Software Engineering Department

Capstone Project Phase B – 61999

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

Table of Contents

1. Introduction

Authorship verification in Arabic literature poses significant linguistic and computational challenges. Classical Arabic texts exhibit rich morphology, stylistic diversity, orthographic variation, and inconsistent use of diacritics, all of which complicate automated textual analysis. Many historically significant works suffer from uncertain or disputed authorship, limiting scholarly consensus and motivating the development of computational tools capable of offering reproducible, data driven insights.

While modern authorship attribution techniques have achieved strong performance in English and other European languages, Arabic remains comparatively underrepresented, particularly in open set verification scenarios where disputed or unknown authors may appear. Existing approaches often rely on supervised classification or handcrafted stylometric features, which are difficult to generalize across historical genres and periods.

This project addresses these challenges by implementing a research framework for Arabic authorship verification based on the Deep Impostors methodology [4]. The framework reframes authorship verification as a signal comparison problem, combining contextual embeddings, neural similarity learning, time series alignment, and unsupervised anomaly detection to identify stylistic consistency and deviation.

1.1 Scope of This Document

This document presents Phase B of the capstone project and focuses on the implementation, experimentation, and evaluation of the framework proposed in Phase A. While Phase A introduced the conceptual model and theoretical background, this phase details how the methodology was translated into a functioning system and empirically validated on classical Arabic texts.

The paper describes the implemented pipeline, experimental setup, datasets, results, and conclusions. The intended audience includes researchers and practitioners in computational linguistics, digital humanities, and Arabic literary studies.

1.2 Case Study: Al-Jāḥiẓ

Al-Jahiz (Abū ʿUthmān ʿAmr ibn Baḥr al-Kinānī, d. 869 CE) [1] is one of the most prominent prose writers in classical Arabic literature. His works cover diverse genres and are known for a distinctive rhetorical and argumentative style.

Several texts traditionally attributed to Al-Jahiz remain disputed, as some exhibit stylistic patterns that differ from his widely accepted works. This combination of a well established authorial core and contested texts makes his corpus particularly suitable for authorship verification research.

Al-Jahiz was therefore selected as the target author in this study to evaluate whether the proposed Deep Impostors based framework can distinguish stable authorial style from stylistic deviation in a challenging classical Arabic setting.

2. Proposed Solution

This section presents the implemented authorship verification framework. Figure 1 is a general describtion of the work flow that was built in Phase A. The system described here is fully implemented and experimentally evaluated.

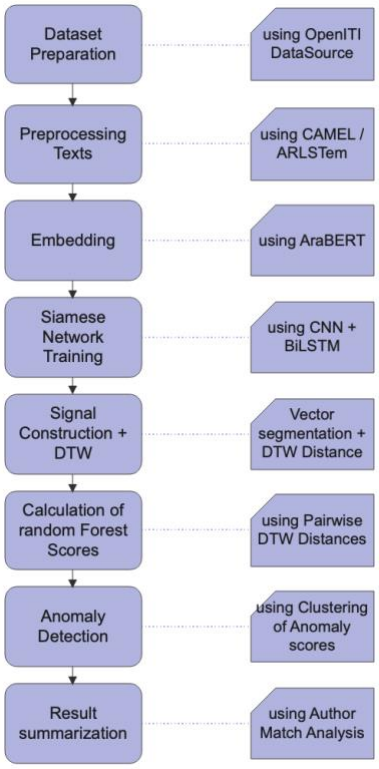


Figure 1. WorkFlow Diagram of the proposed Framework.

2.1 System Overview

The system is a modular, end to end research framework that accepts Arabic text documents and produces authorship verification outcomes based on stylistic similarity and anomaly detection. It is designed for reproducibility and extensibility rather than interactive end user operation.

At a high level, the framework consists of the following stages:

1. Dataset preparation
2. Arabic text preprocessing
3. Contextual embedding using AraBERT
4. Siamese neural network training for stylistic similarity
5. Stylometric signal construction
6. Signal comparison using Dynamic Time Warping (DTW) [6]
7. Anomaly detection and clustering

Each stage is implemented as a distinct processing module, enabling systematic experimentation and controlled evaluation.

2.2 Algorithmic Pipeline

During training, the system constructs multiple impostor pairs and trains a Siamese neural network to distinguish between their writing styles using contrastive learning. At inference time, the trained model is applied to test texts, producing sequential similarity predictions that are aggregated into stylometric signals.

These signals are compared across documents using DTW, and the resulting distance profiles are analyzed using Isolation Forest [7] to detect stylistic outliers. Final authorship interpretation is derived through clustering of anomaly scores. In Figure 2, we present the psudocode of the full algorithm that we implemented.

Figure 2. Pseudocode of our authorship verification algorithm.

