

Capstone Project Phase A

Analysis of Russian Texts Using Deep Impostor Method

25-2-R-12

By: Daniel Feldman, Arthur Cherniy

Supervisors: Prof. Ze’ev Volkovich, Dr. Renata Avros

[Link to GIT folder.](https://github.com/DanielFeldman1/russian-imposter)

**Table of Contents**

1 Introduction………………………….…………………………………………………………………..3

2 Literature Review……………………………………………………………………………………….4

2.1 The Debate……………………………………………………………………………...……4

2.2 Word Embedding………………………………………………………………..…………..5

2.3 Convolutional Neural Network (CNN)………………………………………………..……6

2.4. Long Short-Term Memory (LSTM)……………………………………………………..….7

2.5. Bidirectional Encoder Representations (BERT)…………………………………………8

2.6. BERT Fine Tuning…………………………………………………………………………11

2.7. DTW Distance………………………………………………………………………..……12

2.8. Isolation Forest Algorithm……………………………………………………………..…13

3. Project…………………………………………………………………………………………………14

3.1. Requirements (FR & NFR)……………………………………………………………….14

3.2. Requirements Gathering………………………………………………………………….16

3.3. Description of Research………………………………………………………………..…17

3.3.1. Text Preprocessing and Tokenization……………………………………...…18

3.3.2. Fine Tuning Russian BERT……………………………………………………19

3.3.3. Fine Tuned Russian BERT Classification……………………………………21

3.3.4. Converting documents into signals…………………………………...………22

3.3.5. Signal comparison and ranking………………………………………….……24

3.3.6. Final Result…………………………………………………………….…..……25

3.4. Expected Challenges………………………………………………………………..……26

3.5. Tools …..……………………………………………………………………………..…….26

3.6. Interface with Client during development……………………………………….………26

3.7. Algorithm & Tests Description……………………………………………………………26

3.8. Success Indicators……………………………………………………………….……..…27

4. Tests Made While Developing, Methods & Tools Used…………………………………..………27

5. Links to AI tools used…………………………………………………………………………...……30

6. Academic References APA format…………………………………………………………………31

**Abstract**

This article explores the question of authorship regarding Mikhail Sholokhov’s works through a method known as the ‘Deep Impostor’ approach, particularly the novel “And Quiet Flows the Don”. This technique involves using a collection of known impostor texts to examine the origins of a specific set of target texts. Both the target texts and the impostor texts are split into an equal number of word segments. A deep neural network either a Convolutional Neural Network (CNN) or a pre-trained BERT transformer is then trained and refined to distinguish between the impostor segments. After this process, each target text is converted into a numerical signal by averaging the segment assignments. The similarity between these signals is measured using the Dynamic Time Warping distance. Subsequently, the Isolation Forest algorithm detects outliers within the target text collection for each impostor pair, assigning appropriate scores to each text under review. In the final step, the analyzed works are grouped into two clusters. When using a CNN-based model, the first cluster consists of fifteen works, which, based on overall analysis, are suggested to not have been authored by Sholokhov. The remaining texts are classified as genuine works by Sholokhov.

**1. Introduction**

The authorship of “And Quiet Flows the Don”, traditionally attributed to M. Sholokhov, has long been a subject of scholarly debate. While Sholokhov received the Nobel Prize in Literature for this work, questions have persisted regarding the novel’s origin. Some researchers have pointed to the remarkable literary maturity and historical accuracy of the text especially given Sholokhov's youth and limited prior output at the time as grounds for re-examining its authorship.

Various theories have emerged over the decades, including the possibility that parts of the manuscript may have originated from another source, potentially lost or unpublished writings from the turbulent period of the Russian Civil War. While no definitive proof has ever resolved the controversy, the debate reflects broader questions about authorship, authenticity, and the complex histories of literary creation.

This project aims to contribute to this ongoing inquiry by applying deep learning methodologies, specifically a deep impostor detection framework to analyze stylistic patterns, manuscript data, and comparative literary texts. The objective is not to make definitive claims or accusations, but to explore the question of authorship through a modern, data-driven lens, offering new insights into one of the most enduring mysteries in 20th-century literature.

**2. Literature Review**

**2.1. The Debate**

The authorship of “And Quiet Flows the Don”, a four-volume novel by M. Sholokhov, published between 1928 and 1940, has been debated since its release. Recognized as a significant work in 20th-century Russian literature and earning Sholokhov the 1965 Nobel Prize in Literature, the novel’s attribution has faced allegations of plagiarism, with Fyodor Kryukov, a Cossack writer who died in 1920, often proposed as a potential author. This review examines the primary arguments, evidence, and scholarly analyses surrounding the controversy, drawing on Donald Ostrowski’s “Who Wrote That? Authorship Controversies from Moses to Sholokhov” (2021) and related studies.

Ostrowski (2021) contextualizes the debate within literary attribution disputes, noting that questions arose in 1928 after the publication of the novel’s first two volumes, when Sholokhov was 23. Some questioned whether a young writer with limited formal education could produce a work of such historical and stylistic complexity. Allegations suggested Sholokhov may have used a manuscript by Kryukov, whose knowledge of Cossack life matched the novel’s depiction of the Don Cossack region during World War I and the Russian Civil War. A 1929 commission, led by Alexander Serafimovich, found no evidence of plagiarism, concluding that the manuscript’s style aligned with Sholokhov’s earlier work, “Tales from the Don” (1926).

The debate continued, notably in the 1960s, when Aleksandr Solzhenitsyn questioned Sholokhov’s authorship. Solzhenitsyn argued that the novel’s portrayal of anti-Bolshevik Whites was inconsistent with Sholokhov’s Communist affiliations, supporting the Kryukov hypothesis. Ostrowski emphasizes the need for textual evidence over political considerations in evaluating these claims.

Quantitative analyses have played a significant role in addressing the debate. In 1984, Geir Kjetsaa and colleagues analyzed sentence lengths in “And Quiet Flows the Don”, comparing it to works by Sholokhov and Kryukov. Their study, published in “The Authorship of The Quiet Don”, found Sholokhov’s stylistic patterns consistent with the novel, unlike Kryukov’s. A 2007 study by Nils Lid Hjort reinforced these findings through prose analysis. Additionally, Sholokhov’s manuscripts, discovered in the 1980s and analyzed by the Russian Academy of Science in 1999, provided evidence of his creative process, with Felix Kuznetsov’s study supporting his authorship.

Counterarguments persist. In the 2000s, Zeev Bar-Sella suggested the manuscripts were written no earlier than 1929 and proposed Viktor Sevsky as the author, based on textual analysis. A 2020 study by Marina Iosifyan and Igor Vlasov used information-based similarity analysis and Burrows’s Delta method, finding Kryukov’s writings distinct from the novel but noting some stylistic differences. Ostrowski notes that these studies often lack the empirical rigor of Kjetsaa’s and Hjort’s analyses.

