, further enhance adversarial training using the Deep Impostors Projection Method and outperform traditional approaches as well as CNN-based architectures.

Expected outcomes of the project are:

1. Develop a robust Siamese network framework that incorporates pre-trained BERT, refined through CNN-BiLSTM layers, embeddings to mitigate the semantic and syntactic shortcomings of previous approaches.
2. Demonstrate improved scalability and adaptability of the model to diverse datasets, including those made up of short, noisy, or domain-specific text corpora.
3. Possibility of a generalized model for authorship verification, and set new benchmarks for accuracy and robustness, applicable across domains such as digital forensics, plagiarism detection, and text analysis.

4. Research Approach

In this section, we present the structured methodology followed during the research of the proposed model for authorship verification. It outlines the systematic steps taken to address the challenges identified thus far, the model’s training process, inference

methodology, and architecture; the complete integration of the components & techniques detailed in the Background and Related Works– section 2.

4.1. Research Process

The first phase of our capstone project focuses on the study of advanced neural networks, transformers, and clustering algorithms that improve authorship verification. We began by going through the basic “authorship verification” problem, studying a presentation provided by our mentor, Zeev [12], about previous works such as traditional methods and the architectural framework. This pilot study has allowed us to realize the flaws of existing methods and architectures and encouraged us to go through many research papers that provided enhancements on current techniques, besides some that proposed new neural network architectures.

Next, we studied and identified the key features, dependencies, and patterns that were necessary to be extracted and retained. This analysis has been done in order to improve the accuracy of the verification and labeling process while adapting these findings to the requirements of the selected networks.

We then explored some of the efficient techniques to process and handle outputs of newly proposed components and networks. The efforts would head toward optimizing the decision-making process so that it could come up with better results with streamlined methodologies.

After that, we focused on studying optimizations and training techniques that could effectively train every component of the model. That included fine-tuning strategies for transformers, hybrid architecture improvement, and general improvement in the training process.

Finally, we synthesized all the results from our research in order to design the flow of the new proposed model and its components.

4.2. Approach

4.2.a. Proposed Model

Our proposed model for authorship verification is the culmination and integration of the various components and techniques we introduced in the Background and Related Works section– Section 2. This integration represents an advanced integration of neural network architectures, pre-trained transformers, and clustering algorithms. It addresses the limitations of traditional methods by combining the strengths of BERT, CNN-BiLSTM, Dynamic Time Warping (DTW), and Isolation Forest. The training process utilizes the “deep” variant of the Impostors Projection Method, effectively optimizing the core network weights and acting as an extrinsic authorship verification model.

The proposed model operates in a Siamese configuration, where the architecture ensures consistency and comparability across input pairs. The model begins by preprocessing the input texts, tokenizing them into smaller, manageable batches of up to 512 tokens each. These batches are then divided into small chunks/“tweets”, and are passed through two separate pre-trained BERT encoders that generate contextualized embeddings for each chunk. To simplify and aggregate the embeddings, mean pooling is applied, resulting in a fixed-size vector for each chunk. This ensures that semantic and syntactic relationships between tokens are captured effectively, providing a general representation adaptable for fine-tuning in authorship verification tasks.

The BERT-generated embeddings are then refined using a CNN-BiLSTM architecture. The CNN layers, Conv1D and max pooling, focus on extracting localized stylistic cues, such as n-gram patterns, while the BiLSTM layers incorporate long-range dependencies and stylistic evolution across the text. This combination produces enriched feature vectors that capture both local and global stylistic patterns, crucial for distinguishing between genuine and impostor text pairs. The feature vectors are further refined through a series of layers: dropout, fully-connected with ReLU activation and a softmax in the context of a fuzzy classification.

Next, the refined feature vectors are passed through a fully connected layer with ReLU, which allows the model to learn further transformations specific to the refined feature vectors of each chunk of text. The outputs of these fully connected layers are then compared using the Euclidean distance, producing a similarity score between the chunk pairs. At last, the similarity score is compared to a threshold value of $\frac{1}{2}$; if it’s greater, the chunk pair is labeled “1”, otherwise “0”, denoting that the chunk pair’s similarity is attributed more to impostor 1 or impostor 2. The label is used for loss calculation during training.

These chunk-level labels enable fine-grained analysis of stylistic patterns, which are later aggregated into batch-level scores, by calculating the mean value of each batch’s chunks scores.

To ensure robustness against adversarial inputs and stylistic noise, the aggregated batch scores are translated into structured signals. These signals are aligned using DTW by calculating a distance matrix, thus dynamically accounting for variations in sequence and structure between texts. This is highly valuable information that allows the model to detect and distinguish between the stylistic features of the text pairs. Afterwards, Isolation Forest is employed to identify anomalies within the signals by applying the algorithm to the distance matrix, where each anomaly score represents the likelihood that a specific alignment within the distance matrix exhibits unusual patterns that deviate from the typical alignment patterns learned during training. High anomaly scores suggest a potential “impostor” relationship between the text pairs. Finally, we use K-medoids, where the number of clusters is K=2, to cluster the anomaly scores in order to provide categorization of the texts into groups of interest.

