Software Engineering Department

Capstone Project Phase A – 61998

Analysis of Authorship in Arabic sources

25-2-R-7

https://github.com/mayarsaleh4/CapstoneProject.git

Mayar Salih	Mayar.Salih@e.braude.ac.il
Bolos Khoury	Bolos.Khoury@e.braude.ac.il

Supervisors:

Dr. Renata Avros

Prof. Zeev Volkovich

Abstract

Verifying authorship in literary texts has long been a challenging task, particularly in languages such as Arabic, which are rich in stylistic variation and historical depth. This study presents a novel application of the Deep Impostors methodology to the problem of authorship verification in Arabic literature. Inspired by the work of Volkovich and Avros on Shakespearean texts, we adapted this approach to the unique characteristics of classical Arabic, leveraging deep learning techniques to model the stylistic fingerprints of authors. The methodology involves generating embeddings from selected Arabic texts and applying impostor-based decision-making mechanisms to determine the authenticity of authorship. The project aims to distinguish between genuine and disputed works attributed to a given author, providing a data-driven approach to a problem that has traditionally been addressed through philological analyses.

Keywords: Authorship Attribution, Arabic Text Analysis, Stylistic Analysis, Machine Learning, AraBERT.

Table of Contents

1. Introduction

Authorship attribution aims to identify who wrote a given text based on writing style. In this study, we focused on Arabic texts, which pose unique challenges owing to their linguistic complexity and limited tools. We propose using a modified version of the Deep Impostor method to improve authorship analysis in Arabic.

1.1 Authorship Attribution

In recent years, authorship attribution has become a growing area of research within the field of Natural Language Processing (NLP). The goal of authorship attribution is to identify or verify the author of a given text based on linguistic and stylistic features. Although many advanced methods exist for authorship verification in English and other widely studied languages, there is still a significant gap in resources and research regarding Arabic texts, especially classical and literary works. Arabic, with its rich linguistic structure and historical significance, presents unique challenges for computational analysis. Many classical Arabic texts have uncertain authorship or lack proper documentation, which creates difficulties for scholars, historians, and researchers seeking to understand the origins and authenticity of these works. Therefore, developing reliable tools for Arabic authorship attribution is both a scholarly necessity and a technological challenge.

1.2 The Problem

Currently, there is a lack of modern algorithmic tools tailored for Arabic text verification. Most existing solutions are either language-dependent (designed for English or European languages) or fail to capture Arabic's the unique linguistic characteristics. Additionally, many Arabic works, particularly those from historical or religious contexts, are not easily verifiable using conventional methods. This creates a gap in both academic research and digital humanities.
Our project addresses the specific challenges of Arabic authorship attribution by applying and modifying a novel technique known as the Deep Impostor method, which was recently introduced by our supervisors Volkovich and Avros. This method uses deep learning and a comparative approach with "impostor" texts to evaluate authorship credibility.

1.3 Review of Existing Solutions

Existing solutions to authorship attribution typically fall into categories such as statistical feature analysis, machine learning classifiers, and, more recently, neural network-based models. However, most of these methods were developed for English and lack adaptability to the syntactic and semantic characteristics of Arabic. Even when applied to Arabic texts, they often fail to capture deeper stylistic nuances or require extensive manual-feature engineering.

The Deep Impostor method introduces a fresh perspective by using neural networks to compare segments of known and unknown texts, transforming them into signals and measuring their similarities using distance metrics. This method has shown promising results in English and now provides a foundation for adaptation to Arabic.

1.4 Book Structure

Chapter 1 presents an overview of authorship attribution, the challenges of verifying authorship in Arabic texts, and the proposed computational framework of this study. Chapter 2 reviews related work, including key concepts and prior studies. Section 3 outlines the project's expected outcomes and success criteria. Section 4 details the research process, including preprocessing, embedding generation, and model architecture. Chapter 5 discusses the evaluation plan and validation methods used for the framework.

2. Background and Related Work

2.1 Related Work

Authorship attribution has been widely explored in Arabic texts using both classical stylometry and modern deep-learning methods. AlZahrani and Al-Yahya [1] used ARBERT and AraELECTRA on Islamic legal texts, reaching up to 96% accuracy. Alqurashi et al. [2] applied BERT with topic modeling to Arabic poetry, achieving F1 scores between 0.97–1.0. Howedi et al. [3] employed stylometric features with KNN for historical Arabic texts, yielding 90% accuracy despite limited data. Al-Khatib et al. [4] demonstrated the effectiveness of Naïve Bayes on stylistic cues, while Hajja et al. [5] used POS tags and rare words for attributing Arabic articles. Most of these studies address closed-set attribution. In contrast, Volkovich and Avros [6] introduced the Deep Impostors framework, combining BERT/CNN with impostor corpora and isolation forests for open-set authorship verification, successfully identifying pseudo-Shakespearean works and achieving consistent clustering over 127 classifier iterations. Our work extends this framework to classical Arabic literature, adapting it to the unique structure and stylistic diversity of the language.

2.2 Background

This section outlines the key concepts and techniques used in our authorship verification approach, including word embedding, BERT, binary classifiers, the Deep Impostors method, DTW signal comparison, Isolation Forest, and clustering.

2.2.1 Word Embedding and Vectorization

Word embedding transforms textual data into continuous vector representations in a multidimensional space, where semantically similar words are positioned close together. Traditional models, such as Word2Vec [11] and GloVe [16], rely on co-occurrence statistics to generate embeddings based on the distributional hypothesis, which states that words appearing in similar contexts tend to have similar meanings.
Formally, given a vocabulary $V = \{w_1, w_2, \dots, w_n\}$ word embeddings map each word wi∈V to a dense vector $v_i \in R^d$, where d is the embedding dimension. In the Skip-gram model of Word2Vec, the objective is to maximize the conditional probability of context words $c \in C$ given a center word w:

$$max \sum_{w \in V} \sum_{c \in context(w)} \log P(\frac{c}{w})$$

$$P\left(\frac{c}{w}\right) = \frac{\exp\,(v_c^T v_w)}{\sum_{j \in V} \exp\,(v_j^T v_w)}$$

This technique effectively captures semantic regularities, enabling efficient downstream processing in deep learning architectures [21].
For Arabic, context-sensitive models such as AraBERT, CAMeLBERT, and Arabic FastText further refine embeddings by incorporating morphological richness and right-to-left sentence structure [12].
In the context of Arabic, authorship verification is further complicated by the morphological complexity, diacritic variability, and syntactic flexibility of the language. These linguistic features increase ambiguity, making context-aware models like BERT and clustering-based approaches crucial for accurate authorship discrimination [23]. Models such as AraBERT and CAMeLBERT, pre-trained on Arabic corpora, are leveraged to capture these nuances better.