Sholokhov’s historical context adds complexity. As a Communist Party member and 1941 Stalin Prize recipient, his Soviet affiliations led some to question the novel’s nuanced depiction of Cossack life and occasional sympathy for the Whites. Ostrowski highlights Sholokhov’s Don Cossack background and interactions with Soviet authorities, suggesting he had the knowledge and confidence to portray complex political dynamics.

In conclusion, the authorship debate involves textual evidence, statistical analysis, and historical context. Ostrowski (2021), supported by Kjetsaa’s studies, manuscript evidence, and Kuznetsov’s analysis, argues for Sholokhov’s authorship. While skeptics raise questions about Sholokhov’s age and affiliations, their arguments often rely on less robust evidence. Future research, including further manuscript analysis and stylistic studies, could provide additional clarity to this ongoing discussion.

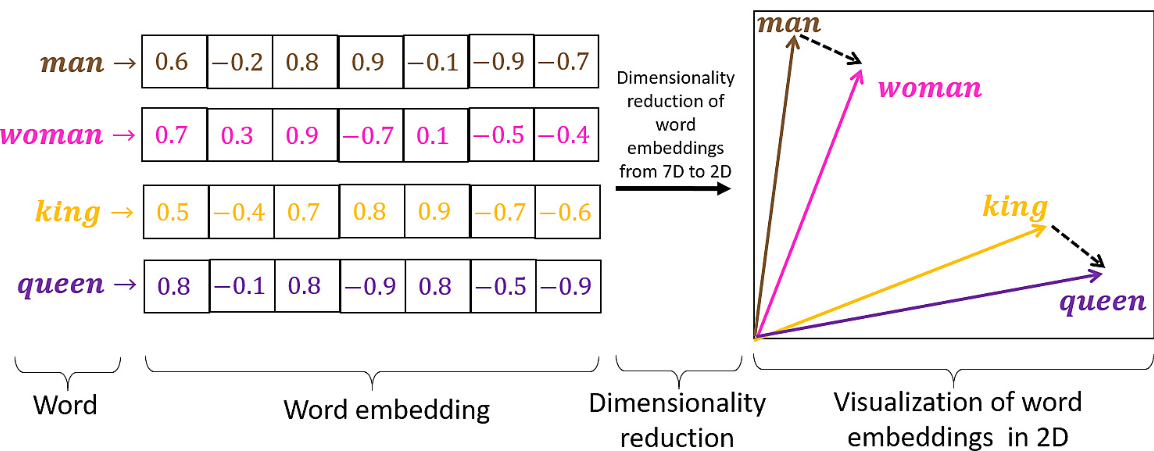
**2.2. Word Embedding**

A word embedding represents words as vectors in a multidimensional space, where the distance and direction between vectors reflect how closely related words are in meaning. These dense vectors, composed of continuous values, are trained using machine learning techniques, typically neural networks, on large text corpora, allowing the model to learn from word usage in context.

A popular method for generating word embeddings is Word2Vec, which trains a neural network to predict a word based on its surrounding words or vice versa, depending on the model variant.

Embeddings are essential for natural language processing (NLP) tasks, enabling models to understand language more effectively than traditional methods. They power applications like text classification, named entity recognition, translation, search, question answering, clustering, and text generation.

Since machine learning models can't process raw text, embeddings provide a way to convert language into numerical form. Each word's vector encodes patterns in how it's used, placing similar words closer together in the vector space. The training process adjusts the model to minimize prediction errors based on surrounding context, shaping a representation space that captures linguistic nuance.



**Figure 1.** Visualization of word embedding in 2d space.

**2.3. Convolutional Neural Network (CNN)**

A Convolutional Neural Network (CNN) is a type of artificial neural network designed to process and analyze structured grid-like data, such as images or time-series data. It uses a mathematical operation called convolution, which involves sliding a small filter (or kernel) over the input data to extract relevant features like edges, textures, or patterns. These features are then passed through multiple layers, including pooling layers to reduce dimensionality and fully connected layers for classification or prediction.

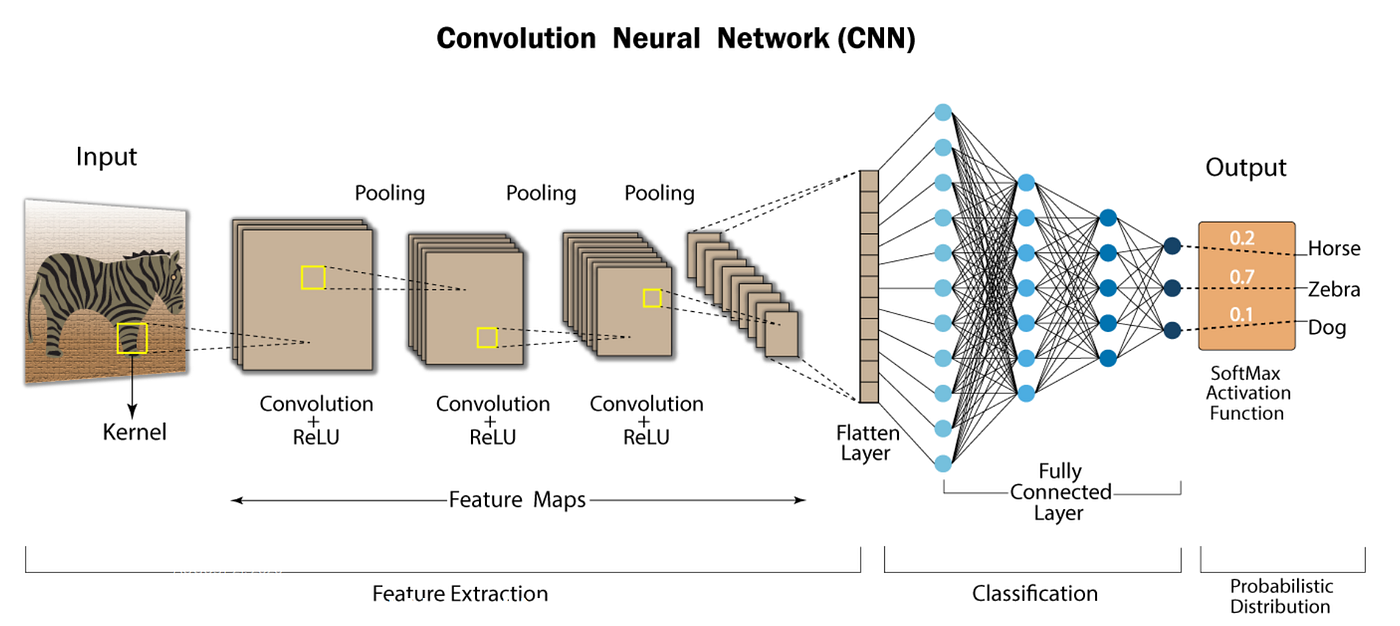
**Convolutional Layer**

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width and depth which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

**Pooling Layer**

Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.