Training Procedure

The training phase incorporates the Deep Impostors Projection Method, which introduces adversarial impostor pairs to enhance the model's robustness against mimicry. Let X be the number of impostor pairs from a given set, the training is performed for each impostor pair, where each tokenized text pair is processed iteratively, with embeddings generated for each chunk. Let m_1 and m_2 be the number of tokenized text pairs (batches) for impostor 1 and impostor 2 respectively. Let n_1 be the number of chunks per impostor 1 batch, and n_2 per impostor 2 batch. Therefore, the number of chunks in impostor 1's text is $m_1 \cdot n_1$ and $m_2 \cdot n_2$ in impostor 2's text.

We pass each chunk pair, i and j, of text 1 and text 2 respectively into a Siamese network, where each chunk is converted into a word embedding using BERT, producing a vector of size 768, resulting in a matrix of size $n_1 \cdot 768$ for impostor 1 and a matrix of size $n_2 \cdot 768$ for impostor 2. Each embedding vector is then passed through a CNN-BiLSTM network, and a fully connected layer with the ReLU activation. Finally, the two vectors are compared using Euclidean distance, and the chunk pair are labeled accordingly.

After completing the loop over all impostor chunk pairs, the network which has been trained provides a separating manifold (a decision boundary).

By the end of the procedure detailed above for all X impostor pairs, X trained networks (1 for each impostor pair) are stored for further use.

Procedure Summarization

Let Y be the number of texts in the tested collection, which have been pre-processed and divided.

Let m be the number of batches and n the number of chunks in each text.

The chunks of each of those Y texts are streamed down each one of the X trained networks, producing a label from each one which are used to fill the X corresponding labels matrices of sizes. The final result of this process is X matrices of size $m \, x \, n$ for each text, where each row in each matrix is filled with chunk labels belonging to a batch. Next, for each matrix from the networks and for each text, the mean value of each row is calculated in order to produce batch scores, which are then aggregated into a single list, effectively obtaining X signal representations for each text, one per trained network.

Afterwards, the time-series of all X signals of each text are compared using Dynamic Time Warping (DTW) to calculate a distance matrix of size $m\,x\,m$ by computing a distance score between aligned signals of each text. This process results in Y DTW matrices overall that serve as input to the Isolation Forest algorithm. The Isolation Forest detects whether the alignment between the signals of each text are anomalous. The output of this algorithm per matrix is a vector of anomaly scores of size m, resulting in a total Y vectors, which are later aggregated into a single matrix.

Finally, we employ the K-medoids algorithm to produce stylistically coherent clusters, where K = 2. Clustering reveals two groups allowing us to infer the classification of the text under consideration. Cluster 1 - Same author. Cluster 2 - Different authors (including pseudo-authors and impostors).

Optimizations and Hyperparameters

To enhance the model's performance, several optimizations were applied during training, including gradient clipping, dropout regularization, and dynamic learning rate scheduling. These techniques stabilized the training process, reduced overfitting, and ensured efficient convergence.

The model is trained and optimized using the binary cross-entropy (BCE) loss function. BCE is mathematically defined as:

$$L = \; -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot log(\hat{y}_i) \; + \; (1 - y_i) \cdot log(1 - \hat{y}_i)]$$

where:

- N: Number of samples.
- y_i: Ground truth label (0 or 1) for sample i.
- $\hat{y}_i$: Predicted similarity score (distance metric output) for sample i.

Essentially, BCE measures the difference between predicted probabilities and actual labels, by punishing incorrect predictions, more heavily the farther they are from the true label (0 or 1). It is used to fine-tune BERT and the Siamese architecture (including CNN and BiLSTM layers) by optimizing the model to produce probabilities that minimize the BCE loss for genuine and impostor pairs, ensuring accurate classification of authorship similarities.

Furthermore, the training process of the model leverages backpropagation using gradient descent as the core optimization technique. Backpropagation involves two primary steps: the forward pass and the backward pass. In the forward pass, input data (tokenized text pairs) is processed through the network, generating outputs (e.g., embeddings or binary labels) and calculating the loss by comparing these outputs with the ground truth. During the backward pass, gradients of the loss with respect to the

model's parameters (weights and biases) are computed using the chain rule of calculus. These gradients are then used in gradient descent, where the parameters are updated iteratively to minimize the loss function.