Algorithm 1 Authorship Verification in Arabic Sources via Deep Impostors

Input: Impostor corpus $\mathcal{I}$, Test corpus $\mathcal{T}$
Output: Cluster labels $\mathbf{c}$ for $\mathcal{T}$, anomaly score matrix $\mathbf{A}$

$(\mathcal{I}, \mathcal{T}) \leftarrow \mathsf{PreprocessAll}(\mathcal{I}, \mathcal{T})$
$\mathcal{P} \leftarrow \mathsf{AllUniquePairs}(\mathcal{I})$
$\theta_0 \leftarrow \mathsf{LoadMLM}(\mathsf{AraBERTv2})$
$\mathcal{D} \leftarrow \mathsf{BuildDataset}(\mathcal{I} \cup \mathcal{T})$
$\theta^* \leftarrow \mathsf{MLMTrain}(\theta_0, \mathcal{D})$
$\mathbf{E}_{\mathcal{T}} \leftarrow \mathsf{EmbedAll}(\mathcal{T}, \theta^*)$
for each impostor pair $(i_1, i_2) \in \mathcal{P}$ **do**
  $\mathcal{S}_1 \leftarrow \mathsf{SegmentText}(i_1)$
  $\mathcal{S}_2 \leftarrow \mathsf{SegmentText}(i_2)$
  $\mathbf{E}_1 \leftarrow \mathsf{Embed}(\mathcal{S}_1, \theta^*)$
  $\mathbf{E}_2 \leftarrow \mathsf{Embed}(\mathcal{S}_2, \theta^*)$
  $\mathbf{X}_1 \leftarrow \mathsf{Chunkify}(\mathbf{E}_1)$
  $\mathbf{X}_2 \leftarrow \mathsf{Chunkify}(\mathbf{E}_2)$
  $f \leftarrow \mathsf{InitSiamese}(\mathsf{CNN{+}BiLSTM})$
  $\mathsf{TrainSiamese}(f, \mathbf{X}_1, \mathbf{X}_2)$
  for each test document $t \in \mathcal{T}$ **do**
    $\mathbf{s}_t \leftarrow \emptyset$
    for each batch $b \in \mathbf{E}_{\mathcal{T}}(t)$ **do**
      $d \leftarrow \mathsf{PredictDistance}(f, b)$
      $\mathbf{s}_t \leftarrow \mathbf{s}_t \cup \{d\}$
    end for
    $\mathsf{SaveSignal}(t, i_1, i_2, \mathbf{s}_t)$
  end for
end for
for each impostor pair $(i_1, i_2) \in \mathcal{P}$ **do**
  $\{\mathbf{s}_t\}_{t \in \mathcal{T}} \leftarrow \mathsf{LoadSignals}(\mathcal{T}, i_1, i_2)$
  $\mathbf{D}_{(i_1,i_2)} \leftarrow \mathsf{DTWMatrix}(\{\mathbf{s}_t\})$
  $\mathsf{SaveDTWMatrix}(i_1, i_2, \mathbf{D}_{(i_1,i_2)})$
end for
$\mathbf{A} \leftarrow [\,]$
for each impostor pair $(i_1, i_2) \in \mathcal{P}$ **do**
  $\mathbf{D} \leftarrow \mathsf{LoadDTWMatrix}(i_1, i_2)$
  $\widehat{\mathbf{D}} \leftarrow \mathsf{Standardize}(\mathbf{D})$
  $\mathbf{a}_{(i_1,i_2)} \leftarrow \mathsf{IsolationForest}(\widehat{\mathbf{D}})$
  $\mathbf{A} \leftarrow \mathsf{AppendColumn}(\mathbf{A}, \mathbf{a}_{(i_1,i_2)})$
end for
$\mathbf{c} \leftarrow \mathsf{KMeans}(\mathbf{A}, k = 2)$
return $\mathbf{c}, \mathbf{A}$

3. Experiment Setup

3.1 Dataset

The dataset consists of classical Arabic texts collected from reliable open access digital repositories [2] [3]. Texts are divided into two groups:

1. Impostor Group: The impostor set contains texts written by authors from similar historical periods and overlapping genres with Al-Jahiz , including theology, philosophy, ethics, and jurisprudence. This group is intentionally diverse to expose the model to stylistic variation while remaining contextually comparable.In Table 1 is a list of total 25 impostor books that were used.

Table 1. All Impostors used in our Experiment with reasons.

