Capstone Project Phase A

Analysis of Russian Texts Using Deep Impostor Method

25-2-R-12

By:
Daniel Feldman 208134700
Arthur Cherniy 206466500

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

Link to GIT folder.

Table of Contents

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 Convolutional Neural Network (CNN) and a pre-trained BERT transformer are 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. The first cluster consists of 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including the possibility that parts of the manuscript may have originated from another source, potentially lost or unpublished writings from the turbulent period of the Russian Civil War.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 Survey

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 Ostrowski, D. (2020), and related studies.

Ostrowski, D. (2020) 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 is a technique in natural language processing (NLP) that represents words as dense, continuous-valued vectors in a high-dimensional space, such that semantically similar words are located closer together (Suthaharan, 2021). These embeddings are learned from large text corpora using machine learning models, often neural networks, that analyze the context in which words appear, enabling the capture of both semantic and syntactic relationships among words.

One of the most widely used approaches for generating word embeddings is Word2Vec, which employs shallow neural networks to learn word representations by predicting a word given its surrounding context (Continuous Bag of Words, CBOW) or by predicting the context given a target word (Skip-Gram) (Suthaharan, 2021). Through this training, the model adjusts the vector representations to minimize prediction errors, resulting in a vector space where linguistic similarity and analogy relationships are encoded geometrically.

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.

By leveraging the patterns of word co-occurrence in large datasets, word embeddings provide a powerful means of capturing the nuanced meanings and relationships of words, thus supporting more effective language understanding in computational systems (Suthaharan, 2021).

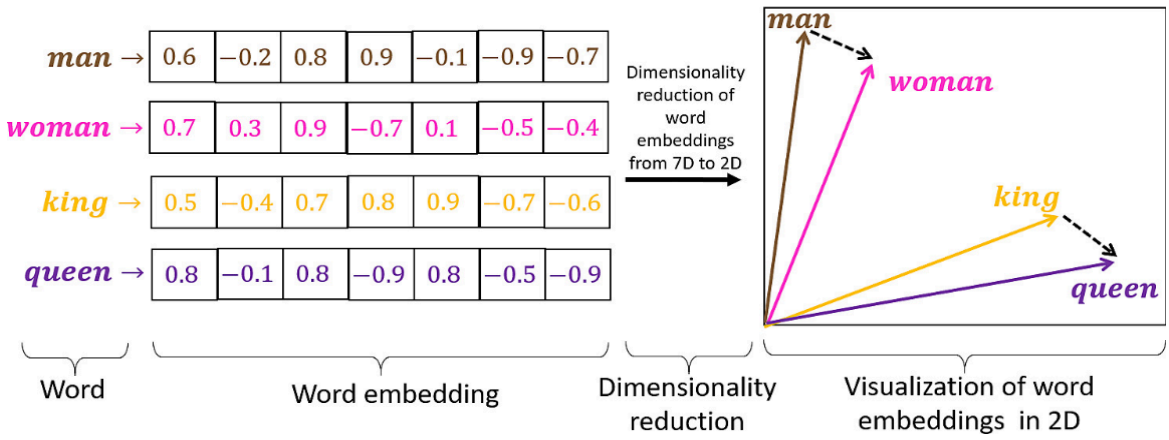


Figure 1. Visualization of word embedding in 2d space. Created by Palaniyappan T. (Medium).

2.3. Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a specialized type of artificial neural network designed to process data with a grid-like topology, such as images. CNNs are composed of several key layers: convolutional layers, pooling layers, and fully connected layers (O'Shea & Nash, 2015).

The convolutional layer is the fundamental building block of a CNN. It applies a set of learnable filters (kernels) that convolve across the input data, producing feature maps that capture local patterns such as edges or textures. Each filter is spatially small but extends through the full depth of the input, and as it slides across the input, it detects specific features at various spatial locations (O'Shea & Nash, 2015).

Pooling layers perform a downsampling operation along the spatial dimensions of the feature maps, reducing the dimensionality and the number of parameters in the network. This is typically achieved by applying an aggregation function, such as max pooling, over local regions of the input. Pooling helps to make the representations approximately invariant to small translations of the input (O'Shea & Nash, 2015).