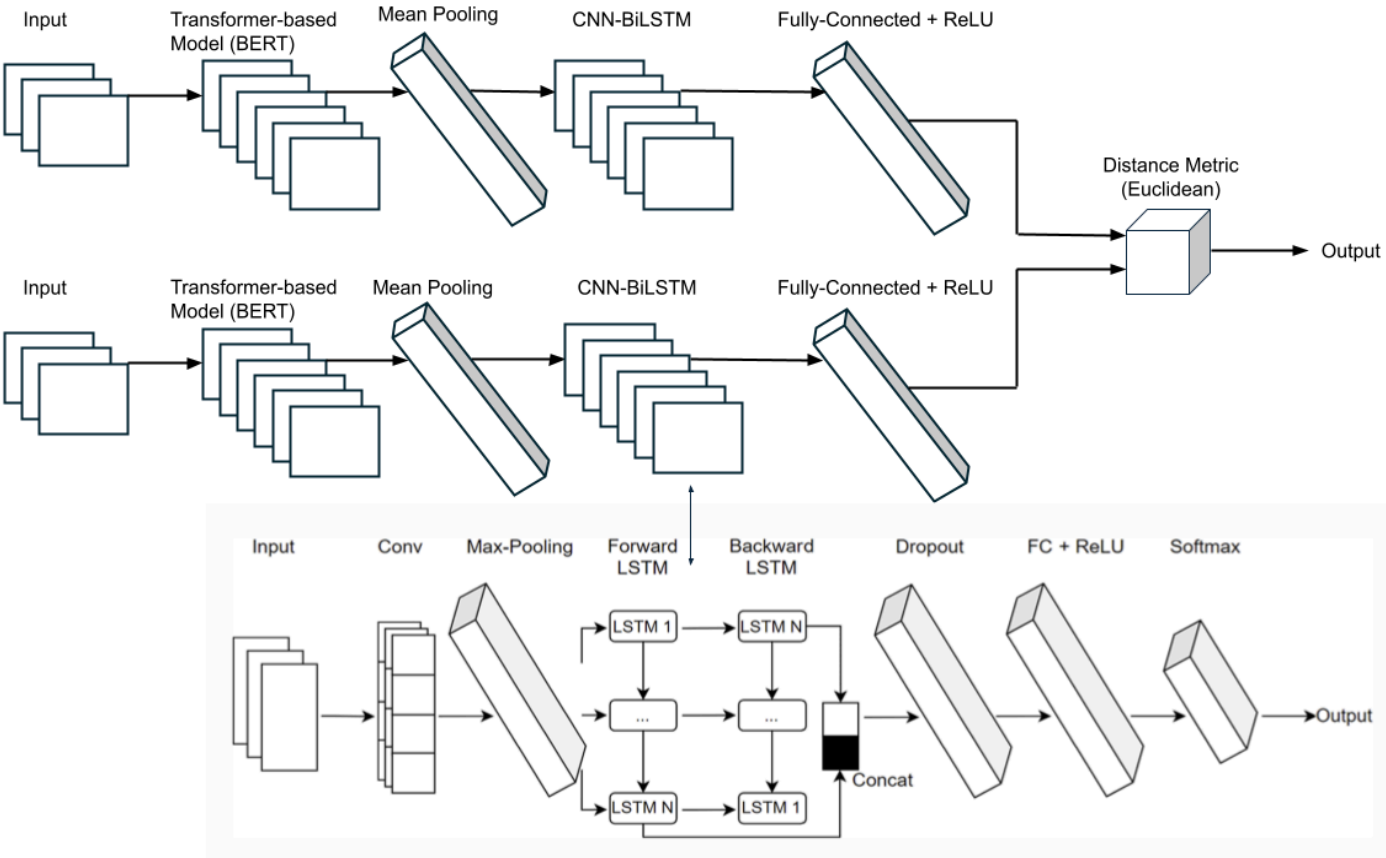
To stabilize this training process, gradient clipping is applied to address the issue of exploding gradients, which can occur in deep or recurrent architectures like BiLSTM. Exploding gradients cause excessively large updates to the model's parameters, destabilizing or halting the training process.

To further enhance training robustness, dropout regularization is employed. Dropout randomly deactivates a fraction of neurons during each forward pass by setting their outputs to zero with a probability p. This stochastic behavior prevents the network from over-relying on specific neurons and forces it to learn generalized patterns. This ensures the model is less prone to overfitting and performs better on unseen data.

Finally, dynamic learning rate scheduling is applied to adjust the step size during training. Initially, a higher learning rate allows the model to explore the parameter space broadly. As training progresses, the learning rate is reduced to fine-tune the parameter updates.