	Book	Author	Why it is a good impostor for al-Jahiz
1	*al-Fāḍil*	al-Mubarrad	Dry linguistic prose. Close in time, lacks Jahiz's satire
2	*al-Kāmil fī al-Lugha wa al-Adab*	al-Mubarrad	Long, stable analytical prose. Strong stylistic signal
3	*al-Muqtaḍab*	al-Mubarrad	Strict grammatical reasoning.
4	*Adab al-Kātib*	Ibn Qutayba	Structured literary prose without irony
5	*al-Imāma wa al-Siyāsa*	Ibn Qutayba	Political prose. Tests attribution in political genre
6	*ʿUyūn al-Akhbār*	Ibn Qutayba	Narrative adab with stable authorial voice
7	*Ansāb al-Ashrāf*	al-Baladhuri	Formal historical prose. No rhetorical play
8	*Futūḥ al-Buldān*	al-Baladhuri	Administrative historical style. Clear genre separation
9	*Tārīkh al-Yaʿqūbī*	al-Yaqubi	Early analytical historiography close to Jahiz's era
10	*Kitāb al-Buldān*	al-Yaʿqubi	Descriptive geographic prose. Non-literary
11	*Mushākalat al-Nās*	al-Yaʿqubi	Short social prose. Secondary impostor
12	*Yatīmat al-Dahr*	al-Tha'alibi	Anthological adab. Superficially similar, stylistically different
13	*Aḥsan mā Samiʿtu*	al-Thaʿālibī	Compiled literary prose. Weak individual authorial voice
14	*al-ʿIqd al-Farīd*	Ibn Abd Rabbih	Encyclopedic adab. Medium strength impostor
15	*Dīwān Ibn ʿAbd Rabbih*	Ibn ʿAbd Rabbih	Mixed prose and poetry. Tests genre sensitivity
16	*Kalīla wa Dimna*	Ibn al-Muqaffa'	Ethical narrative prose. Rational but non-satirical
17	*al-Adab al-Ṣaghīr*	Ibn al-Muqaffaʿ	Concise didactic prose. Clear stylistic contrast

18	*Durra Yatīma*	Ibn al-Muqaffaʿ	Short moral epistle. Secondary impostor
19	*al-Muqābasāt*	Abu Hayyan al-Tawhidi	Philosophical dialogue prose. Jahiz-influenced
20	*al-Baṣāʾir wa al-Dhakhāʾir*	al-Tawḥīdī	Reflective encyclopedic prose. Tests imitation
21	*al-Imtāʿ wa al-Muʾānasa*	al-Tawḥīdī	Literary conversations. Close but non-satirical
22	*Murūj al-Dhahab*	al-Masudi	Historical encyclopedic prose. Distant stylistically
23	*al-Tanbīh wa al-Ishrāf*	al-Masʿūdī	Analytical historical prose. Strong genre contrast
24	*al-Shifāʾ (Ilāhiyyāt)*	Ibn Sina	Philosophical prose. Ideal distant impostor
25	*al-Najāt*	Ibn Sīnā	Logical exposition. Expands impostor space

2. Test Group: The test set contains 55 texts attributed to Al-Jahiz and includes both widely accepted and disputed works. Each book was segmented into equal sized parts, producing a total of 90 test samples.

To prevent bias introduced by document length, all texts were aligned to a uniform size using a median based strategy. Larger texts were divided, and smaller texts were repeated until the target size was reached.

3.2 Preprocessing

Arabic text preprocessing was designed to reduce orthographic noise while preserving stylistic information. The following steps were applied:

- Orthographic normalization of letter variants
- Removal of diacritics
- Arabic aware tokenization handling clitics and affixes
- Stopword removal
- Light stemming to reduce inflectional variation
- Filtering of non-Arabic characters and digitization artifacts

These steps ensured consistency across the corpus and improved embedding stability.

3.3 Model Configuration

Contextual embeddings were generated using AraBERT [5] , a transformer based model pretrained on large Arabic corpora. Due to model input length limitations, documents were processed as sequential chunks.

The Siamese network architecture combines convolutional layers for local stylistic feature extraction with bidirectional LSTM layers for modeling long-range dependencies. Similarity between chunks is measured using Euclidean distance and optimized using contrastive loss.

Experiments were conducted in Python using PyTorch and the Transformers library. Initial development and testing were performed on Google Colab Pro+, while large scale training and signal generation were executed on Lambda Cloud using high performance GPUs to reduce runtime. All used parameters and methods in each step of our pipeline are presented in Table 2.

Table 2. Experimental Configuration and full Model Parameters and Methods.

Category	**Component**	**Parameter**	**Value / Description**
Environment	Programming Language	Version	Python 3.x
	Deep Learning Framework	Library	PyTorch
	NLP Framework	Library	HuggingFace, Transformers
	Development Platform	Environment	Google Colab Pro+
	High Performance Training	Environment	Lambda Cloud
	GPU	Hardware	NVIDIA GH200
	Reproducibility	Random Seed	Fixed per run
Experiment Scale	Impostor Iterations	Count	300
	Impostor Books	Count	25
	Test Books	Count	55
	Test Samples	Segmented Parts	90
Document Alignment	Impostor Text Size	Chunk Size	~250 KB
	Test Text Size	Chunk Size	~200 KB
Segmentation	Segment Size	Words	50
	Chunk Size	Segments	8 segments (~400 words)
Embedding	AraBERT Version	Model ID	aubmindlab/bert-base-arabertv2
	Tokenizer	Type	SentencePiece
	Embedding Dimension	Hidden Size	768
	Maximum Token Length	Tokens	128
	Sequence Length	Tokens	64
Siamese Network	Architecture	Type	Siamese CNN-BiLSTM
	CNN Layers	Count	3
	CNN Kernel Sizes	Sizes	(3, 6, 12)
	CNN Feature Maps	Filters	300
	CNN Dropout	Rate	0.25
	LSTM Layers	Count	2
	LSTM Hidden Size	Units	300
	LSTM Dropout	Rate	0.25
	Activation Function	Type	ReLU
Training	Loss Function	Type	Contrastive Loss
	Optimizer	Type	Adam
	Learning Rate	Value	2×10^{-5}
	Batch Size	Samples	100
	Validation Split	Ratio	0.25