2.2.2 Bidirectional Encoder Representations from Transformers (BERT)

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art language representation model introduced by Devlin et al. (2018), designed to understand the context of a word based on all of its surroundings both to the left and right [15]. Unlike earlier models that read text sequentially (left-to-right or right-to-left), BERT uses a bidirectional training mechanism, which significantly improves contextual understanding.
Architecture Overview:
BERT is based on the Transformer encoder architecture introduced by Vaswani et al. (2017), which relies entirely on self-attention mechanisms instead of recurrent layers [18]. Given a sequence of tokens $X = (x_1, x_2 \dots, x_n)$, BERT transforms it into contextual embeddings $H = (h_1, h_2 \dots, h_n)$, where each $h_i \in R^d$ captures the full bidirectional context of token x_i.
Each encoder layer includes:

- Multi-head self-attention:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$where\ Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$are\ the\ query, key, and\ value\ matrice, respectively, and\ d_k is\ the\ dimension\ of\ the\ key\ vectors$

- Feed-forward network:

$$FNN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$$

Each attention head focuses on different parts of the sentence, and outputs are concatenated and linearly projected back to the model dimension.
Positional Encoding
Transformers lack inherent awareness of sequence order. To overcome this, BERT incorporates positional encodings to retain word order information. The positional encoding vector for position pospos and dimension i is defined as:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

These encodings are added to the input embeddings to inform the model of token positions.

Training Objectives

BERT is pretrained using two self-supervised tasks:

1. Masked Language Modeling (MLM):
    - It randomly masks 15% of input tokens and trains the model to predict them.
    - Given input $X = (x_1, x_2 \dots, x_n)$, the model learns to predict x_i given all other tokens:

$$L_{MLM} = -\sum_{i \in M} \log P(x_i | X_{\setminus i})$$

$$where\ M \subset \{1, \dots, n\} is\ the\ set\ of\ masked\ positions.$$

2. Next Sentence Prediction (NSP)[15]:
    - Given a pair of sentences A and B, the model learns to predict whether B logically follows A.
    - Binary classification loss:

$$L_{NSP} = -\log P(IsNext|A, B)$$

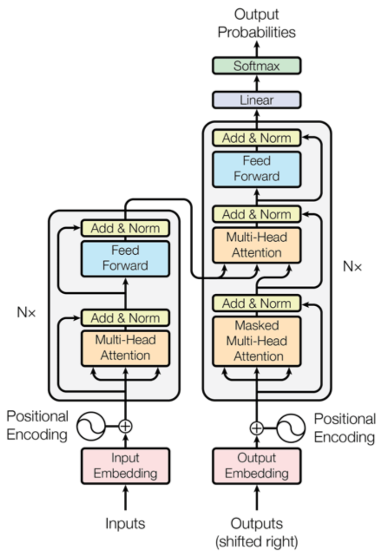

Figure 1. BERT Algorithm. Adapted from *All You Need to Know About BERT Algorithm*

Fine-Tuning:

After pretraining, BERT can be fine-tuned for specific tasks by adding minimal, task-specific layers. For instance, in the Deep Impostors method, BERT is fine-tuned as a binary classifier that distinguishes batches of impostor texts. Fine-tuning slightly adjusts the pre-trained weights to fit the new classification objective, benefiting from BERT's deep linguistic knowledge.

Relevance to Authorship Verification

In the context of authorship attribution, BERT serves as a powerful feature extractor and classifier. Its ability to capture contextual nuances and stylistic differences across batches makes it highly suitable for determining authorship patterns when comparing segments of candidate and texts.

2.2.3 Binary Classifiers

A binary classifier is a predictive model trained to assign inputs to one of two possible classes, typically labeled 0 and 1. In the context of authorship attribution using the Deep Impostors method, the classifier's task is to distinguish between writing styles of two impostor sources or to separate stylistic signals between candidate and non-candidate texts.

Mathematical Formulation

Let x $\in R^d$ be the feature vector representing a text batch (e.g., a word embedding or aggregated vector). The binary classifier learns a function:

$$f : R^d \longrightarrow \{0,1\}$$

During training, it minimizes a loss function such as binary cross-entropy:

$$L(y, \hat{y}) = -ylog(\hat{y}) - (1 - y)\log(1 - \hat{y})$$

Where:

- $y \in \{0,1\}$ is the true label
- $\hat{y} = \sigma(w^T x + b)$ is the predicted probability via a sigmoid activation
- $\sigma(z) = \frac{1}{1+e^{-2}}$

Classical Binary Classifiers [22]:

Classical models used in binary classification include the following:

- Logistic Regression: A linear model that outputs probabilities.
- Support Vector Machines (SVMs): Finds a hyperplane that maximizes the margin between classes.
- Decision trees / random forests: Hierarchical decision rules are used to partition the input space.

While effective for structured, low-dimensional data, these models are limited in their ability to capture complex patterns in high-dimensional text embeddings.

Neural Binary Classifiers [15]:

Deep neural classifiers overcome these limitations through multiple nonlinear transformations:

- Feedforward Neural Network (FNN):
$$\hat{y} = \sigma(W_2 \cdot ReLU(W_1 \cdot x + b_1) + b_2)$$
- Convolutional Neural Network (CNN): Applies filters to capture local n-gram features within text embeddings.
- Bidirectional LSTM: Captures sequential patterns by processing the input both forward and backward.
- Fine-tuned BERT: Learns contextual representations and classifies based on sentence-level embeddings (e.g., the [CLS] token vector).

Use in Deep Impostors Framework:

In the Deep Impostors methodology, the binary classifier is trained to distinguish between batches of two impostor documents Z_i and Z_j. After training, it is used to classify batches of unknown text. The classifier's decisions (0 or 1) are aggregated into a stylometric signal representing the text's similarity to each impostor. This role makes the classifier central to the transforming of textual patterns into analyzable time-series signals.