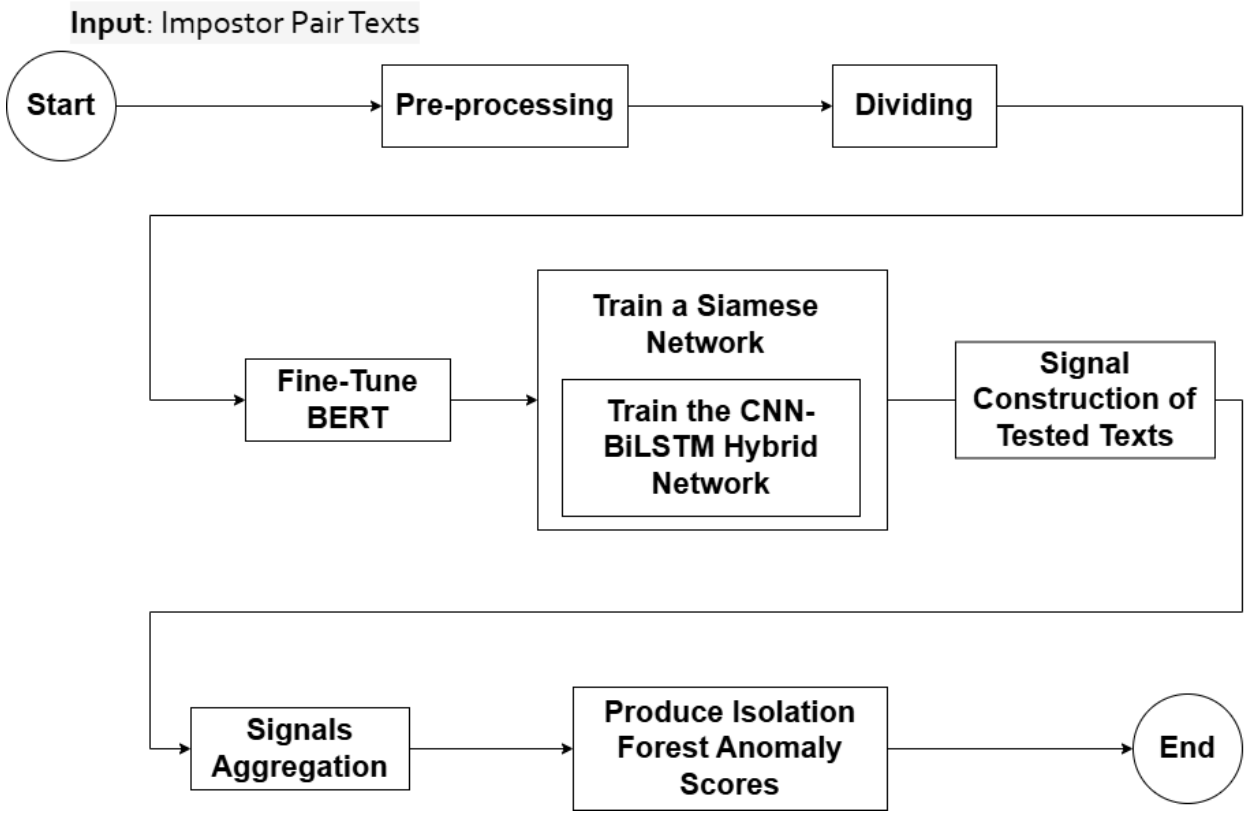
These techniques collectively address the challenges of training deep networks, ensuring numerical stability, preventing overfitting, and enabling efficient convergence to an optimal solution.

4.2.b. Model Architecture



4.2.c. Flow Chart

The following detailed flow outlines the steps for the new authorship verification model. Each step is described with inputs, processes, and outputs to ensure a transpicuous workflow.



1. **Start**
   - **Purpose:** Initiate the authorship verification process.
   - **Input:** A set of texts for evaluation– Impostor pairs and tested collection
   - **Output:** Text pairs ready for pre-processing.

---

2. **Preprocessing**
   - **Purpose:** Prepare raw text data for embedding and training.
   - **Steps:**
     1. Tokenization:
        - Use BERT's WordPiece tokenizer to split text into subword units.
        - Lemmatization and punctuation mark removal process to retrieve base words.
        - Truncate sequences to a maximum length of 512 tokens.
     2. Pair Labeling:
        - Assign "1" if the texts are from the same author, "0" otherwise (impostor pairs).
     3. Data Collection:
        - Split the impostor pairs into training, validation and test sets.

4. Data Structures:
    - Initialize chunks' label scores matrices.
- **Input:** Raw text pairs.
- **Output:** Tokenized text pairs and corresponding labels.

---

3. **Dividing**
- **Purpose:** Divide the tokenized text pairs into small chunks/"tweets" for easier processing.
- **Input:** Tokenized texts.
- **Output:** Structured text pairs ready for embedding.

---

4. **Fine-Tuning BERT**
- **Purpose:** Adapt BERT embeddings for stylistic and semantic differences relevant to authorship verification.
- **Steps:**
    1. Initialize Pre-Trained BERT:
        - Load a pre-trained BERT model.
        - Remove any classification head to retain the embedding layers.
    2. Define Siamese Setup:
        - Clone the BERT model for use in the Siamese architecture.
        - Each text in the set is processed through its own BERT encoder instance.
    3. Train BERT:
        - Pass tokenized text pairs through the BERT encoder to generate token-leveling embeddings ([n_tokens, 768]).
        - Apply mean pooling to reduce the embeddings to fixed-size pooled vectors ([1, 768] for each text).
    4. Optimization:
        - Use the AdamW optimizer with learning rate scheduling.
        - Fine-tune BERT for several epochs, monitoring validation loss for early stopping.
    5. Save Fine-Tuned BERT:
        - Save the trained weights for use in the Siamese network.
- **Input:** Tokenized text pairs and labels.
- **Output:** Fine-tuned BERT model capable of producing optimized embeddings for stylistic analysis.

---

5. **Train a Siamese Network**
- **Purpose:** Train the entire Siamese Network (BERT, BiLSTM and Dense layer) for impostors, and produce labelings.
- **Steps:**
    1. Generate Input Embeddings:
        - Pass each chunk through a fine-tuned BERT to obtain a vector embedding.
    2. Define CNN-BiLSTM:

- Initialize CNN-BiLSTM hybrid network.
- Initialize BiLSTM with an appropriate hidden size.
- Use a bidirectional setup to capture sequential dependencies in both directions.
- Stack additional BiLSTM layers if needed for hierarchical processing.