	Epochs	Count	5
Signal Processing	Signal Representation	Type	1D stylometric signal
	Signal Aggregation	Method	Mean over chunks
Similarity Analysis	Time-Series Comparation	Algorithm	Dynamic Time Warping
	DTW Distance Metric	Type	Euclidean
	Distance Structure	Output	Pairwise DTW matrix
Anomaly Detection	Algorithm	Method	Isolation Forest
	Anomaly Score	Range	Normalized
Clustering	Algorithm	Method	K-Means
	Number of Clusters	K	2

4. Research Process

The research process followed a modular and iterative workflow. After dataset collection and preprocessing, each component of the pipeline was implemented and validated independently before integration.

Embeddings were cached to avoid redundant computation, and stylometric signals were stored to support repeated experimentation. Multiple full runs were executed using randomized impostor pair selections to ensure robustness and reduce sensitivity to individual author pair choices.

4.1 Implementation Details

The Siamese model was implemented using PyTorch, guided by established contrastive learning principles. Chunk size, kernel dimensions, and hidden layer sizes were selected based on empirical stability rather than aggressive optimization.

Signal construction aggregated chunk level similarity predictions sequentially, preserving document structure. DTW was applied to align signals with differing local stylistic pacing, and Isolation Forest was used to identify stylistic deviations without supervised labels.

4.2 Challenges and Adjustments

Several challenges emerged during implementation:

- Data acquisition: Some historically important impostors for Al-Jahiz, such as his teacher (Al-Naẓẓām), could not be included because their works are lost and not digitally available.
- Computational cost: Training Siamese networks across multiple impostor pairs was computationally intensive. Migrating from Colab to Lambda Cloud significantly improved feasibility.
- Document size bias: Early experiments revealed clustering driven by file size rather than style. This issue was resolved through strict document size alignment.
- Impostor embedding leakage: Precomputing impostor embeddings introduced information leakage, leading us to embed impostor data inside the training loop to preserve experimental validity.

- Runtime instability and resumability: Due to runtime crashes in cloud environments, long running processes such as Siamese training and DTW matrix generation were susceptible to interruption. To mitigate data loss and avoid repeating completed computations, both stages were implemented as resumable loops. DTW matrices were generated and saved incrementally, and training progress was tracked using log files recording completed impostor pairs, allowing execution to resume from the last successful state.

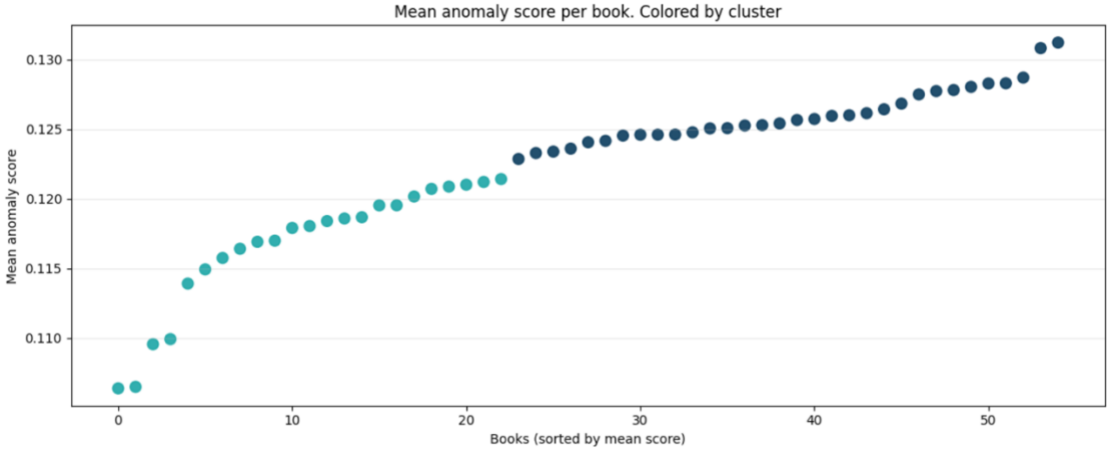
5. Experiment Results

This section presents the experimental results obtained from our Deep Impostors based authorship verification framework. The analysis focuses on anomaly score behavior, clustering outcomes, and agreement with external scholarly labels.

The results are presented progressively, following the decision pipeline. Mean anomaly scores. Cluster separation. Final classification performance.

Figure 3 presents the mean Isolation Forest anomaly score for each test book. Scores are averaged across all impostor pair runs and sorted in ascending order.

Figure 3. Mean anomaly score per book (sorted by mean score) , colored by cluster.



In Figure 3 we see the distribution of mean anomaly scores across all test books. Each point represents a single book, with scores averaged over all impostor pair evaluations. Lower scores indicate stylistic consistency with the dominant authorial pattern, while higher scores suggest stylistic deviation.

The visualization reveals a gradual but clear separation between two groups of books. This supports the hypothesis that aggregated anomaly scores capture meaningful stylistic differences, even before explicit clustering is applied.