**Fully-Connected (FC) Layer**

The name of the fully-connected layer aptly describes itself. In the fully-connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

**Figure 2.** Illustration of a CNN.

**2.4. Long Short-Term Memory (LSTM)**

A Long short-term memory (LSTM) is a type of Recurrent Neural Network specially designed to prevent the neural network output for a given input from either decaying or exploding as it cycles through the feedback loops. The feedback loops are what allow recurrent networks to be better at pattern recognition than other neural networks. Memory of past input is critical for solving sequence learning tasks and Long short-term memory networks provide better performance compared to other RNN architectures by alleviating what is called the vanishing gradient problem.

LSTMs due to their ability to learn long term dependencies are applicable to a number of sequence learning problems including language modeling and translation, acoustic modeling of speech, speech synthesis,speech recognition, audio and video data analysis, handwriting recognition and generation, sequence prediction, and protein secondary structure prediction.

**LSTM Architecture**

LSTM architecture involves the memory cell which is controlled by three gates: the input gate, the forget gate and the output gate. These gates decide what information to add to, remove from and output from the memory cell.

Input gate: Controls what information is added to the memory cell.

Forget gate: Determines what information is removed from the memory cell.

Output gate: Controls what information is output from the memory cell.

This allows LSTM networks to selectively retain or discard information as it flows through the network which allows them to learn long-term dependencies. The network has a hidden state which is like its short-term memory. This memory is updated using the current input, the previous hidden state and the current state of the memory cell.

**2.5. Bidirectional Encoder Representations (BERT)**

BERT, or Bidirectional Encoder Representations from Transformers, is an open-source machine learning framework developed by Google for natural language processing (NLP). Unlike traditional language models that process text in one direction (left-to-right or right-to-left), BERT is bidirectional, meaning it analyzes the entire context of a sentence by looking at words before and after a given word simultaneously. This is made possible by its transformer architecture, which uses a self-attention mechanism to understand relationships between words. BERT is pre-trained on vast amounts of text using two tasks: predicting masked words in a sentence and determining if one sentence follows another. This pre-training allows BERT to be fine-tuned for various NLP tasks, such as question answering and sentiment analysis, making it a versatile and powerful tool for understanding human language.

**Pre-training BERT**

The success of BERT largely stems from its novel pre-training strategy, which employs two unsupervised learning objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). These tasks are designed to encourage the model to learn rich, bidirectional representations of text, capturing both word-level semantics and inter-sentential relationships.

**Masked Language Modeling (MLM)**

The Masked Language Modeling task enables BERT to learn deep contextual word representations. During training, a random 15% of tokens in each input sequence are selected for potential masking. Of these selected tokens: 80% are replaced with the special [MASK] token, 10% are replaced with a randomly chosen word from the vocabulary, 10% are left unchanged.

The model is then trained to predict the original identity of these masked tokens based on their surrounding (left and right) context. For example, given the input:

"The cat sat on the [MASK].",

BERT is expected to predict the missing word: "mat".

This differs fundamentally from traditional language models that predict the next word in a sequence (e.g., "The cat sat on the…"), which inherently limits them to a unidirectional view. By predicting masked words in bidirectional contexts, BERT captures richer syntactic and semantic relationships.

**Next Sentence Prediction (NSP)**

BERT is given pairs of sentences and trained to predict whether the second sentence logically follows the first. This helps in understanding relationships between sentences.

**Text Embedding with BERT**

One of the core contributions of BERT is its ability to generate rich, contextualized text embeddings. Unlike traditional word embedding models such as Word2Vec or GloVe which produce static, context-independent vectors BERT generates dynamic embeddings that vary depending on the surrounding words, enabling more nuanced and accurate representations of language.

**Tokenization and Input Representation**

Before generating embeddings, BERT tokenizes the input text using a subword tokenization strategy called WordPiece. This allows it to represent rare or complex words as compositions of more common subword units. For instance, the word “playing” may be split into “play” and “##ing”. Each token is then mapped to three embeddings:

* Token embeddings: the core representation of each word or subword token.
* Segment embeddings: used to differentiate between two input sentences in a pair.
* Positional embeddings: added to capture the order of tokens, since the Transformer model is not inherently sequential.

These embeddings are summed and passed as input to BERT's encoder stack.

**Contextual Embedding via Transformer Encoders**

BERT’s encoder consists of multiple layers, each containing self-attention and feed-forward sublayers. Through multi-head self-attention, BERT models the relationships between all tokens in the input, regardless of their positions. This enables each token’s representation to be updated iteratively, informed by its full left and right context. The result is a sequence of contextual embeddings one for each token that capture the token’s meaning in its specific usage.

For example, in the sentences: “He went to the bank to fish.” and “He went to the bank to deposit money.”, the token “bank” will have different vector representations, influenced by the surrounding context (“fish” vs. “deposit money”).

**Types of Embeddings Produced**

BERT produces two main types of embeddings useful for downstream tasks:

* Token-level embeddings: Each token in the input has a contextualized embedding derived from the final hidden layer. These are commonly used in token-level tasks such as named entity recognition or part-of-speech tagging.
* Sentence-level embedding ([CLS] token): BERT adds a special classification token [CLS] at the beginning of each input sequence. The final hidden state corresponding to this token is typically used as a fixed-length representation of the entire sentence or input sequence. It is especially useful in sentence-level tasks such as classification, semantic similarity, or entailment detection.

**Russian BERT**

There are several BERT-based models specifically developed for the Russian language. These models are tailored to capture the linguistic nuances of Russian and have been trained on large Russian corpora. In our work we explore

**ruBERT by DeepPavlov**

Developed by the DeepPavlov team, ruBERT is Russian-language BERT model, based on the original BERT architecture. It was trained on the Russian part of Wikipedia and news data, utilizing a vocabulary of Russian sub tokens. The model has been adapted for various downstream tasks such as classification, tagging, question answering and ranking.

**RuBioRoBERTa by Alex Yalunin**

RuBioRoBERTa is a BERT-based model designed for Russian biomedical text mining. It was pre-trained on a corpus of freely available Russian biomedical texts and has demonstrated state-of-the-art results on RuMedBench, a benchmark for Russian medical language understanding tasks, including text classification, question answering, natural language inference, and named entity recognition.

**2.6. BERT Fine Tuning**

Fine-tuning BERT involves adapting a pre-trained BERT model to a specific task by training it further on a smaller, task-specific dataset. While BERT is pre-trained on vast amounts of general text data using self-supervised learning, fine-tuning tailors its capabilities to specialized tasks like sentiment analysis, question answering, or domain-specific language processing (e.g., legal or medical text). This process adjusts the model's weights to optimize performance for the target task, typically using a small learning rate (e.g., 2e-5 to 5e-5) to retain pre-trained knowledge while incorporating task-specific features. Fine-tuning is essential because pre-trained BERT lacks domain-specific vocabulary and task objectives, making this step crucial for achieving high accuracy in downstream applications.