2.2.4 Deep Impostors Method

The Deep Impostors Method [13] enhances classical impostor-based verification by combining external style references with neural-network-based classification. The method operates by:

1. Dividing each text D into batches of L words, $D = \{b_1, b, \ldots, b_T\}$.
2. Selecting pairs of impostor corpora Z_i, Z_j, and training a binary classifier $S_{i,j}$ to distinguish between them.
3. Applying $S_{i,j}$ to classify each batch in a tested document D_k, yielding a binary sequence $\{0,1,\ldots,1\}$ indicating similarity to Z_i or Z_j.
4. Aggregating consecutive batch results into chunks of size k, and averaging to obtain real-valued signal components:

$$signal_t = \frac{1}{k} \sum_{l=(t-1)k+1}^{tk} s_l \;\; for\; t = 1,2,\ldots,\left\lfloor \frac{T}{k} \right\rfloor$$

This generates a stylometric signal representation of the document that reflects how its style aligns with each pair of impostors across the text. The use of deep classifiers, such as CNNs or fine-tuned BERT models, improves the sensitivity to subtle stylistic patterns.

2.2.5 Signal Comparison Using Dynamic Time Warping (DTW)

Once documents are converted into stylometric signals, the next step is to compare their temporal stylistic behavior using Dynamic Time Warping (DTW). DTW is a robust technique for comparing time series data that may be misaligned in time [19] or have variable lengths.
Given two signal sequences $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, DTW finds an optimal alignment path π that minimizes the total distance:

$$DTW(A,B) = min_{\pi}\left(\sum_{(i,j)\in\pi} d(a_i, b_j) \right)$$

Where $d(a_i, b_j)$ is typically the squared Euclidean distance: $d(a_i, b_j) = (a_i - b_j)^2$.
The optimal path π satisfies monotonicity and continuity constraints and is computed using dynamic programming with the recurrence:

$$D(i,j) = d(a_i, b_j) + min\begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

DTW enables the model to compare the stylistic structure of two documents despite local variations in writing pace or section boundaries.

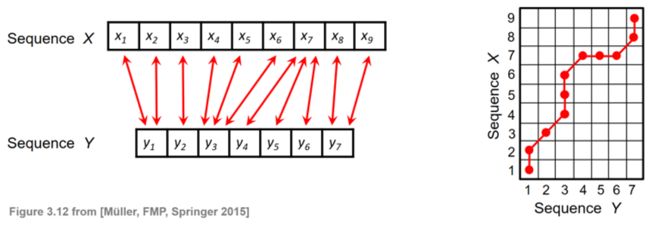

Figure 2. DTW Basic idea. *Dynamic Time Warping (DTW)*

2.2.6 Isolation Forest for Outlier Detection

The Isolation Forest algorithm is employed to identify documents that stylistically deviate from the rest of the corpus, serving as a method for authorship verification or anomaly detection. Unlike density-based methods, the Isolation Forest isolates anomalies using randomly constructed binary trees[17].

Each data point x_i is recursively partitioned until it is isolated. The path length $h(x_i)$ is the number of splits required to isolate x_i. The average path length is normalized using:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

$$where\ H(i) \approx \ln(i) + \gamma, \qquad \gamma \approx 0.5772\ (Euler - Mascheroni\ constant)$$

The anomaly score is computed as follows:

$$s(x_i, n) = 2^{-\frac{h(x_i)}{c(n)}}$$

Scores close to 1 indicate strong outliers, whereas scores near 0.5 suggest normal data points. This technique allows the detection of non-authentic or suspiciously different texts in an unsupervised and scalable way.

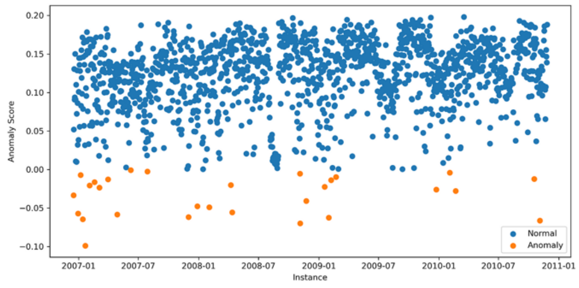

Figure 3. Isolation Forest Results Example. Adapted from *Isolation Forest Guide: Explanation*

2.2.7 Clustering

Clustering is an unsupervised machine learning technique that groups data points based on shared characteristics or similarities in the feature space [14]. In the context of authorship verification using the deep impostors method, clustering is applied to the final anomaly score vectors generated for each tested document. The objective is to distinguish between documents that follow the dominant writing style (presumably genuine) and those that deviate from it (potentially authored by someone else).

Let $D = \{d_1, d_2, \dots, d_n\}$ be a set of documents represented by their respective anomaly score vectors $s_i \in R^k$, where k is the number of impostor-based iterations. Clustering aims to partition D into K clusters $C_1, C_2, \dots, C_K$ such that:

$$\sum_{i=1}^{K}\sum_{s\in C_i}\|s-\mu_i\|^2$$

is minimized, where μ_i is the centroid (or medoid) of cluster C_i.
Common techniques include the following:

- K-Means: Partitions data by minimizing within-cluster variance using the Euclidean distance.
- K-Medoids: Similar to K-Means but uses actual data points as centers (robust to outliers).
- Hierarchical Clustering: Builds a tree (dendrogram) of nested clusters based on distance linkage.

In the Deep Impostors pipeline, a small cluster typically contains texts with anomaly scores that differ significantly from the majority, suggesting potential misattribution or stylistic inconsistency. The larger cluster is interpreted as representing the dominant authorial style. Thus, this method facilitates the identification of outlier texts without prior labeling, aiding in unsupervised authorship analysis.

3. Expected Achievements

This section outlines the expected achievements of the project and anticipated outcomes, providing an overview of the goals and measures of success.

3.1 Objectives and Capabilities

The primary objective of this project is to develop a computational framework capable of verifying the authorship of Arabic texts, particularly classical or historical writings, by applying the Deep Impostors methodology. The system integrates a combination of preprocessing techniques, word embedding, deep learning models, and anomaly detection algorithms. Its capabilities will include the ability to:

- Process Arabic texts from different time periods and authors. For example, texts by Al-Jahiz (e.g., *Kitab al-Hayawan*), Ibn Qutaybah (e.g., *Adab al-Katib*), Ibn Hazm (e.g., *Tawq al-Hamamah*), and Al-Ghazali (e.g., *Ihya' Ulum al-Din*) can be analyzed.
- Vectorize texts and extract stylometric features that capture authorial style.
- Apply segment-based comparisons using signal analysis and clustering.
- Output authorship verification decisions.

This system is designed to support authorship attribution tasks across diverse Arabic sources, with a modular structure that allows adaptation to new datasets or additional authors. The framework can be extended to other well-known classical corpora such as the works of Al-Mutanabbi**,** Ibn Khaldun, or anonymous texts suspected to belong to specific authors.