To further analyze stylistic separation, K-Means clustering (k = 2) was applied to the averaged anomaly scores.

Figure 4. Mean Isolation Forest score per book. Colored by cluster ID..

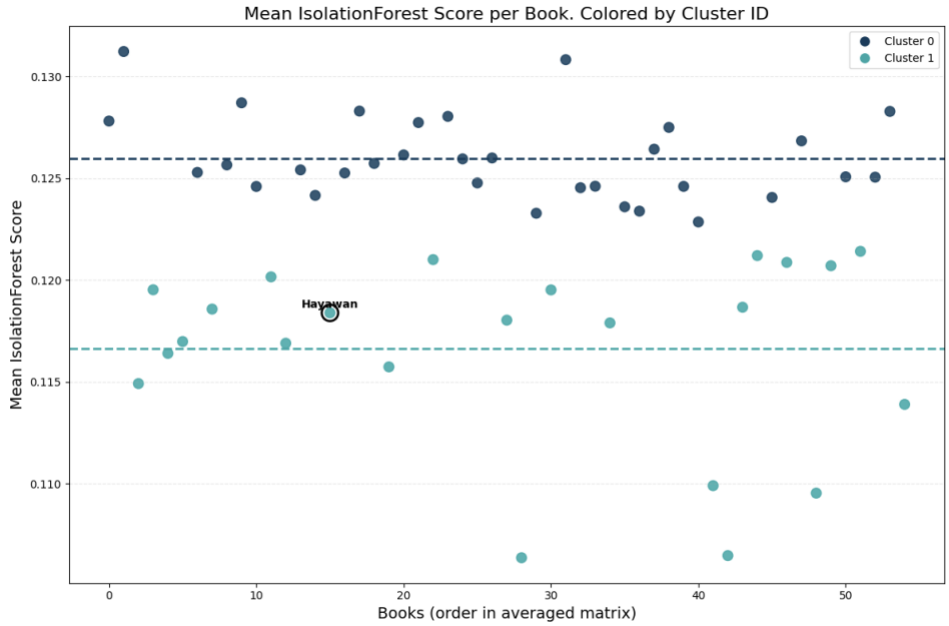


Figure 4 illustrates the same mean anomaly scores, now colored by their K-Means cluster assignment. Two compact and well-separated clusters are observed.

The cluster with lower mean anomaly scores corresponds to stylistically consistent texts, while the higher-score cluster contains texts that deviate more strongly from the dominant style. Dashed horizontal lines indicate the mean score of each cluster, highlighting the separation margin.

The work Hayawan, a widely accepted authentic book by Al-Jahiz, appears within the low-anomaly cluster. This observation was used as an anchor to interpret this cluster as representing authentic authorship.

Based on the clustering outcome, the cluster containing Hayawan was labeled as authentic, while the opposite cluster was interpreted as potentially disputed.

To assess alignment with scholarly consensus, predicted cluster labels were compared against reference classifications derived from historical and literary research.

Figure 5. Confusion matrix comparing predicted clusters with external authorship labels.

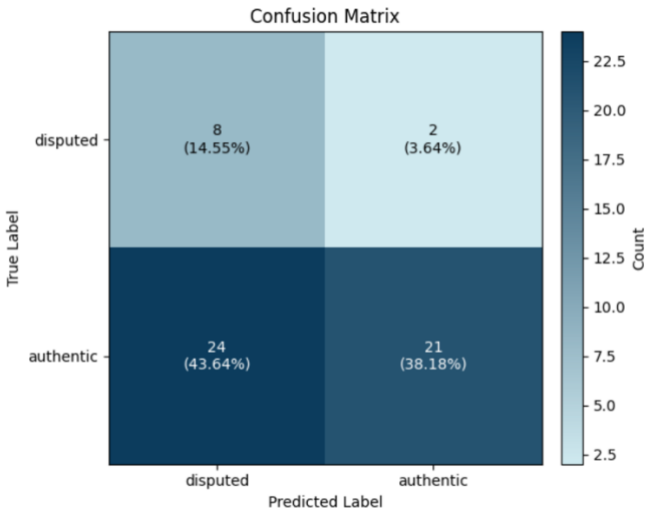


Figure 5 summarizes the agreement between the system's stylistic clustering and external scholarly classifications. Texts labeled as authentic by external sources largely concentrate in the stylistically consistent cluster, indicating that the proposed framework captures a stable authorial signal for Al-Jahiz.

To evaluate the performance of the proposed authorship verification system, we analyze the results using a confusion matrix. The confusion matrix compares the predicted labels produced by the model with the true labels derived from historical and scholarly sources regarding potentially disputed works. The two classes considered in this experiment are disputed and authentic. Rows represent the true labels, while columns represent the predicted labels.

From the confusion matrix, four outcomes are observed. True Positives correspond to disputed books that were correctly classified as disputed, with a total of 8 books. False Negatives correspond to disputed books that were incorrectly classified as authentic, with a total of 2 books. False Positives correspond to authentic books that were incorrectly classified as disputed, with a total of 24 books. True Negatives correspond to authentic books that were correctly classified as authentic, with a total of 21 books. The total number of evaluated books is therefore 55.