**Fine-Tuning BERT for Authorship Attribution via Linguistic Style**

In this project, we fine-tune the BERT model to perform authorship attribution by capturing individual linguistic styles. BERT’s deep contextual embeddings, learned through large-scale language modeling objectives, provide a robust foundation for identifying subtle stylistic patterns across texts. This makes BERT particularly well-suited for tasks involving nuanced language variation, such as distinguishing between authors.

To adapt BERT to our classification task, we draw inspiration from *Comprehension of the Shakespeare Authorship Question Through Deep Impostors Approach* by Volkovich and Avros (2020). In their work, a pre-trained BERT model was fine-tuned for sentiment classification by appending a task-specific output layer and training it on a dataset labeled with positive and negative sentiment. This approach enabled the model to differentiate between opposing textual characteristics.

We adopt a similar strategy by reframing authorship attribution as a binary classification problem. Specifically, batches of text from one impostor group are labeled as “positive,” while batches from another are labeled as “negative.” Although this labeling strategy originates from sentiment analysis, it is repurposed here to emphasize contrasts in linguistic style between groups. The goal is for the fine-tuned BERT model to internalize these stylistic distinctions and generalize them to new, unseen text samples drawn from the target collection.

**2.7. DTW Distance**

Dynamic Time Warping (DTW) distance, is a measure used to compute the similarity between two time series by finding an optimal alignment that minimizes the total distance between their points, even if they vary in timing or length.

The first step in DTW involves constructing a distance matrix between the two sequences. Each matrix element represents the distance (typically Euclidean) between corresponding points in the two sequences.

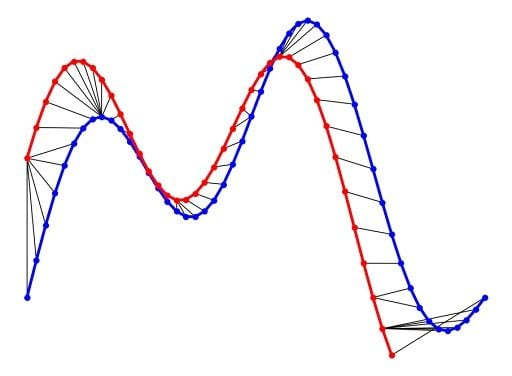
*C(i,j) = distance(ai,bj) = |ai - bj|*

Next, a cost matrix is created by accumulating the minimum distances from the start of the sequences to the current point. This accumulated cost represents the optimal path's cumulative distance up to that point.

*D(i,j) = C(i,j) + min {D(i - 1,j), D(i,j - 1), D(i - 1,j - 1)}*

The optimal alignment path is found by tracing back from the last element in the cost matrix to the first element. This path represents the best alignment between the two sequences, minimizing the total distance.

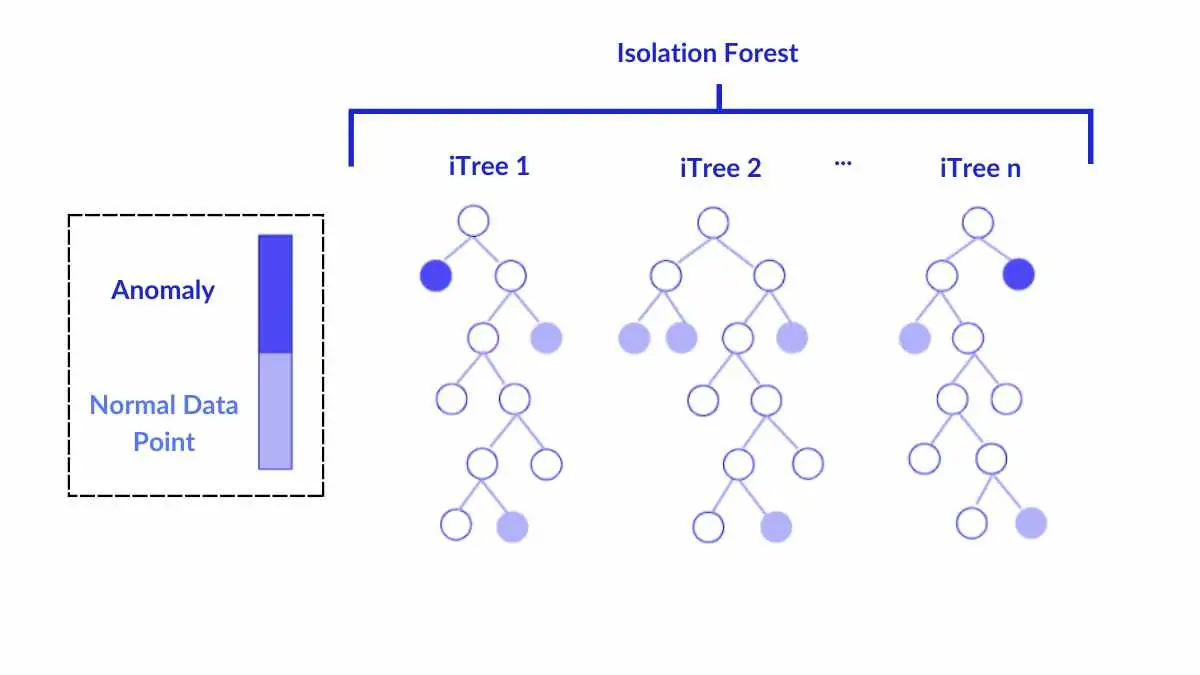
The warping path shows how one sequence can be warped (stretched or compressed) along the time axis to match the other sequence best.



**Figure 3.** Illustration of DTW distance between two time series.

**2.8. Isolation Forest Algorithm**

Isolation Forest is an unsupervised machine learning algorithm for anomaly detection, it isolates data points by constructing a set of isolation trees through random partitioning. It works by repeatedly selecting a random feature and a random split value within that feature’s range, recursively dividing the dataset until each point is isolated or a limit is reached. Anomalies, being rare and distinct, require fewer splits to isolate and thus have shorter average path lengths across the trees, while normal points in denser regions take longer to separate. This approach, which doesn’t rely on labeled data, makes Isolation Forest efficient and effective for identifying outliers in high-dimensional datasets.



**Figure 4.** Illustration of Isolation Forest Algorithm.

**3. Project Description**

This project explores the application of machine learning techniques to authenticate authorship of literary texts written in a foreign language, specifically, Russian. Utilizing the Deep-Impostor approach introduced by Avros and Volkovich (2025), the study aims to provide data-driven insights into the longstanding authorship debate surrounding the works attributed to M. Sholokhov.