3. Train CNN-BiLSTM:
   - Pass vector embedding through the CNN-BiLSTM network.
   - CNN produces a feature vector.
   - Use the final hidden state of the BiLSTM to produce a refined feature vector.
4. Optimization:
   - Use AdamW optimizer and apply dropout regularization to prevent overfitting.
5. Save Trained CNN-BiLSTM:
   - Save the trained weights, along with the hybrid network, for use in the Siamese network.
6. Initialize Siamese Architecture:
   - Use two fine-tuned BERT encoders to process a chunk pair, producing vector embeddings for each chunk.
   - Pass each vector embedding through the trained CNN-BiLSTM.
   - Add a fully connected layer with ReLU activation, which produces a refined feature vector.
   - Calculate the Euclidean distance between the refined feature vectors from both Siamese networks, producing a similarity score.
   - Compare the similarity score to a threshold of 0.5 and assign the label-score $X \in \{0,\ 1\}$ accordingly, effectively labeling the chunk pair.
7. Train the Siamese Network:
   - Use binary cross-entropy loss function.
   - Optimize weights for BERT, BiLSTM, convolutional layers and the dense layer.
8. Optimization:
   - Use AdamW optimizer with carefully tuned learning rates for different components.
   - Apply gradient clipping to stabilize training.
9. Save the Siamese Network:
   - Save the entire trained model for inference.

- **Input:** Tokenized text pairs.
- **Output:** Fully trained Siamese network, that labels chunks.

---

6. **Signals Construction of Tested Texts**
   - **Purpose:** Fill tested texts chunks' labels matrix using the label-scores generated by the trained network.
   - **Steps:**

    - For each tested text i in the tested texts collection:
        1. Generate the label-score for each chunk using the trained network, resulting in a vector.
        2. Assign the vector to matrix $M[i]$.
- **Input:** Pre-processed and divided tested texts.
- **Output:** Chunks' labels matrix.

---

7. **Signals Aggregation**
    - **Purpose:** For each text in the tested texts, aggregate chunk labels to generate structured signals for each batch.
    - **Steps:**
        1. For each vector in the chunks' labels matrix M, calculate the mean value of each batch's chunks.
        2. Aggregate the batch scores and obtain signal representation of each tested text.
    - **Input:** Chunks' labels matrix.
    - **Output:** Signals for each tested text (structured batch-level signals).

---

8. **Produce Isolation Forest Anomaly Scores**
    - **Purpose:** Identify impostor texts or anomalous patterns using Isolation Forest and Dynamic Time Warping (DTW) similarity matrix.
    - **Steps:**
        1. Apply DTW to align the aggregated signals for each text pair.
        2. Generate a DTW similarity matrix.
        3. Train the Isolation Forest:
            a. Use DTW to compute a distance score between two texts.
            b. Combine the DTW distance and other features into a feature vector representing the pair
            c. Train the Isolation Forest on these feature vectors to detect anomalies.
        4. Generate anomaly scores for each text, indicating how likely it is an anomaly.
    - **Input:** Aggregated signals for text pairs.
    - **Output:** Isolation Forest scores.

---

9. **<u>Additional Training Flow Step - Post All Impostor Pairs Iterations</u>**
   Clustering of Anomaly Scores
    - **Purpose:** Clustering and Final Classification; Group texts into clusters based on stylistic similarity and classify them based on the Isolation Forest scores.
    - **Steps:**
        1. Filter out texts flagged as outliers by Isolation Forest.
        2. Apply clustering to the scores through the K-Medoids algorithm.
        3. Assign cluster labels to each text.
        4. Combine cluster labels with Isolation Forest scores.
        5. Assign texts flagged by Isolation Forest as anomalies.