3.2 Expected Outcomes

The expected outcomes of the project include both technical deliverables and research contributions:

- A complete authorship verification system capable of analyzing Arabic texts and returning binary decisions (authentic/not authentic).
- An implementation of the Deep Impostors approach adapted for the Arabic language, including preprocessing tools, embedding pipelines, and DTW-based comparison modules.
- Visualization tools to support internal analysis such as signal similarity plots, cluster distributions, or distance heatmaps.
- A structured Arabic dataset labeled by author, with appropriate preprocessing and segmentation applied.
- Empirical results that demonstrate the effectiveness of the approach, benchmarked using quantitative evaluation metrics.

The project will also contribute insights into the challenges of applying computational authorship verification to Arabic texts, such as linguistic variation, orthographic complexity, and lack of standardized formatting.

3.3 Success Criteria

Success for this project will be determined by its ability to reliably differentiate between authentic texts written by selected Arabic authors and those misattributed to them within the evaluation set. A key benchmark will be the successful grouping of genuine works under a coherent stylistic profile while isolating texts of uncertain or false attribution into separate clusters. This distinction should be grounded in measurable linguistic and stylistic features that are consistent across an author's known corpus. The project also aims to validate the authorship of at least one hallmark work per author such as a widely accepted classical text by confirming its alignment with that author’s verified writing style. This will illustrate the framework’s strength in capturing deep stylistic signatures and ensuring trustworthy authorship verification in Arabic literary studies.

4. Research Process

This research adapts the Deep Impostors methodology [13] for authorship verification to classical Arabic texts, where stylistic markers are complex, and orthographic variability is high. The approach transforms authorship into a signal comparison problem: textual embeddings are converted into vector signals, compared via time-series analysis (DTW), and evaluated with anomaly detection (Isolation Forest). This structure balances deep learning's representational power with interpretable statistical methods.

4.1 Workflow Diagram

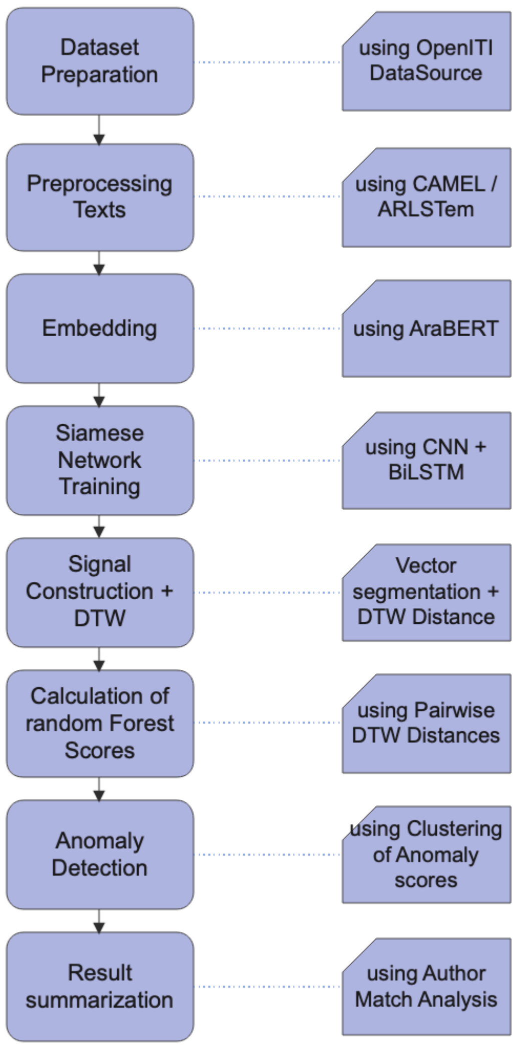

Figure 4. WorkFlow Diagram of the proposed algorithm.

4.2 Detailed Workflow

Step 1 : Dataset Preparation

In this stage, our primary objective is to structure a comprehensive and well-balanced corpus that supports both supervised and unsupervised evaluation methods for authorship verification. To achieve this, we will curate a large dataset of Arabic texts, carefully selected to represent the diversity and stylistic richness of classical and literary Arabic. The process includes the following steps:

- Corpus Collection:
  We will assemble a broad collection of Arabic texts from well-established digital repositories such as OpenITI (Open Islamicate Texts Initiative) [7] and Al-Maktaba al-Shamilah [8], which contain thousands of verified historical and religious Arabic texts. These sources offer access to classical authors such as Al-Ghazali, Ibn Taymiyyah, and others across various genres.
- Dataset Division:
  The curated corpus will be divided into two key subsets:
  - Target Set:
    This includes a collection of texts of one known Arabic author. These texts are not part of the training process and are used to test authorship verification capabilities.
  - Impostor Set:
    This set comprises texts by other authors from similar historical or literary contexts. It may include writings in related subject areas (e.g., theology, philosophy) as well as contrasting ones (e.g., poetry, law), providing necessary stylistic contrast for the system to learn meaningful distinctions.
- Stylistic Balance:
  Special attention will be given to balancing document length, genre, and vocabulary complexity between the target and impostor sets to prevent skewed model learning based on non-stylistic clues.
- Research Utility:
  This corpus structure allows for realistic authorship verification experiments. The model is trained not only to identify similarity but also to discriminate between close yet distinct styles mirroring real-world cases where authorship is disputed or unknown.
- Data Alignment by Document Size:
  To prevent bias from document length differences, we align all documents to the same size before processing.
  Process:
  1. Record the size of each document.
  2. Find the median size across all documents.
  3. For documents larger than the median, divide them into smaller parts matching the median size.
  4. For documents smaller than the median, repeat the document's content as many times as needed until it reaches the median size.
  5. Verify that all documents now match the median size exactly.

  By aligning document sizes prior to embedding and signal generation, we improve the fairness, accuracy, and interpretability of our authorship verification framework, ensuring that results reflect true stylistic signals rather than superficial length-based differences.

By organizing the data in this way, we aim to build a robust foundation for training and evaluating deep learning models capable of high-fidelity Arabic authorship verification.

Step 2 : Preprocessing the Arabic Texts:
Preprocessing is a foundational phase in the authorship verification pipeline, especially for a linguistically complex language like Arabic. Classical Arabic texts often exhibit irregular orthography, variable spelling conventions, and inconsistent punctuation. Furthermore, diacritics may be present, partial, or absent entirely. These inconsistencies can obscure stylistic patterns unless they are carefully normalized and prepared. The goal of this step is to produce clean, standardized input that preserves stylistic features while minimizing noise, thereby enhancing the performance and reliability of downstream embedding and classification models.
The preprocessing phase includes the following key components:

- Orthographic Normalization:
  Arabic script includes multiple characters that represent the same sound but are written differently, depending on context or author. Normalization reduces this variability by unifying alternate forms:
    - "أ", "إ", and "آ" are replaced with "ا".
    - "ى" is converted to "ي".
    - "ة" is standardized to "ه".
    - Diacritics (tashkil) are removed entirely, as they are inconsistently used and not essential for stylistic modeling.