**3.1. Requirements**

**Functional Requirements**

1. The system shall classify M.Sholokhov’s works to suspected impostors and genuine works.
2. The system shall accept a collection of impostor texts (works not authored by M. Sholokhov).
3. The system shall split texts into equal sized word segments.
4. The system shall tokenize and clean input Russian text using lowercasing and punctuation removal.
5. The system shall fine tune a Russian BERT model to distinguish among impostor text segments.
6. The system shall embed text segments by using a Fine Tuned Russian BERT model.
7. The system shall classify text segments using a Convolutional Neural Network (CNN) to determine their similarity to either impostor A or impostor B.
8. The system shall turn segments to chunks by averaging the classification results across every 8 segments.
9. The system shall convert each target text into a numerical signal by concatenating scores across its chunks.
10. The system shall compute the Dynamic Time Warping (DTW) distance between signals of target texts for each impostor pair.
11. The system shall apply the Isolation Forest algorithm to detect outlier target texts based on DTW distances.
12. The system shall cluster target texts into two groups: suspected impostors and genuine works.
13. The system shall process texts in the Russian language

**Non-Functional Requirements**

1. Performance
   1. The system shall process and classify a corpus of 20 texts within 1.5 hours.
2. Accuracy and Reliability
   1. The system shall produce consistent results when run on the same dataset under identical conditions.
   2. The system shall achieve at least 85% classification accuracy in distinguishing between impostor text segments.
3. Usability
   1. The system shall execute the entire process with the press of a single button.
   2. The system shall provide a summary of classification results, including visualizations (e.g., cluster plots, DTW graphs) to support human interpretation.
4. Security
   1. The system shall restrict codebase access to authorized users only.
   2. All data handled by the system shall be stored in a secure environment (Google Drive).
5. Maintainability
   1. The system shall be modular, allowing easy updates or replacement of the embedding model (e.g., swapping BERT with Word2Vec).
   2. The system codebase shall include documentation and unit tests for all major components.
6. Portability
   1. The system shall be deployable on Linux, macOS, and Windows operating systems.
7. Testability
   1. At least 80% of the codebase shall be covered by unit tests.

**3.2. Requirements Gathering**

The system requirements were gathered by studying the research paper by Avros and Volkovich (2025) and the Deep Impostor method described. Additional research was conducted on relevant topics, including BERT, CNN, and DTW distance, to ensure a thorough understanding. The system was designed at a high level and divided into modules, each representing a processing stage for the text data. This involved analyzing how BERT functions and its preprocessing needs, understanding CNNs and their reliance on word embeddings, and recognizing the need for a DTW distance module and text-to-signal conversion based on the paper’s methodology. The limitations of the development platform, Google Colab, were also evaluated to ensure compatibility.

During routine meetings, Z. Volkovich emphasized the importance of native support for Russian-language texts, given that the disputed works were originally written in Russian. In response to this feedback, the requirements expanded to ensure full compatibility with Russian input, including appropriate pre-processing, tokenization and the use of language-specific models tailored to Russian.

**3.3. Description of Research**

The analysis centers on texts attributed to M. Sholokhov, referred to as the target texts throughout the study. These texts are examined by comparing their linguistic features to those of other writers, referred to as impostors, in order to isolate stylistic patterns unique to Sholokhov.

For each selected pair of impostors, a fine-tuned version of the Russian BERT model is trained to distinguish between their respective writing styles. This model is then used to assess how similar each segment of a target text is to either impostor. The target texts are divided into segments, each of which is classified using the fine-tuned BERT model to determine its stylistic alignment. These segment-level predictions are then aggregated to form a signal that represents the overall similarity of the target text to the impostor pair under consideration.

iterate over **pairs** of imposter texts.

1. Divide the texts into segments of L words (batches)
2. Loop on impostors pairs:
   1. Preprocess texts.
      1. Lowercase text
      2. Remove special characters to keep only Russian letters.
      3. Tokenize texts using Russian BERT tokenizer.
   2. Train the model to distinguish between the two current impostor texts.
   3. Feed tested texts into the model and get a score/label for each batch (How similar the writing style is to imposter A or imposter B, 0 and 1).
   4. Every k batches are concatenated into a chunk with averaging the batch scores.
   5. The chunk scores are concatenated to create a signal.
   6. For each target creation the Dynamic Time Warping is calculated distance from the other target texts.
   7. Anomaly score is detected by Isolation Forest Alg. for each one of the texts.
   8. Save the obtained anomaly score for each text.
3. The accumulated anomaly scores are clustered into two clusters, aiming to recognize the genuine texts.

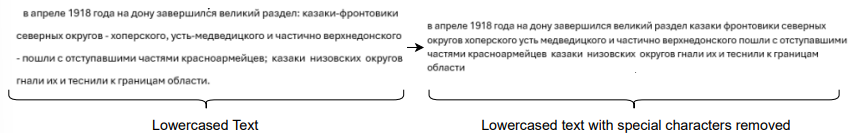
**3.3.1. Text Preprocessing and Tokenization**

Prior to being input into the Russian BERT model, all textual data undergoes a preprocessing pipeline to ensure compatibility with the model’s requirements. As the variant of Russian BERT used is case-sensitive, all text is first converted to lowercase to prevent the model from interpreting identical words in different cases (e.g., Дону vs. дону) as distinct tokens.



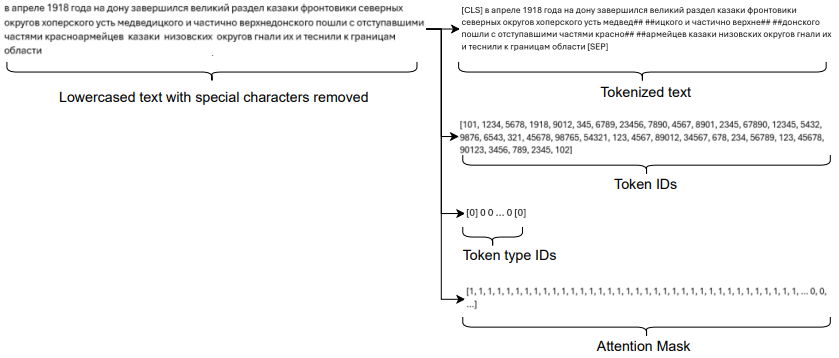
**Figure 5.** Illustration of text lowercasing.

Following lowercasing, all special characters that are not part of the Russian alphabet (e.g., punctuation marks such as colons, hyphens, and periods) are removed. This step reduces noise and focuses the model on meaningful linguistic content.



**Figure 6.** Illustration of removal of special characters.

The cleaned text is processed by the Russian BERT tokenizer, which divides it into subword tokens and inserts [CLS] and [SEP] tokens to denote the start and end of each input sequence. These tokens are mapped to numerical token IDs for model input. An attention mask is generated to differentiate actual tokens from padding, accommodating sequences below the model’s 512-token limit. Token type IDs are also created to specify which tokens belong to each sentence, facilitating BERT’s handling of multi-sentence inputs. In our study, each segment of L words is treated as a single sentence for BERT processing.