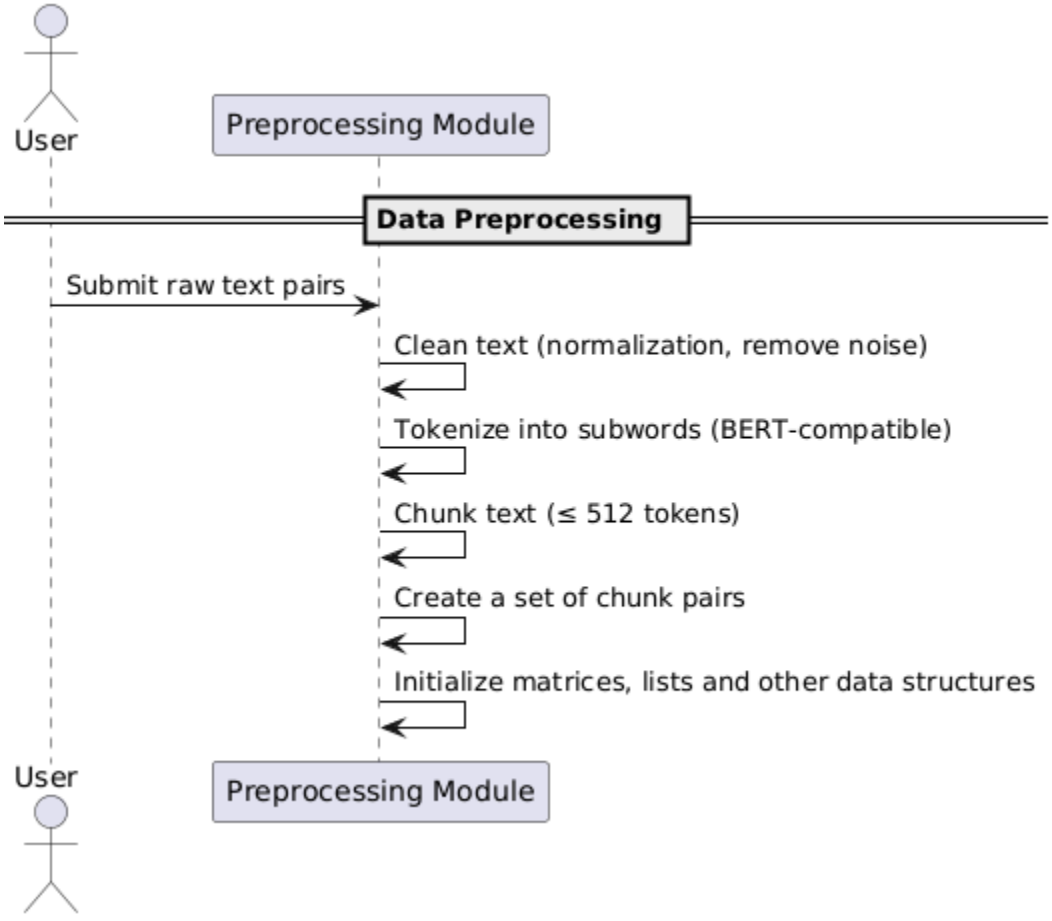
6. Group remaining texts into stylistic clusters based on clustering results (Note: Clustering leverages the separation of genuine and anomaly signals).

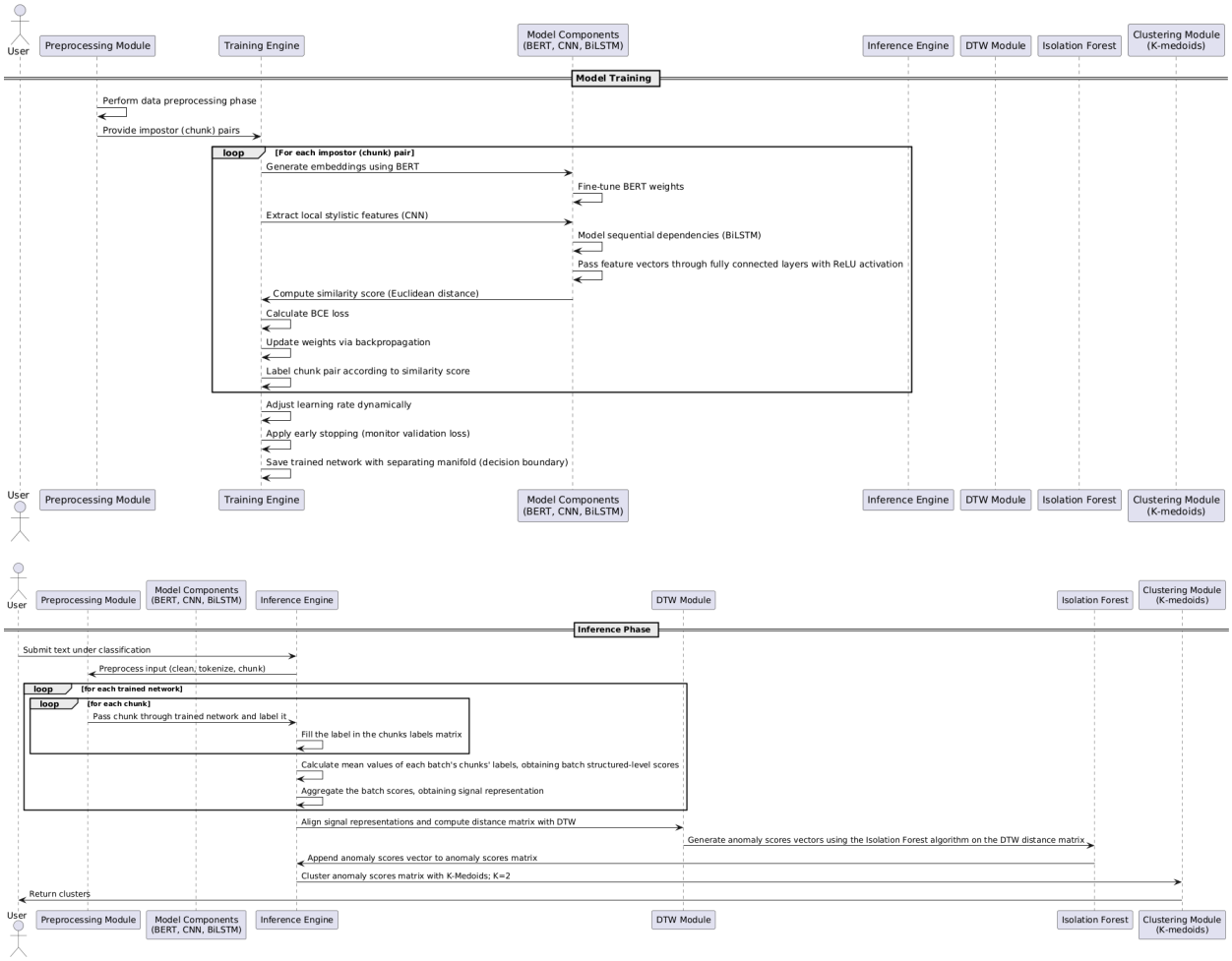
- **Input:** Isolation Forest scores.
- **Output:** Clusters of groups of interest.