Accuracy measures the overall proportion of correctly classified books. It is computed by dividing the sum of true positives and true negatives by the total number of books. In this experiment, accuracy is calculated as (8 + 21) divided by 55, which equals approximately 52.7 percent. This indicates that the system correctly classified slightly more than half of the evaluated texts.

Precision for the disputed class measures how reliable the system is when it predicts a book as disputed. Precision is computed by dividing the number of true positives by the sum of true

positives and false positives. In this case, precision is calculated as 8 divided by (8 + 24), which equals 25 percent. This result indicates that many books predicted as disputed are in fact authentic.
Recall for the disputed class measures the system's ability to detect truly disputed books. Recall is computed by dividing the number of true positives by the sum of true positives and false negatives. In this experiment, recall is calculated as 8 divided by (8 + 2), which equals 80 percent. This shows that the system successfully detects most of the disputed books.

The results show that our proposed system captures stylistic behavior effectively and prioritizes the detection of disputed works, reflecting the conservative nature of authorship verification under historical uncertainty.

6. Conclusion

This project presented an authorship verification framework for Arabic texts that formulates the problem as a comparison of stylistic signals rather than direct author classification. By combining contextual embeddings from a fine-tuned AraBERT model with Siamese impostor training, the system models stylistic boundaries without relying on the true author during training. The resulting stylometric signals enable the analysis of stylistic consistency and deviation across texts using Dynamic Time Warping and anomaly detection techniques.
The experimental results demonstrate that the proposed approach is effective at capturing stylistic behavior and identifying potentially disputed works. The system prioritizes recall over precision, successfully detecting most disputed. This behavior aligns with the nature of authorship verification in historical and literary settings, where missing a disputed text is often considered more critical than incorrectly flagging an authentic one. Overall, the results support the viability of signal-based impostor methods for Arabic authorship verification under conditions of limited and uncertain ground truth.

7. Reflection and Lessons Learned

One lesson derived from this project is that authorship verification requires evaluation criteria that differ from those used in standard classification tasks. Because verification focuses on identifying stylistic deviation rather than assigning a single correct label, metrics such as recall and sensitivity to anomalies provide more meaningful insight than overall accuracy, particularly when reference labels are based on historical and scholarly assessments rather than definitive ground truth.
Another important insight is the value of impostor-based training. Training Siamese models on impostor author pairs allows the system to learn stylistic boundaries without bias toward the target author, making the approach more robust in verification scenarios. However, this also increases computational cost and requires careful management of training stability and data leakage.

The project also highlighted the importance of representing texts as sequences of stylistic signals rather than single aggregated embeddings. Signal-based representations preserve internal variation within books and enable alignment methods such as Dynamic Time Warping to account for structural differences between texts.

Finally, working with classical Arabic texts emphasized the challenges of limited, uneven, and historically incomplete data. Variability in text length, genre, and transmission history directly impacts stylistic analysis and must be carefully considered when interpreting results. These lessons reinforce the need for cautious interpretation and conservative decision strategies in computational authorship verification.

References

[1] Al-Jahiz. https://en.wikipedia.org/wiki/Al-Jahiz

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

[3] Shamela Library. https://shamela.ws

[4] 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

[5] Antoun, W., Baly, F., & Hajj, H. (2020). *AraBERT: Transformer-based model for Arabic language understanding*. arXiv. https://arxiv.org/abs/2003.00104

[6] 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. https://ieeexplore.ieee.org/document/1163055

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

User Guide

This user guide provides operating instructions for running the authorship verification system developed in this project.
The guide explains how to execute the research pipeline successfully, from data preparation to final authorship results, using the provided code and notebooks.
The guide focuses exclusively on the nominal successful operation of the system.

System Overview

The system performs authorship verification of Arabic texts using a Deep Impostor method.
The operational pipeline implemented in the code includes the following stages:

1. Arabic text preprocessing
2. Deep embedding generation using AraBERT
3. Siamese network training on impostor author pairs
4. Stylometric signal generation
5. Similarity analysis using Dynamic Time Warping (DTW)
6. Anomaly detection and clustering for authorship decision

All stages are executed sequentially through the main notebook.

Requirements

To operate the system successfully, the user requires:

- Google Colab environment (recommended: GPU enabled runtime, A100 GPU, Colab PRO+ subscribtion, enable High RAM)
- Python execution environment via Google Colab
- Internet access
- Active Google Drive account for data storage
- Access to the project repository and provided notebook (from the github link in the first page)

Repository Location

The system is operated from the Phase B Folder using the main notebook:

- Phase B/Pipeline Setup/Analysis_of_Authorship_in_Arabic_sources.ipynb

All outputs are generated inside the user's Google Drive.