**Figure 7.** Illustration of Russian BERT Tokenizer outputs.

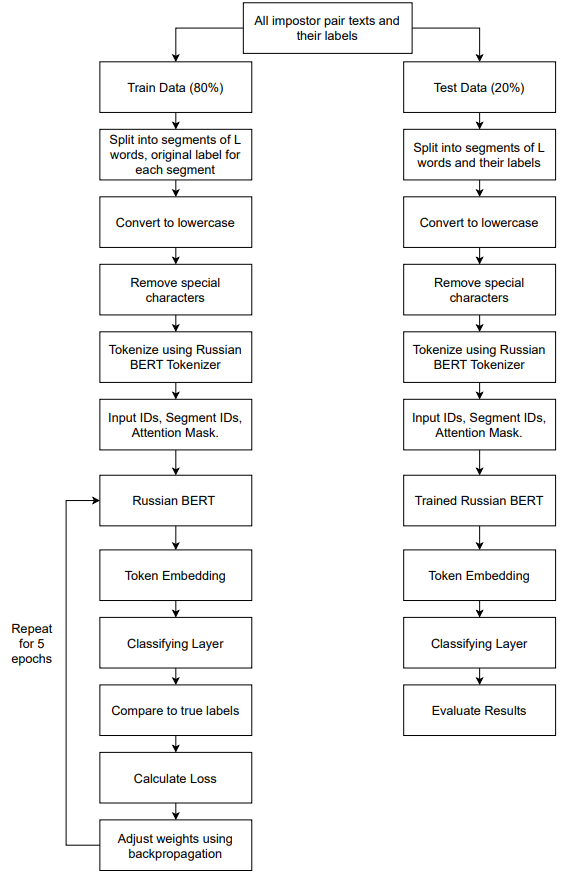
**3.3.2. Fine Tuning Russian BERT to distinguish between impostors**

In each iteration, a pair of impostors is selected, and the pre-trained Russian BERT model is fine-tuned to distinguish between their respective writing styles. This fine-tuning process adapts the general-purpose language model to the specific binary classification task of identifying whether a given text was written by impostor A or impostor B.

The procedure resembles standard supervised training of machine learning models. A labeled dataset containing text samples from both impostors is constructed and split into training and testing subsets. Each text sample is preprocessed as previously described and fed into the BERT model, which generates contextual embeddings. These embeddings are passed through an added classification head consisting of a fully connected layer and a binary output layer. The predicted class labels are then compared with the true labels using a binary cross-entropy loss function.

Backpropagation is employed to update the weights of both the BERT model and the classification layer. The Adam optimizer is used to perform the gradient updates. This training process is repeated for a predefined number of epochs.

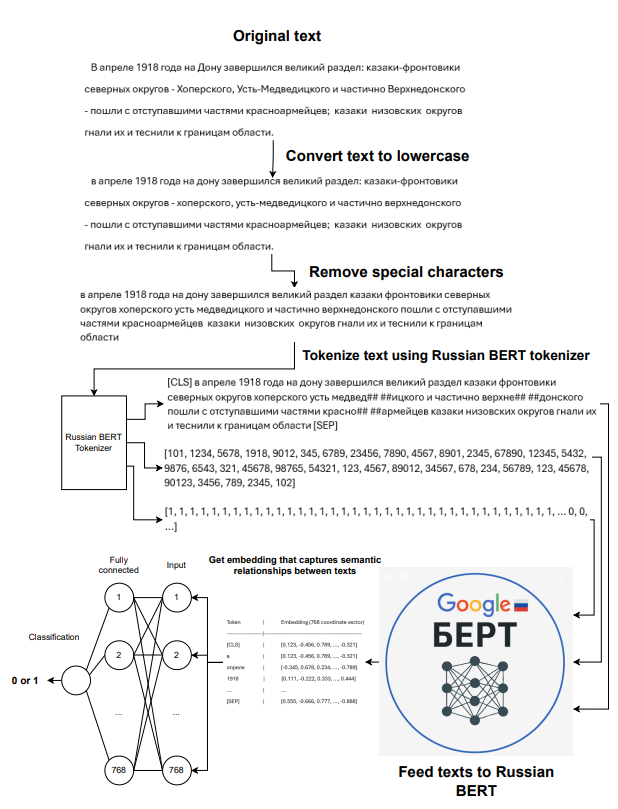
After training, the model's performance is evaluated on the held-out test set. The predicted labels for the test samples are compared to the ground truth labels, and the classification accuracy is computed to assess the model’s effectiveness in distinguishing between the two impostors.

****

**Figure 8.** Flowchart of BERT Fine Tuning Process.

**3.3.3. Fine Tuned Russian BERT Classification**

Since the base BERT model generates contextualized embeddings rather than direct classifications, a custom classification head is appended to enable binary decision making. Specifically, a fully connected (dense) layer is added on top of the BERT output, taking as input the 768-dimensional embedding corresponding to the [CLS] token. This is followed by a binary classification layer that outputs the probability of the input segment being more stylistically similar to either impostor A or impostor B. Through this modification, the pre-trained Russian BERT model is adapted for the specific binary classification task required in the authorship verification process.



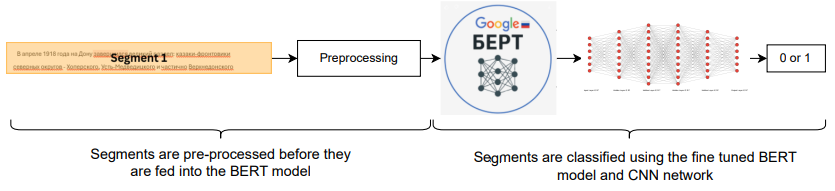
**Figure 9.** Illustration of Russian BERT classification process.

**3.3.4. Converting documents into signals**

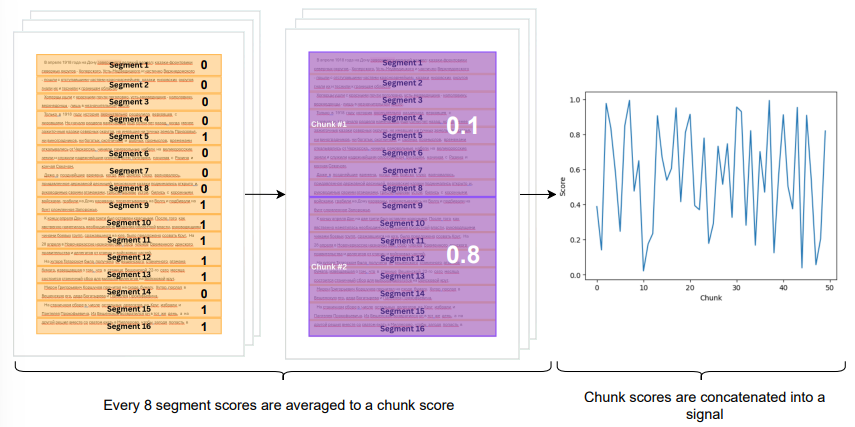
To convert a given text by M. Sholokhov into a usable and comparable signal, the text is first divided into segments of L words.