---

10. **End**
    - **Purpose:** Complete training procedure, and utilize its outcome during inference.
    - **Output:** Clusters of groups of interest, anomaly scores and trained networks.

4.2.d. Sequence Diagrams





5. Evaluation and Testing Plan

The proposed evaluation and testing plans aim to systematically evaluate the effectiveness, efficiency, and robustness of the proposed model for authorship verification. It is structured around two main dimensions: performance metrics and stress testing in various model scenarios. Each step ensures that the model is able to fulfill its intended objectives — accuracy, scalability, and robustness under ideal and adversarial conditions. We conclude this plan with a list of comprehensive test cases.

5.1. Evaluation

5.1.a. Performance Metrics

Model performance will be evaluated with the help of the following measures:

- **Accuracy**: Measures the proportion of correctly classified text pairs (genuine vs. impostor).

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

- **Precision**: Assesses the correctness of positive (genuine) predictions.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall (Sensitivity)**: Evaluates the model's ability to detect genuine authorship cases.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- **F1-Score**: This score provides a balance between precision and recall.

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

5.1.b. Stress Testing

To ensure robustness, the model will be stress-tested primarily on adversarial scenarios:

1. Test the model's robustness to adversarial inputs, where an impostor tries to emulate the style of real authors.
2. Measure the effect on false positive and false negative rates.

5.1.c. Implementation and Iteration

Evaluation will be conducted in the following iterative manner:

1. **Optimization**: Refine hyperparameters in the light of preliminary results.
2. **Evaluation**: Evaluate the model on full-sized datasets and adversarial use cases.

5.1.d. Reporting

The results of the evaluation will be reported in the form of:

- Quantitative performance (metrics like accuracy, F1-score).
- Qualitative observations (model behavior under stress testing).

This evaluation and testing plan would ensure that the proposed model is tested effectively for its efficacy, scalability, and robustness, which would be a strong foundation for real-world applications in authorship verification.

5.2. Testing

The testing plan for our project is divided into three categories: Data Preparation, Model Training, and Network-Components Integration. Each module contains specific unit and functional tests to ensure correctness, reliability, and performance.

5.2.a. Data Preparation and Validation

Test	Function Tested	Input	Expected Result
1. Tokenization	Tokenize input texts into valid subword tokens	Example text: "Tokenization ensures each text fragment is split into valid subword tokens."	All tokens conform to BERT tokenizer (WordPiece) rules
2. Pairwise Creation	Generate text pairs (genuine and impostor pairs)	Dataset of texts from different authors, e.g., ["Text A by Author 1", "Text B by Author 2", "Text C by Author 1"]	Pairs correctly labeled and stored
3. Chunking	Divide texts into appropriate-sized chunks	A long text (e.g., 1000+ tokens): "This is a very long input text..." (add sufficient text so its length exceeds 1000 tokens for realistic chunking)	Chunk size $\leq$ 512 tokens
4. Dataset Integrity Validation	Ensure the dataset has not corrupt or missing entries	Dataset file with various text entries, some valid and some corrupted, e.g., ["Valid Text Entry 1", "", None, "Valid Text Entry 4"]	All rows contain valid text data
5. Adversarial Cases	Inject mimicry-based adversarial examples for testing	Dataset with adversarial examples, e.g., pairs like ("Text A", "Text A rewritten with mimicry from a different author")	Model identifies adversarial different authors.

5.2.b. Model Training Tests

Test	Function Tested	Input	Expected Result
1. Siamese Network	Siamese architecture consistency during forward passes	Pair of texts: ("Text A: Example input text", "Text B: Another example text")	Embeddings generated for both inputs
2. Loss Function	Binary cross-entropy calculation	Model predictions and ground truth labels, e.g. predictions = [0.1, 0.9], labels = [0, 1]	Loss values decrease as epochs progress
3. Gradient Propagation	Gradients flow through BERT, CNN, BiLSTM layers	Model with architecture including BERT, CNN, and BiLSTM layers, input sequence: "Sample text input for gradient testing."	All layers update weights without errors
4. Isolation Forest	Train Isolation Forest on DTW-aligned signal matrices (unsupervised)	Dynamic Time Warping (DTW)-aligned signal matrices from time-series data, e.g., [[1.2, 0.5, 1.1], [0.8, 0.9, 1.0], ...]	Models anomaly detection without overfitting
5. Overfitting Check	Monitor training and validation loss during training	Training and validation datasets with similar distributions, e.g., Dataset 1 (training), Dataset 2 (validation), with labels and features aligned appropriately.	Avoid significant divergence between losses
6. Early Stopping	Ensure training stops when validation performance hits a plateau	Model training data with early-stopping parameters, e.g., patience = 5 epochs, validation set performance plateauing, such as validation_loss = [0.5, 0.5, 0.5].	Stops without exceeding patience threshold