Files Provided

The following files and datasets are required to run the system, users should import them into their Google Drive. The folder path in github is : Phase B/Pipeline Setup. It containes:

- Main notebook containing the full pipeline : Analysis of Authorship in Arabic sources.ipynb
- Folder containing impostor author texts : impostors_data.zip
- Folder containing test texts : test_data.zip
- Index CSV files mapping text identifiers to book names : test_data_table.csv , test_data_table_no_duplicates.csv

Unzip zipped folders in Google Drive Then make a new folder named (capstone) and move all the data to it.

Setup Instructions

1. Open the notebook from the folder (capstone)
2. Select a GPU enabled runtime (A100 GPU , with High RAM)
3. Mount Google Drive
4. Run the notebook cells sequentially from top to bottom

* For the training section we suggest running it on a stronger GPU environment if possible.
* All sections are modular so if at any step the user stoped or the environment crashed , the user can continue on to the next step, since every step saves it outcomes to the user's drive.

System Operation Flow

Step 1 : Text Preprocessing

The preprocessing stage cleans and normalizes Arabic texts.

The system removes noise, normalizes characters, and prepares texts for embedding generation.

Output directories created:

- test_preprocessed /
- impostors_preprocessed /

Sub-step : Fine-Tuning AraBERT

in this sub stage we fine tuned the AraBERT model on our impostor and test data for better embedding results , then saved the fined tuned model to the Google Drive.

Output directory created:

- arabert_finetuned

Step 2 : Embedding Generation

In this stage, the system converts each text segment into a deep contextual embedding using AraBERT.

These embeddings capture both stylistic and semantic information.

Output directory created:

- embeddings_test/

Step 3 : Impostor Pair Training

The system trains Siamese neural networks on all possible impostor pairs.

Each network learns stylistic boundaries between two impostors.

The training process is executed automatically within the notebook.

The training on Colab (GPU A100) would take approximatly ~25 minutes per impostor pair so if you have the access to stronger GPU we suggest running the training there. We used a cloud based GPU service named LAMBDA which provied stronger GPUs online with an hourly payment system. We used NVIDIA's GH200 GPU to train the siamese network and the training time was reduced to ~8 minutes per pair.

Output directory created:

- results/

Step 4 : Stylometric Signal Generation

The trained Siamese models are applied to the test texts.
For each test book, the system generates a stylometric signal representing its stylistic behavior across impostor pairs.
Output subdirectory:

- results/signals/

Step 5 : Similarity Analysis (DTW)

The system computes Dynamic Time Warping distances between stylometric signals.
DTW aligns signals of different lengths and produces distance matrices.
Output directory structure:

- results/dtw/per_pair/<pair_id>/
    - dtw.npy - DTW distance matrix
    - heatmap.png - visual heatmap of the DTW matrix
    - meta.json - metadata describing the DTW run
- results/dtw/aggregate/
    - state.json - global DTW state (book order, indices, impostor pairs)
    - manifest.csv - summary of DTW runs and their execution status

Step 6. Anomaly Detection (Isolation Forest)

This step applies Isolation Forest to the DTW distance matrices to detect stylistic outliers.
Each book receives an anomaly score indicating how atypical its style is across impostor pair comparisons.
Output directory structure:

- results/iforest/per_pair/<pair_id>/
    - scores.npy - normalized anomaly scores per book
    - labels.npy - anomaly labels per book
    - run.csv - per-pair anomaly results
    - scatter.png - PCA-based visualization of anomalies
    - meta.json - metadata for the Isolation Forest run
- results/iforest/aggregate/
    - score_matrix.csv - anomaly scores per book per run
    - score_matrix_avg.csv - averaged anomaly scores per book
    - frequency.csv - anomaly frequency per book
    - frequency_heatmap.png - visualization of anomaly frequency

Step 7. Clustering (K-Means)

This step applies K-Means clustering to the aggregated anomaly scores.
Books are grouped based on stylistic similarity to separate consistent texts from potential outliers.
Output directory structure:

- results/kmeans/aggregate/final/
    - book_clusters_avg.csv - final cluster assignment per book

- kmeans_clusters_scatter_plot_avg.png - visualization of clustered mean anomaly scores
- meta.json - clustering parameters and output references

Google Drive Output Structure

After successful execution of the full pipeline, the system automatically generates a structured set of folders in Google Drive, corresponding to each processing stage.
These folders store cleaned texts, embeddings, trained models, signals, distance matrices, anomaly scores, clustering results, and visualizations as shown in Figure 1, and the structure of the results folder in shown in Figure 2.

Figure 1. Google Drive setup After running the pipeline.