**Figure 7.** Illustration of text division into segments.

Each segment undergoes preprocessing as previously described and is then input into the fine-tuned Russian BERT model. The fine tuned model classifies each segment by assessing its stylistic similarity to one of two impostors labeling it as more similar to impostor A or to impostor B.

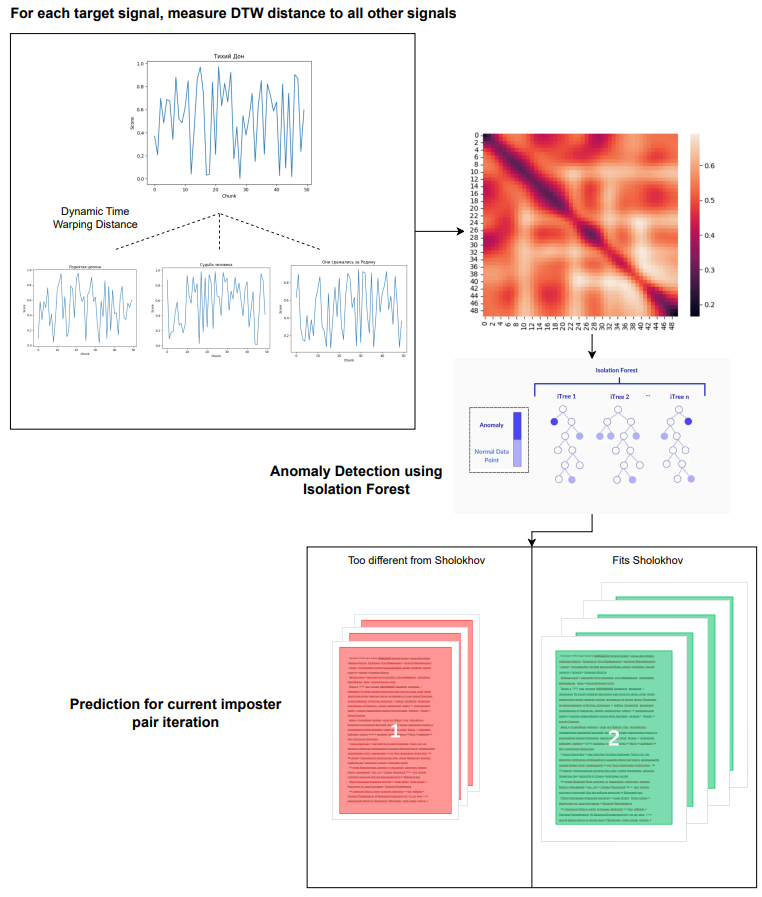
**Figure 8.** Illustration of segment to classification process.

Subsequently, every eight consecutive segments are grouped into a chunk by averaging the model’s predictions across those segments. These chunk-level scores are then concatenated sequentially to construct the final signal representation of the text.

**Figure 9.** Illustration of segment score averaging to chunks and concatenation of chunk scores to a signal.

**3.3.5. Signal comparison and ranking**

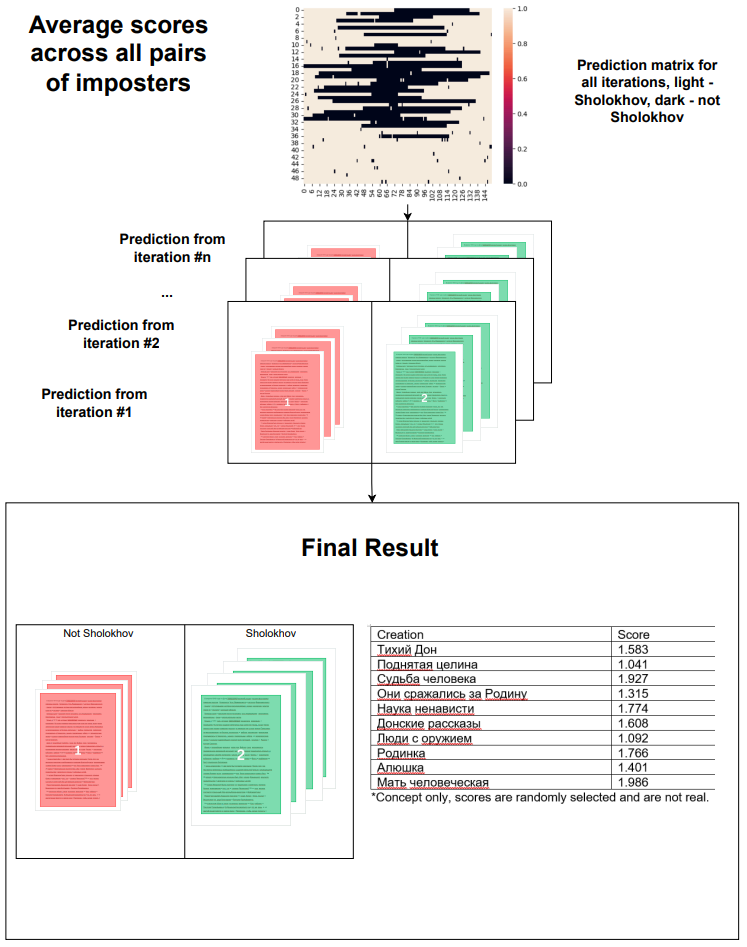
Each text authored by M. Sholokhov is transformed into a signal representation for every selected impostor pair, following the method described earlier. In each iteration, a single impostor pair is selected. For each signal derived from the text, the Dynamic Time Warping (DTW) distance is computed against all other signals, and the results are stored in a distance matrix. Subsequently, the Isolation Forest algorithm is applied to detect outlier signals whose DTW distances significantly deviate from the rest suggesting a low probability of being authored by the same individual. While it is acknowledged that an author's writing style may evolve over time or be influenced by external factors, it is assumed that their core linguistic characteristics and semantic patterns remain relatively consistent.



**Figure 10.** Illustration of the target text classification process in each impostor pair iteration.

**3.3.6. Final Result**

For the final evaluation, classification results from all iterations and all target texts are aggregated. Specifically, for each target text, the classification scores obtained across all impostor pairs are averaged, yielding a final score ranging between 1 and 2. A threshold of 1.5 is applied for authorship attribution: texts with an average score above 1.5 are considered to have been written by Sholokhov, while those with scores below 1.5 are considered unlikely to be authored by him.



**Figure 11.** Illustration of final result extraction for each text after all iterations.

**3.4. Expected Challenges**

The challenges we expect to face are locating the entire corpus of M.Sholokhov, converting all different work files into usable text, integration of existing Russian BERT tokenizer and model into the project. We intend to overcome these challenges by using more research, trial and error and constructing our own modules for file conversion or using an external site.

**3.5. Tools**

The tools used in the project are:

* Google colab - for writing the project code.
* AI chatbots - as learning sources and work refinement tools.
* Google drive - file storage (M. Sholokhov Corpus, impostor texts, codebase).