After several convolutional and pooling layers, the high-level features extracted are fed into one or more fully connected layers. In these layers, each neuron is connected to every neuron in the previous layer. The final fully connected layer typically uses a softmax activation function to produce class probabilities for classification tasks (O'Shea & Nash, 2015).

By stacking these layers, CNNs can learn hierarchical feature representations, from simple edges in early layers to complex patterns in deeper layers, enabling effective analysis of visual and other grid-structured data (O'Shea & Nash, 2015).

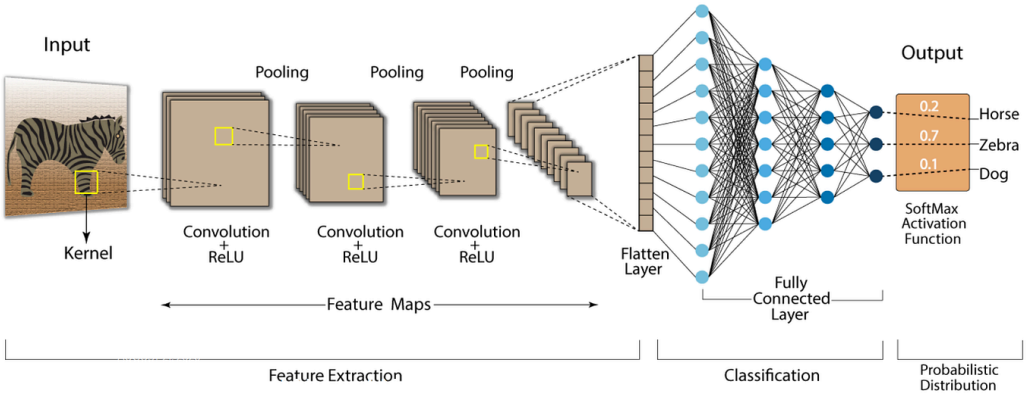


Figure 2. Illustration of a CNN. Created by Waqas, M. T. (Medium)

2.4. Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) specifically designed to overcome the vanishing and exploding gradient problems that can occur when learning long-term dependencies in sequence data (Hochreiter & Schmidhuber, 1997). LSTMs introduce a memory cell and three gating mechanisms: input, forget, and output gates, that regulate the flow of information into, out of, and within the cell. This architecture enables the network to maintain and update information over extended sequences, making LSTMs particularly effective for tasks that require memory of past inputs.

The input gate controls which information is added to the memory cell, the forget gate determines which information is removed, and the output gate governs what information is output from the cell. By selectively retaining or discarding information, LSTMs can learn long-term dependencies in data.

LSTMs have demonstrated superior performance over standard RNNs in a variety of sequence learning tasks, including language modeling, speech recognition, handwriting recognition, time series prediction, and protein secondary structure prediction (Hochreiter & Schmidhuber, 1997).

2.5. Bidirectional Encoder Representations (BERT)

BERT (Bidirectional Encoder Representations from Transformers) is a neural language model introduced by Devlin et al. (2018) that has significantly advanced the field of natural language processing (NLP). Unlike traditional language models, which process text either left-to-right or right-to-left, BERT is fundamentally bidirectional, enabling it to consider the full context of a word by looking at the words both before and after it in a sentence (Devlin et al., 2018). This bidirectionality is achieved through the Transformer encoder architecture, which utilizes self-attention mechanisms to model relationships between all tokens in the input sequence (Devlin et al., 2018).