This normalization helps the model focus on the structural and lexical characteristics of the text, rather than orthographic noise.

- Tokenization:
  Tokenization in Arabic is non-trivial due to the language's agglutinative nature. Words often consist of a root plus multiple affixes, clitics, or attached conjunctions. To segment words accurately, we employ morphological analyzers such as CAMeL Tools [9] or Farasa [10]. These tools effectively disambiguate prefixes, suffixes, and particles, yielding tokenized sequences suitable for embedding.
- Stopword Removal:
  Common Arabic function words (e.g., "في", "من", "الذي") appear frequently and offer little discriminative power for authorship. We apply comprehensive stopword lists such as the El-Khair stopword list or the Arabic Stopword List (ASL) to eliminate these words [24]. This reduces dimensionality and helps the model concentrate on stylistic choices rather than functional grammar.
- Light Stemming:
  To preserve the author's lexical style while reducing inflectional variation, we apply light stemming using tools like ARLSTem [25] or ISRI stemmer. Unlike root-based stemming, light stemming strips common affixes without collapsing words into their root form. This approach retains more of the author's lexical and stylistic footprint.
- Noise Filtering and Cleanup:
  Non-Arabic characters, numerals, and extraneous symbols are removed. Additionally, long character repetitions and inconsistent punctuation are filtered to ensure consistency across the dataset. This is especially important when working with scanned or digitized texts that may include OCR artifacts.

By carefully executing these preprocessing steps, we standardize the text input for embedding models while preserving the linguistic signals relevant for stylistic analysis. This phase transforms raw historical Arabic into structured, computationally tractable input ready for deep modeling and similarity comparison.

Step 3 : Embedding and Representation Learning with AraBERT
After preprocessing, the next step is to convert Arabic text into numerical representations that capture both meaning and stylistic nuance. Traditional vectorization methods such as Bag-of-Words or TF-IDF fail to preserve context or sentence structure, which are crucial for identifying authorial style. To overcome these limitations, we use deep contextual embeddings derived from transformer-based models specifically AraBERT, which is pre-trained on large Arabic corpora and optimized for morphological and syntactic features unique to the language.
The embedding process encompasses the following stages:

- Model Selection and Rationale:
  AraBERT is based on Google's original BERT architecture, adapted specifically for Arabic by training on multi-domain corpora including Arabic Wikipedia, the OSIAN corpus, and news articles [12]. This makes it a strong candidate for capturing deep contextual meaning and stylistic tendencies in Arabic texts. We utilize AraBERT v2 or v3, depending on performance evaluations, as these newer versions incorporate improved token coverage and training stability.
- Tokenization with SentencePiece:
  AraBERT uses the SentencePiece tokenizer to segment Arabic text into subword units. This approach addresses the out-of-vocabulary problem by breaking rare or unseen words into smaller, known components. Unlike word-level tokenization, subword units preserve important stylistic markers (e.g., prefixes, roots) even for rare forms, an important feature in classical Arabic texts where poetic or rhetorical variation is common.
- Sequence Handling and Chunking:
  Due to input length constraints in BERT-based models (typically 512 tokens), each document is split into multiple overlapping chunks. These chunks preserve the sequential order of the original text and are processed independently. Padding and special tokens are added as required by the model architecture.
- Embedding Extraction:
  Once tokenized, each chunk is passed through AraBERT, which outputs a dense vector for each token in a for example 768-dimensional space. These contextual embeddings encode rich linguistic information capturing semantics, syntax, and local dependencies allowing stylistic cues to be embedded implicitly in the vector space. The output for each chunk is typically a matrix of shape, for example [sequence_length × 768].
- Representation Aggregation:
  For downstream tasks such as similarity modeling and authorship verification, embeddings are aggregated using one of several strategies:
  - Mean pooling: averaging all token embeddings in a chunk.
  - CLS vector: using the special [CLS] token representation as a summary.
  - Feature maps: passing the embedding matrix to convolutional or recurrent layers for further processing (as in the Siamese network step).

The result of this stage is a numerical, high-dimensional representation of the text that encodes context-aware stylistic features. These embeddings form the basis for pairwise similarity comparison, signal construction, and final authorship assessment.
By leveraging AraBERT, we bridge the gap between classical Arabic literature and modern deep learning methods. The model's ability to capture nuanced semantic relationships and stylistic markers makes it a powerful tool for authorship verification in a linguistically rich and underrepresented language.

Step 4: Siamese Network Training and Similarity Modeling
Once text chunks have been converted into contextual embeddings using AraBERT, the next objective is to model stylistic similarity between different pieces of text. To accomplish this, we employ a Siamese neural network architecture specifically designed to compare pairs of inputs and measure their similarity based on learned representations. This approach is particularly well-suited to authorship verification tasks, where the goal is to determine whether two pieces of writing are stylistically aligned [26][27].
The Siamese network structure consists of two identical subnetworks (sharing weights) that process input text pairs in parallel. These subnetworks are composed of:

- Convolutional Neural Networks (CNNs): These layers apply filters to detect local stylistic patterns such as recurring syntactic constructions or characteristic phrasing. In the context of Arabic, CNNs are effective for capturing morphological forms, rhetorical markers, and fixed expressions common to specific authors.
- Bidirectional Long Short-Term Memory (BiLSTM) layers: These layers capture long-range dependencies and stylistic flow across the chunk. They process the input both forward and backward, which is crucial in Arabic due to its flexible word order and clause structure.

Each input text chunk represented as an embedding matrix is passed through these layers, resulting in a fixed-size vector. The Euclidean distance ($d = \sqrt{\sum_{i=1}^{n}(z_{1i} - z_{2i})^2}$) between the vectors output by the twin subnetworks is then computed. The closer the distance, the more stylistically similar the texts are.
To train the network, we use a contrastive loss function, which minimizes the distance between embeddings of similar pairs (from the same author) and maximizes it for dissimilar pairs (from different authors). The formula is:

$$L = (1 - Y) \cdot \frac{1}{2} \cdot D^2 + Y \cdot \frac{1}{2} \cdot \max(0, m - D)^2$$

Where:

- Y = 0 for similar pairs, 1 for dissimilar pairs
- D = Euclidean distance between the vectors
- m = margin that defines the minimum separation for dissimilar pairs

This approach allows the model to learn what makes two texts stylistically alike or different, enabling it to function effectively in real-world authorship attribution scenarios where stylistic comparison is more informative than classification.