**3.6. Interface with Client during development**

Our clients are the researchers which have tried to give an answer to the authorship debate, therefore there is no direct interface with them, so we interact with them and learn their needs through their published research papers.

**3.7. Algorithms/Testing Process Description**

To evaluate the complete method, a set of texts with known authorship is used to verify that the program classifies them properly. Accuracy scores for the model will be measured, with a target of achieving an accuracy of at least 85% in distinguishing impostor text segments.

To confirm reliability, the system will be tested multiple times under the same conditions to ensure it produces consistent results. The classification process will be benchmarked to ensure it can process and analyze a corpus of 20 texts within 1.5 hours, satisfying performance requirements. Visual outputs such as DTW graphs and clustering plots will be generated automatically to support result interpretation.

Security will be upheld by restricting access to the codebase and storing all data in a secure environment, such as Google Drive with controlled permissions.

Maintainability will be supported through modular code design, enabling easy updates (e.g., swapping BERT with Word2Vec), with full documentation and unit tests for core components.

Finally, testability will be enforced with a robust suite of unit and integration tests to ensure at least 80% code coverage.

**3.8. Success Indicators**

The indicators chosen for success are:

1. Above 85% accuracy in impostor distinguishing.
2. Developing a program capable of efficiently managing the task of clustering large volumes of text.
3. Successfully implementing the deep impostor method with Russian text.
4. Results produced by the program show at least partial alignment with conventional scholarly views on the authorship of the texts.

**4. Testing Process**

**4.1 Tests:** To test the project, the program is divided into multiple modules, each module is tested individually, for different types of inputs and scenarios:

* Preprocessing:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter Russian text into the preprocessing module. | Module returns the preprocessed Russian text. |
| 2. | Enter Hebrew text into the preprocessing module. | Module throws an exception asking only to enter Russian letters. |
| 3. | Enter an empty string into the preprocessing module. | Module throws an exception asking to enter an input with letters. |
| 4. | Enter an integer into the preprocessing module. | Module throws an exception asking to only enter Russian letters. |

* Tokenization and Embedding:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter preprocessed Russian text into the Tokenization module. | Module returns the token array, attention mask and token IDs. |
| 2. | Enter unprocessed Russian text into the Tokenization module. | Module throws an exception asking only to enter preprocessed Russian text. |
| 3. | Enter an empty string into the Tokenization module. | Module throws an exception asking only to not enter empty strings. |
| 4. | Enter an integer into the preprocessing module. | Module throws an exception asking only to enter preprocessed Russian text. |
| 5. | Enter preprocessed Hebrew text into the Tokenization module. | Module throws an exception asking only to enter preprocessed Russian text. |

* Classification:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter valid embedding (vector) array. | Module returns a classification, 0 or 1. |
| 2. | Empty embedding array | Module throws an exception asking for more dimensions. |
| 3. | Enter a string into the Classification module. | Module throws an exception asking for a valid embedding array. |

* DTW Distance Calculation:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter 2 valid signals into the DTW Distance Calculation Module. | Module returns a scalar representing the dtw distance of signal 1 to signal 2. |
| 2. | Enter 1 signal and 1 null into the DTW Distance Calculation Module. | Module throws an exception asking to only enter a pair of 2 signals. |
| 3. | Enter a signal with a coordinate higher than 1 and 1 valid signal. | Module throws an exception mentioning that an entered signal has an invalid score. |
| 4. | Enter 2 strings into the DTW Distance Calculation Module | Module throws an exception asking to only enter valid signals. |

* Anomaly Detection:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter a valid distance matrix. | Module returns a valid array of labels. |
| 2. | Enter an empty matrix. | Module throws an exception asking to only enter a matrix with data. |
| 3. | Enter a string. | Module throws an exception asking to only enter a valid distance matrix. |

* Final Classification:

| # | Test Description | Expected Outcome |
| --- | --- | --- |
| 1. | Enter a valid label matrix | Module returns a valid array of labels. |
| 2. | Enter an empty matrix. | Module throws an exception asking to only enter a matrix with data. |
| 3. | Enter a string. | Module throws an exception asking to only enter a valid labeling matrix. |

**4.2 Methods:** The methods used to test the program are Unit Testing, Module Testing, Integration Testing, System Testing.

**4.3 Tools:** The tools used in the testing phase are unittest, unittest.mock, pytest, coverage, subprocess, tempfiles libraries in python.

**5. AI Tools**

To support research, understanding, and refinement throughout the project, a range of AI tools were used. These tools provided different perspectives, cross-validation of ideas, summarization, and learning assistance. The outputs were studied, compared, and incorporated to enhance the quality of the work.

ChatGPT - <https://chatgpt.com/>

Used for clarifying technical concepts (e.g., CNN, BERT, Isolation Forest), refining wording, generating academic phrasing. Also helped rephrase and polish final report sections.

“Break down the process of this algorithm step by step.”

“Help me understand how Convolutional Neural Networks (CNNs) work internally, especially for text classification.”

“Refine my wording and sentence structure to sound more academic and professional.”

Google Gemini - <https://gemini.google.com/app>

Provided cross-verification of explanations and offered alternative phrasing. Assisted in identifying gaps in analysis.

“Help me find and summarize relevant research papers on authorship attribution using deep learning.”

“Explain how to handle tokenization and preprocessing for Russian-language NLP tasks.”

Grok - <https://x.ai/>

Explored for perspective diversity; useful for generating concise definitions, comparing opinions, and summarizing external sources.

“Compare CNN and BERT in the context of stylometry or authorship attribution.”

“Help me find a Russian-focused BERT model

“What would be the time complexity of this project?”

Perplexity - <https://www.perplexity.ai/>

Used for sourcing and reviewing related academic papers and background research. Particularly helpful in summarizing scholarly literature and directing to original sources.

"AI models for detecting literary plagiarism"

"Please provide several academic papers discussing the authorship of literary works”

“Find academic works that focus on Convolutional Neural Networks (CNNs) and their application in classification methods”

**6. Academic References APA format**

1. 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
2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://www.bioinf.jku.at/publications/older/2604.pdf
3. Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv. https://arxiv.org/abs/1408.5882
4. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (pp. 413–422). IEEE. https://ieeexplore.ieee.org/document/4781136
5. Müller, M. (2007). Dynamic time warping. In Information Retrieval for Music and Motion (pp. 69–84). Springer. https://www.researchgate.net/publication/220660345
6. Ostrowski, D. (2020). Who wrote that? Authorship controversies from Moses to Sholokhov. Cornell University Press.
7. Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune BERT for text classification? arXiv.

https://arxiv.org/abs/1905.05583

1. Suthaharan, S. (2021). A review on word embedding techniques for text classification. ResearchGate. <https://www.researchgate.net/publication/348989232>
2. Volkovich, Z., & Avros, R. (2025). Comprehension of the Shakespeare authorship question through deep impostors approach. Digital Scholarship in the Humanities, 40(1), 308–328.

https://doi.org/10.1093/llc/fqaf009