5.2.c Network-Components Integration

Test	Function Tested	Input	Expected Result
1. Inference Process	End-to-end process with trained model	Unseen test text or signals, e.g., "This is an example of unseen input for inference testing."	Valid predictions for unseen test data
2. Signal Aggregation	Aggregating chunk-level scores into batch-level structured signal	Chunk-level scores from model predictions, e.g., scores = [0.85, 0.92, 0.75] for different chunks in a batch.	Batch scores computed correctly
3. DTW Alignment	Perform DTW alignment on batch signals	Batch of time-series signal matrices, e.g., [[1.1, 0.8, 1.0], [0.7, 0.9, 1.2]], where rows represent signals from different sequences.	Cumulative distance and correct DTW distance matrix
4. Anomaly Detection	Isolation Forest flags anomalies in aligned signals	Aligned signals or DTW-aligned matrix with normal and anomalous patterns, e.g., [[1.2, 0.8, 0.9], [10.0, 9.8, 9.7]] (second signal represents anomaly).	Correctly flags outliers without overfitting

6. Conclusion

We introduced the topic of authorship verification and explored a hybrid approach combining BERT embeddings, CNN-BiLSTM layers, and advanced signal processing

techniques. The model leverages the Impostor Projection Method, Dynamic Time Warping, Isolation Forest, and K-Medoids to improve robustness against adversarial mimicry. We detailed the architecture with examples, focusing on the labeling of individual chunks and aggregation into signals for better interpretability and decision-making.

In the next phase, Phase B, we will instantiate the model as a modular software solution, allowing us to test and analyze in depth according to the plan we have outlined in Section 5.

7. References

[1] Tyo, J., Dhingra, B., & Lipton, Z. C. (2021). Siamese BERT for authorship verification. Conference and Labs of the Evaluation Forum.

[2] Shrestha, P., Sierra, S., González, F., Montes, M., Rosso, P., & Solorio, T. (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 (EACL), Volume 2: Short Papers (pp. 669–674). Valencia, Spain.

[3] Almutairi, A., Kang, B., & Al Hashimy, N. (2024). BiBERT-AV: Enhancing authorship verification through Siamese networks with pre-trained BERT and Bi-LSTM. In G. Wang, H. Wang, G. Min, N. Georgalas, & W. Meng (Eds.), Ubiquitous Security. UbiSec 2023. Communications in Computer and Information Science (Vol. 2034). Singapore: Springer.

[4] Seidman, D. (2013). Authorship verification using the impostors method. Journal of Digital Forensics, Security and Law, 8(2), 33–44.

[5] Potha, N., & Stamatatos, E. (2017). An improved impostors method for authorship. International Journal of Digital Crime and Forensics, 9(3), 1–17.

[6] Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing.

[7] Keogh, E., & Pazzani, M. (2001). Derivative dynamic time warping. In Proceedings of the First SIAM International Conference on Data Mining.

[8] Salvador, S., & Chan, P. (2007). Toward accurate dynamic time warping in linear time and space. Intelligent Data Analysis.

[9] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM'08).

[10] Volkovich, Z. (2020). A short-patterning of the texts attributed to Al Ghazali: A 'Twitter look' at the problem. Mathematics, 8. https://doi.org/10.3390/math8111937

[11] Park, H.-S., & Jun, C.-H. (2009). A simple and fast algorithm for K-medoids clustering. Expert Systems with Applications, 36(2), 3336–3341.

[12] Volkovich, Z. (2022, June). Text classification using imposter projections method. 12th International Symposium on Foundations of Information and Knowledge Systems, Helsinki.

8. Appendix A: Hyperparameters of the Proposed Model

Table A1. Hyperparameters used in training and optimizing the proposed model.

	Component	Hyper-Parameter	Value
1	BERT Embedding Layer	Maximum Sequence Length	512 tokens
2		Batch Size for Fine-Tuning	16 or 32
3		Learning Rate for Fine-Tuning	1e-5 to 3e-5
4	CNN Layer	Number of Filters	256
5		Kernel Size	3, 5 or 7
6		Stride	1
7		Padding	Same
8		Activation Function	ReLU
9	BiLSTM Layer	Hidden Units	128 or 256
10		Dropout Rate	0.2 to 0.5
11		Number of Layers	300
12		Dropout Rate	0.5
13	Optimization Parameters	Optimizer	AdamW
14		Initial Learning Rate	1.00E-04
15		Learning Rate Decay Factor	0.1
16		Gradient Clipping Threshold	1.0 or 5.0
17	Isolation Forest	Number of Trees	100 or 256
18	General Training Parameters	Batch Size	32 or 64
19		Epochs	10 to 20 (with early stopping)
20		Early Stopping Patience	5 epochs
21		Dropout Rate	0.2 to 0.5