Step 5: Signal Construction and Comparison
After measuring chunk-level similarity using the Siamese network, we aggregate these results to create a signal representation of the entire document. Each text is divided into sequential chunks, and each chunk is scored based on its similarity to reference chunks from the target author.

These scores, collected across the document, form a one-dimensional signal that captures stylistic consistency.
This signal is then compared against signals from other documents using Dynamic Time Warping (DTW) a technique originally developed for time-series alignment [28]. DTW allows us to compute the optimal alignment between two signals even if they vary in length or pace. In authorship verification, DTW is used to match writing style patterns across documents regardless of sentence or section ordering.
This phase enhances robustness, especially when comparing texts that discuss similar themes but present ideas in different sequences. By treating stylistic flow as a temporal signal, we enable a more nuanced form of comparison than traditional feature-based classifiers.

Step 6: Calculation of Random Forest Scores Using Pairwise DTW Distances

After generating DTW-based similarity signals for each document, we extract pairwise DTW distances between the test documents and the reference target author's documents [28]. These distances form a feature space that encodes stylistic similarity across the corpus.
To translate these raw distances into a probabilistic similarity score, we employ a Random Forest classifier. This ensemble learning model analyzes the distance profiles and outputs a prediction score indicating the likelihood that a test document shares authorship with the target author. Random Forest is chosen for its ability to handle non-linear patterns and reduce overfitting, especially valuable in noisy or imbalanced datasets like historical Arabic texts. The classifier is trained using pairs of texts labeled as "same author" or "different author," and it outputs a confidence score for each new test case.
This phase serves as a bridge between low-level signal similarity and higher-level authorship decisions, allowing the system to weigh multiple DTW comparisons and deliver a calibrated score for each document.

Step 7: Anomaly Detection Using Clustering of Anomaly Scores

In the final verification phase, we analyze the Random Forest [17] output scores using unsupervised learning techniques to classify texts as authentic or impostor-written. To accomplish this, we apply Isolation Forest (IF) an anomaly detection algorithm that works by building random decision trees. In this context, a document whose stylistic pattern is unusual will be isolated with fewer tree splits, indicating a high anomaly score and suggesting that the text may be an impostor.
Texts whose Random Forest-based scores are consistent with the target author's style require more splits to isolate, and are therefore assigned lower anomaly scores, classifying them as inliers. This provides a robust mechanism for detecting stylistic outliers in the corpus.
To further validate the anomaly detection process, we apply k-means clustering (for example with k=2) on the resulting anomaly scores. This groups the documents into stylistically consistent and inconsistent clusters. If a document is classified as an outlier by the Isolation Forest and also falls into the outlier cluster via k-means [20], the system's confidence in that classification increases significantly.
This dual-method verification combining anomaly detection with clustering enhances the robustness of our authorship verification process. It not only supports final attribution decisions but also enables deeper insights into the stylistic landscape of the dataset, revealing subtle authorial shifts, disputed texts, and possible misattributions.

4.3 Sequence Diagram

The following sequence diagrams illustrate key stages of the proposed algorithm. Figure 5 represents the process generally, then Figure 6 is a Detailed Sequence Diagram of the proposed algorithm process.

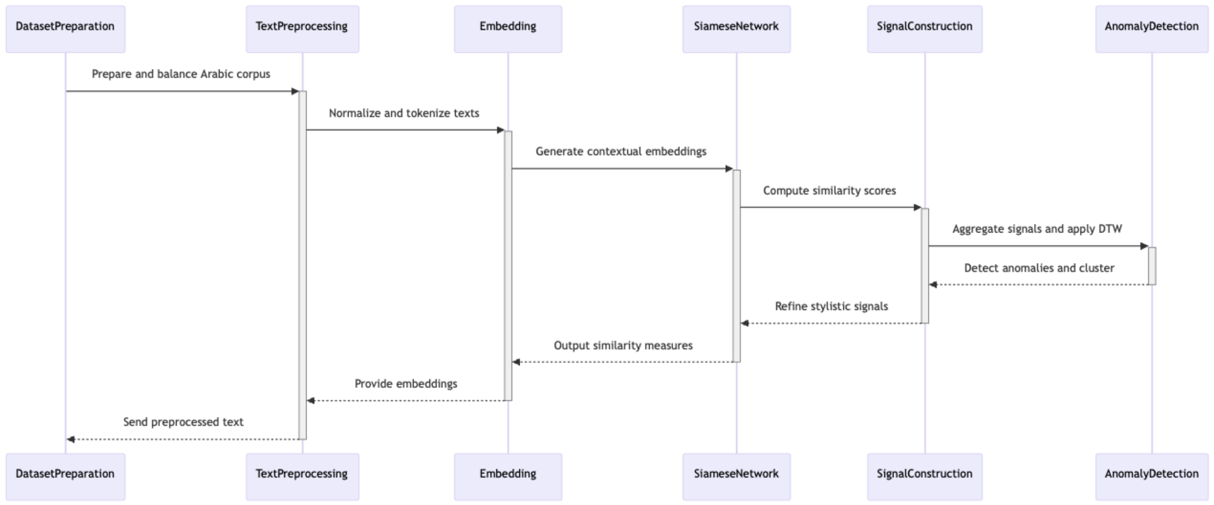
DatasetPreparation
TextPreprocessing
Embedding
SiameseNetwork
SignalConstruction
AnomalyDetection
Prepare and balance Arabic corpus
Normalize and tokenize texts
Generate contextual embeddings
Compute similarity scores
Aggregate signals and apply DTW
Detect anomalies and cluster
Refine stylistic signals
Output similarity measures
Provide embeddings
Send preprocessed text

Figure 5. Mini Sequence Diagram of the proposed algorithm process.