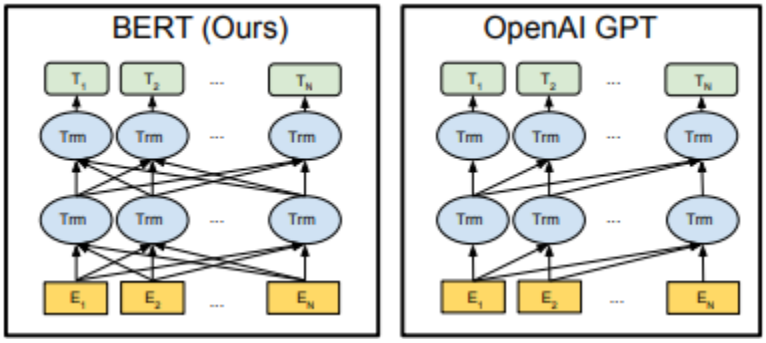


Figure 3. Illustration of BERT vs GPT architecture (bidirectional vs left-to-right). Devlin, J., et al. (2018).

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

Several BERT-based models have been developed specifically for the Russian language, designed to capture its unique linguistic features and trained on large-scale Russian corpora (Zmitrovich et al., 2023). Among these, ruBERT and ruRoBERTa stand out as prominent transformer-based architectures.

ruBERT

ruBERT is a family of BERT-based transformer models tailored for the Russian language. Initially introduced by DeepPavlov and later extended by Zmitrovich et al. (2023). ruBERT models include both ruBERT-base and the newly introduced ruBERT-large. These models are pretrained on Russian Wikipedia and news corpora using a masked language modeling (MLM) and next sentence prediction (NSP) objective. The newer versions leverage larger batch sizes and enhanced computational resources, significantly improving over the original DeepPavlov models in terms of performance on Russian language understanding benchmarks such as Russian SuperGLUE and RuCoLA.

ruRoBERTa

ruRoBERTa is a monolingual Russian adaptation of the RoBERTa-large architecture, trained exclusively with the MLM objective on an extensive 250GB corpus comprising Russian Wikipedia, news, and books (Zmitrovich et al., 2023). The model utilizes byte-level Byte Pair Encoding (BPE) tokenization with a vocabulary size of 50,000 tokens. Trained on approximately 2 trillion tokens using 64 GPUs, ruRoBERTa-large consistently outperforms many multilingual and monolingual models across a variety of Russian NLP tasks, establishing itself as one of the most effective encoder-based models for the Russian language.

2.6. BERT Fine Tuning

Fine-tuning BERT involves taking a pre-trained BERT model and further training it on a smaller, task-specific dataset to adapt it for a particular application (Devlin et al., 2018). While BERT is initially pre-trained on large, general-domain corpora using self-supervised objectives, fine-tuning allows the model to specialize in tasks such as sentiment analysis, question answering, or domain-specific language processing. During this process, the model's weights are updated, typically using a small learning rate (e.g., 2e-5 to 5e-5) to optimize performance for the target task while retaining the general language understanding acquired during pre-training. Fine-tuning is crucial because it enables BERT to learn the specific vocabulary and objectives of the downstream task, resulting in significantly improved accuracy and effectiveness for specialized applications (Devlin et al., 2018).