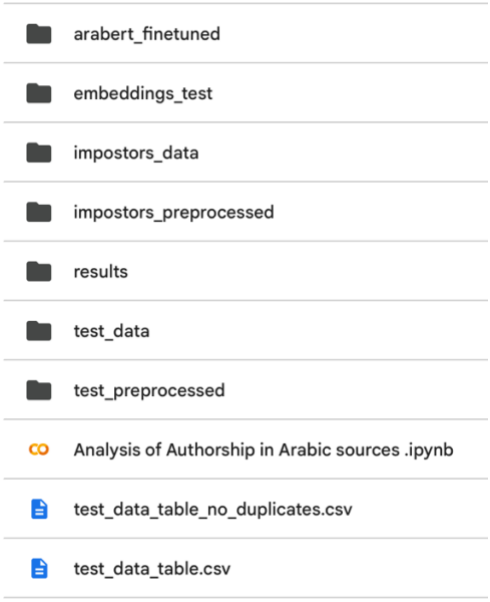
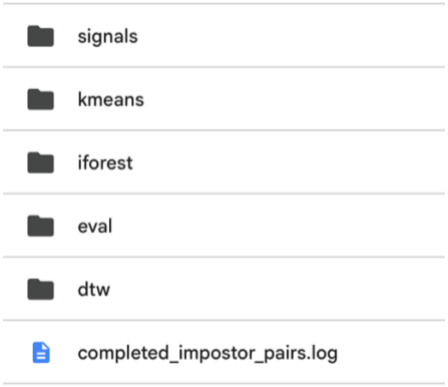


Figure 2. Google Drive structure of the folder (results) in our setup.



In Figure 2, the completed impostor pairs log records the progress of the resumable training process. The evaluation folder contains the final results, including a table that compares the system's predicted classifications with the expected labels. The expected true labels were

determined based on documented scholarly research identifying works of Al-Jahiz as either authentic or potentially disputed.

Maintenance Guide

The purpose of this maintenance guide is to enable continued use of the project after its completion.
It provides guidance for maintainers who wish to apply modifications, updates, or improvements in order to extend the system's lifespan and adapt it to new datasets, authors, or research scenarios.
This guide focuses on maintaining and extending the project specific software components developed in this work.

Project Overview

The system implements an unsupervised authorship verification framework for Arabic texts based on the Deep Impostor methodology.
Arabic texts are transformed into stylometric signals using deep contextual embeddings and Siamese neural networks.
Similarity and anomaly analysis techniques are then applied to determine authorship consistency.
The system is modular and pipeline based, allowing individual stages to be updated independently.

System Operating Environment

Software Environment: The system is designed to run in a Python based notebook environment.
Required components include:

- Python execution environment (Google Colab or compatible Jupyter environment)
- PyTorch for neural network training
- HuggingFace Transformers for Arabic language models
- NumPy and Pandas for data handling
- scikit-learn for anomaly detection and clustering
- DTW libraries for signal comparison

There is no dependency on databases or server side infrastructure such as MySQL or web frameworks.
Hardware Environment

- GPU enabled execution environment is recommended
- High RAM runtime is preferred for large scale experiments
- Cloud based GPU platforms were used during development

The system can execute on CPU only environments, but training and signal generation stages will require significantly more time.

Installation and Setup

Maintenance requires installing only the project's custom software environment.

1. Obtain the project notebook and data directories.
2. Upload all files to Google Drive.
3. Open the main notebook in Google Colab or an equivalent Jupyter environment.

4. Mount Google Drive when prompted.
5. Ensure that directory paths defined in the configuration section of the notebook match the Drive structure.

All required libraries are installed automatically through notebook cells.

Project Structure

The system follows a structured output layout created automatically during execution:

- test_preprocessed / and impostors_preprocessed / - cleaned text outputs
- test_embeddings/ - generated deep embeddings
- results/ - signals, DTW matrices, and anomaly scores
- results/ - clustering outputs and visualizations

Maintainers should preserve this structure to ensure compatibility with downstream processing stages.

Maintenance Guidelines

Updating or Adding New Text Data

- New texts can be added by placing files in the appropriate input folders.
- Index files should be updated to reflect new book identifiers.
- No changes to model architecture or pipeline logic are required.

Changing the Test Author

- Our test author is Al-Jahiz, replace Al-Jahiz's data with another authors data.
- Files of the new author must be segmented into same size files (recommended ~200KB).
- Change file directory to the new authors book folder.
- Change the folder of the impostors to impostors that suits the new author.
- New impostor files must be the same size (recommended ~250KB)]
- Add index files for the test files.

Modifying Model Architecture or Parameters

- Neural network parameters are defined in clearly labeled configuration sections.
- Maintainers may adjust parameters such as:
    - Batch size
    - Learning rate
    - Number of epochs
    - Sequence length

Changes should be validated to ensure compatibility with embedding dimensions and signal construction.

Replacing the Embedding Model

- The embedding stage can be updated to use a different Arabic language model.

- The replacement model must produce fixed size embeddings compatible with the Siamese network.
- No changes are required in DTW or clustering stages if embedding dimensions remain consistent.

Extending Analysis and Classification

- Additional similarity measures or anomaly detection algorithms may be added.
- New result files should follow the existing directory and naming conventions.
- Visualization scripts can be extended without affecting the core pipeline.

Output Management and Reproducibility

All outputs are saved automatically in Google Drive using deterministic file naming.
Maintainers are encouraged to:

- Keep backups of generated results
- Log parameter changes when rerunning experiments
- Avoid overwriting previous experiment outputs when testing modifications

This ensures reproducibility and traceability of results.

Portability and Future Use

The system is portable across cloud and local environments with minimal changes.
Migration requires only path updates and appropriate hardware support.
The modular pipeline design allows reuse of individual components for future authorship verification research.