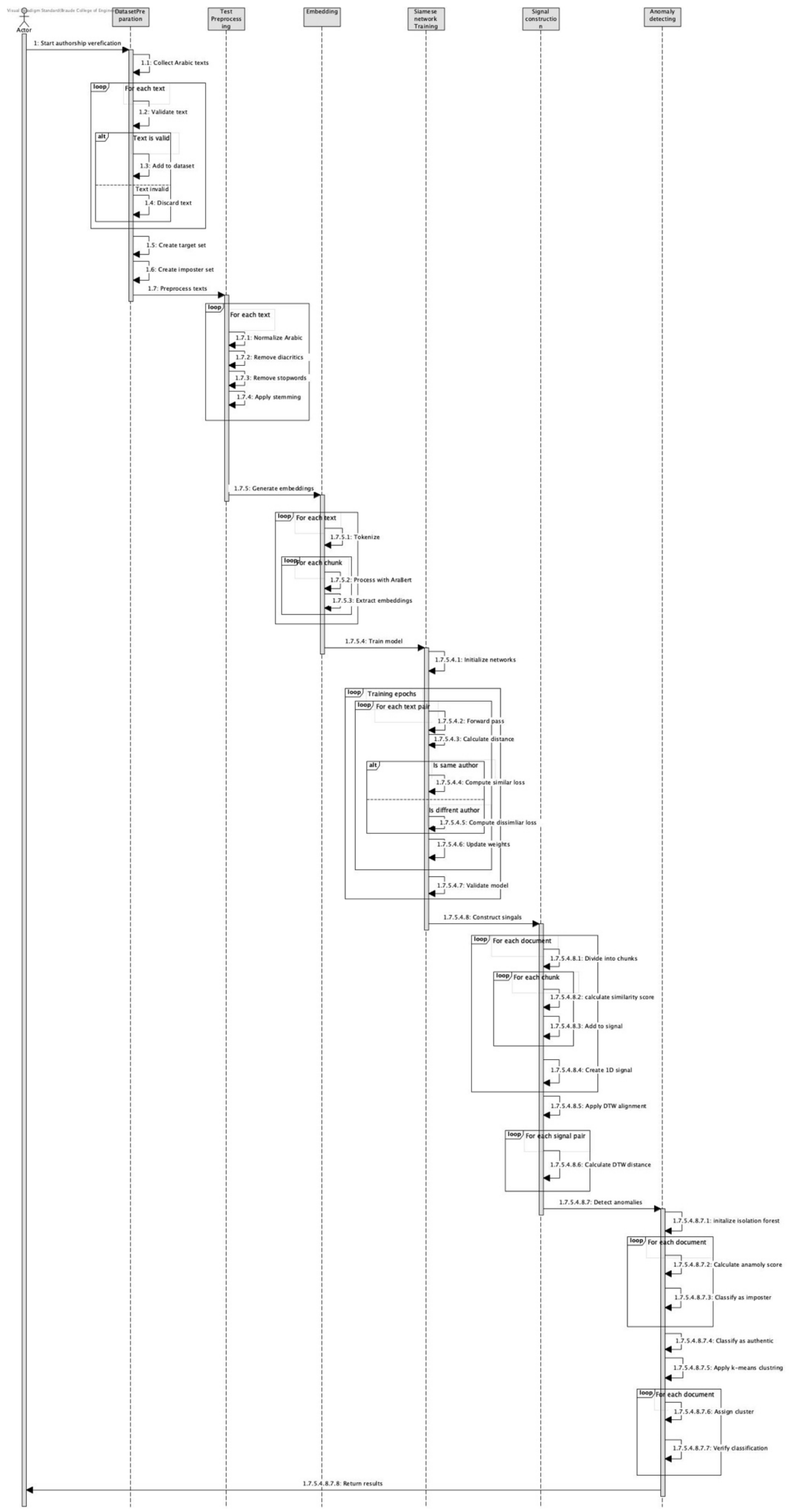
Visual Paradigm Standard(Braude College of Engineering)
Actor
DatasetPreparation
Test Preprocessing
Embedding
Siamese network Training
Signal construction
Anomaly detecting
1: Start authorship verefication
1.1: Collect Arabic texts
loop For each text
1.2: Validate text
alt Text is valid
1.3: Add to dataset
Text invalid
1.4: Discard text
1.5: Create target set
1.6: Create imposter set
1.7: Preprocess texts
loop For each text
1.7.1: Normalize Arabic
1.7.2: Remove diacritics
1.7.3: Remove stopwords
1.7.4: Apply stemming
1.7.5: Generate embeddings
loop For each text
1.7.5.1: Tokenize
loop For each chunk
1.7.5.2: Process with AraBert
1.7.5.3: Extract embeddings
1.7.5.4: Train model
1.7.5.4.1: Initialize networks
loop Training epochs
loop For each text pair
1.7.5.4.2: Forward pass
1.7.5.4.3: Calculate distance
alt Is same author
1.7.5.4.4: Compute similar loss
Is diffrent author
1.7.5.4.5: Compute dissimliar loss
1.7.5.4.6: Update weights
1.7.5.4.7: Validate model
1.7.5.4.8: Construct singals
loop For each document
1.7.5.4.8.1: Divide into chunks
loop For each chunk
1.7.5.4.8.2: calculate similarity score
1.7.5.4.8.3: Add to signal
1.7.5.4.8.4: Create 1D signal
1.7.5.4.8.5: Apply DTW alignment
loop For each signal pair
1.7.5.4.8.6: Calculate DTW distance
1.7.5.4.8.7: Detect anomalies
1.7.5.4.8.7.1: initalize isolation forest
loop For each document
1.7.5.4.8.7.2: Calculate anamoly score
1.7.5.4.8.7.3: Classify as imposter
1.7.5.4.8.7.4: Classify as authentic
1.7.5.4.8.7.5: Apply k-means clustring
loop For each document
1.7.5.4.8.7.6: Assign cluster
1.7.5.4.8.7.7: Verify classification
1.7.5.4.8.7.8: Return results

Figure 6. Detailed Sequence Diagram of the proposed algorithm process.

5. Evaluation Plan

To assess the effectiveness and reliability of the proposed authorship attribution framework, a structured evaluation plan is implemented. The evaluation is designed to test the framework across all stages, including text preprocessing, embedding generation, similarity assessment, and clustering. The goal is to determine whether the system can consistently and accurately distinguish between texts written by the same author and those written by different authors. The evaluation considers both clean and noisy data inputs to test robustness, and applies clustering methods to assess how well texts are grouped based on stylistic and semantic similarities. Performance will be measured using standard metrics such as accuracy, precision, recall, and clustering effectiveness indicators. This process is carried out under the assumption that preprocessed and segmented texts are available, and that similarity in writing style is reflected in the model's output. The evaluation will help confirm the system's readiness for broader application to diverse Arabic texts.
A detailed outline of the evaluation plan is presented in Table 1.

Table 1. Detailed Evaluation Plan.

Test No.	Test Subject	Expected Result
1	Validate tokenization and normalization: Test Arabic text splitting and standardization.	Accurate and consistent segmentation and formatting of text across the dataset.
2	Verify stopword removal and stemming: Ensure reduction of uninformative content while preserving key lexical features.	Text retains core meaning and stylistic elements, reducing dimensionality.
3	Assess AraBERT embeddings: Evaluate the semantic and stylistic representation of text segments.	High-quality embeddings that differentiate writing styles effectively.
4	Evaluate similarity scoring with Siamese Network: Test how well the model distinguishes between similar and dissimilar text pairs.	Lower distance for same-author pairs; higher distance for different authors.
5	Test robustness to noisy input: Introduce variations like typos or missing chunks.	Minimal accuracy drop; model remains stable under imperfect input.
6	Validate signal representation and DTW alignment: Examine the numeric representation of style and its distance comparison.	Signal distances reflect meaningful stylistic similarity or difference.
7	Evaluate clustering: Group texts based on similarity using Isolation Forest and k-means.	Clear and interpretable clustering that aligns with author boundaries.

8	Run end-to-end system test: Assess the framework's full pipeline from raw input to classification.	Consistent and accurate authorship attribution across multiple text cases.

Note: This project focuses on backend authorship verification functionality and does not include a graphical user interface (GUI). Interaction with the system is performed through scripts and command-line tools designed for researchers and developers.

References

[1] AlZahrani, F. M., & Al-Yahya, M. (2023). *A Transformer-Based Approach to Authorship Attribution in Classical Arabic Texts*. Applied Sciences, 13(12), 7255. https://doi.org/10.3390/app13127255

[2] Alqurashi, L., Sharoff, S., Watson, J., & Blakesley, J. (2025). *BERT-based Classical Arabic Poetry Authorship Attribution*. Proceedings of COLING 2025. https://aclanthology.org/2025.coling-main.409

[3] Howedi, F., Mohd, M., Aborawi, Z. A., & Jowan, S. A. (2020). *Authorship Attribution of Short Historical Arabic Texts using Stylometric Features and a KNN Classifier with Limited Training Data*. Journal of Computer Science, 16(10), 1334–1345. https://doi.org/10.3844/jcssp.2020.1334.1345

[4] Al-Khatib, K., Shaalan, K., & Al-Shalabi, R. (2014). *Naïve Bayes Classifiers for Authorship Attribution of Arabic Texts*. Journal of King Saud University – Computer and Information Sciences, 26(3), 272–280. https://doi.org/10.1016/j.jksuci.2014.03.003

[5] Hajja, M., Yahya, A., & Yahya, A. (2019). *Authorship Attribution of Arabic Articles*. In *Arabic Language Processing: From Theory to Practice* (pp. 194–208). Springer. https://doi.org/10.1007/978-3-030-32959-4_14

[6] Volkovich, Z., & Avros, R. (2025). *Comprehension of the Shakespeare Authorship Question through Deep Impostors Approach*. Digital Scholarship in the Humanities. https://doi.org/10.1093/llc/fqaf009

[7] OpenITI: The Open Islamicate Texts Initiative – https://openiti.github.io

[8] Shamela Library – https://shamela.ws

[9] Habash, N., Khalifa, S., Taji, D., Obeid, O., Zaghouani, W., & Bouamor, D. (2021). CAMeL Tools: An Open Source Python Toolkit for Arabic Natural Language Processing. *Proceedings of the Sixth Arabic Natural Language Processing Workshop*, 97–110. https://aclanthology.org/2021.wanlp-1.11/

[10] Abdelali, A., Darwish, K., Durrani, N., & Mubarak, H. (2016). Farasa: A Fast and Furious Segmenter for Arabic. *NAACL 2016: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 11–16. https://aclanthology.org/N16-3003/

[11] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*.
arXiv: https://arxiv.org/abs/1301.3781

[12] Antoun, W., Baly, F., & Hajj, H. (2020). *AraBERT: Transformer-based Model for Arabic Language Understanding*.
LREC paper: https://arxiv.org/abs/2003.00104

[13] Volkovich, Z., & Avros, R. (2023). *Comprehension of the Shakespeare Authorship Question Through Deep Impostors Approach*.

[14] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). *Data Clustering: A Review*.
ACM Computing Surveys: https://dl.acm.org/doi/10.1145/331499.331504

[15] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.
arXiv: https://arxiv.org/abs/1810.04805

[16] Pennington, J., Socher, R., & Manning, C. D. (2014). *GloVe: Global Vectors for Word Representation*.
EMNLP paper: https://aclanthology.org/D14-1162/
Project page: https://nlp.stanford.edu/projects/glove/

[17] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). *Isolation Forest*.
IEEE ICDM: https://ieeexplore.ieee.org/document/4781136

[18] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention is All You Need*.
NeurIPS: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[19] Müller, M. (2007). *Dynamic Time Warping*. In *Information Retrieval for Music and Motion*.
SpringerLink: https://link.springer.com/chapter/10.1007/978-3-540-74048-3_4

[20] Jain, A. K. (2010). Data Clustering: 50 Years Beyond K-Means. *Pattern Recognition Letters*, 31(8), 651–666.

[21] Goldberg, Y. (2016). *A Primer on Neural Network Models for Natural Language Processing*. JAIR: https://jair.org/index.php/jair/article/view/10840

[22] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Book on Springer: https://link.springer.com/book/10.1007/978-0-387-45528-0

[23] Habash, N. (2010). *Introduction to Arabic Natural Language Processing*. Book from Morgan & Claypool: https://www.morganclaypool.com/doi/abs/10.2200/S00277ED1V01Y201003HLT005

[24] El-Khair, I. A. (2012). Effects of Stop Words Elimination for Arabic Information Retrieval: A Comparative Study. *International Journal of Computing and Information Sciences*, 10(2), 75–81.

[25] Larkey, L. S., Ballesteros, L., & Connell, M. E. (2007). Light Stemming for Arabic Information Retrieval. In *Arabic Computational Morphology* (pp. 221–243). Springer. https://doi.org/10.1007/978-1-4020-6046-5_12

[26] Bromley, J., Guyon, I., LeCun, Y., Sackinger, E., & Shah, R. (1994). Signature Verification using a Siamese Time Delay Neural Network. *Advances in Neural Information Processing Systems*, 6, 737–744.

[27] Mueller, J., & Thyagarajan, A. (2016). Siamese Recurrent Architectures for Learning Sentence Similarity. *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2786–2792.

[28] 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.