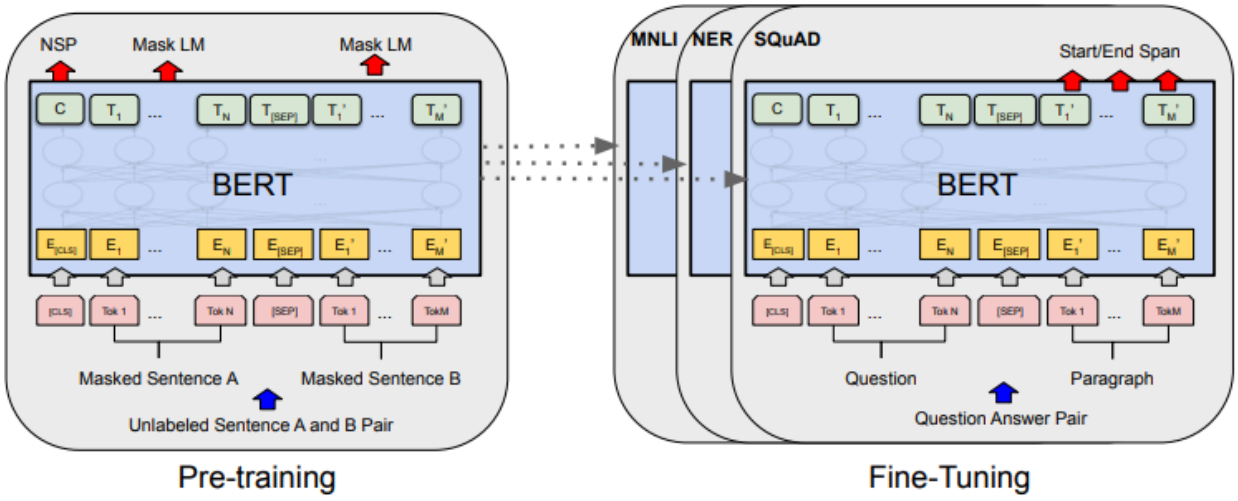


Figure 4. Illustration of BERT stages, pre-training and fine-tuning. Devlin, J., et al. (2018).

Fine-Tuning BERT for Authorship Attribution via Linguistic Style

This study fine-tunes a pre-trained BERT model to perform authorship attribution by capturing individual linguistic styles through deep contextual embeddings. BERT's embeddings, learned via large-scale language modeling objectives, provide a robust foundation for identifying subtle stylistic patterns, rendering the model particularly suitable for tasks involving nuanced language variation, such as distinguishing between different authors (Sun et al., 2019).

The adaptation of BERT for the classification task draws on methodologies employed in sentiment analysis, where a task-specific output layer is appended to the pre-trained model and the entire architecture is fine-tuned on labeled data (Sun et al., 2019). For instance, Volkovich and Avros (2025) fine-tuned BERT for sentiment classification by introducing a classification layer and training on datasets labeled with positive and

negative sentiment, enabling the model to differentiate between contrasting textual characteristics.

In the present context, authorship attribution is reframed as a binary classification problem. Text batches from one group of authors are labeled as “positive,” while those from another group 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 objective is for the fine-tuned BERT model to internalize these stylistic distinctions and generalize them to previously unseen text samples within the target corpus (Sun et al., 2019).

2.7. DTW Distance

Dynamic Time Warping (DTW) is a widely used technique for measuring the similarity between two time series that may differ in speed or length by identifying an optimal alignment that minimizes the cumulative distance between their elements (Senin, 2008). The DTW algorithm proceeds as follows:

A distance matrix C is computed, between the two sequences. Each matrix element represents the distance (typically Euclidean) between corresponding points in the two sequences.

$C(i,j) = distance(a_i,b_j) = |a_i - b_j|$

Next, a cost matrix D 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.

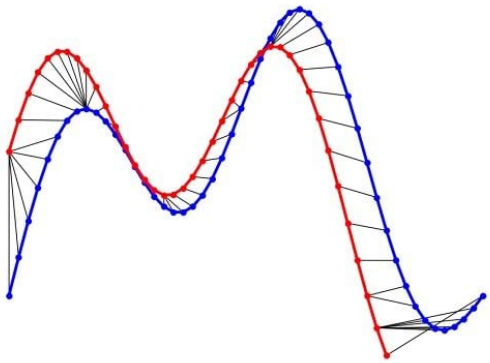


Figure 3. Illustration of DTW distance between two time series. Alizadeh E. (Ealizadeh).

2.8. Isolation Forest Algorithm

Isolation Forest is an unsupervised anomaly detection algorithm that identifies outliers by recursively partitioning data through the construction of multiple isolation trees (Liu, Ting, & Zhou, 2008). The algorithm operates by repeatedly selecting a random feature and a random split value within the feature's range, thereby dividing the dataset into progressively smaller subsets. This process continues until each data point is isolated or a predefined depth limit is reached. Anomalous points, being few and different from the majority, are typically isolated with fewer splits and thus exhibit shorter average path lengths across the ensemble of trees. In contrast, normal points, which reside in denser regions of the data space, require more partitions to achieve isolation. This approach, which does not depend on labeled data, enables Isolation Forest to efficiently detect anomalies in high-dimensional datasets (Liu et al., 2008).

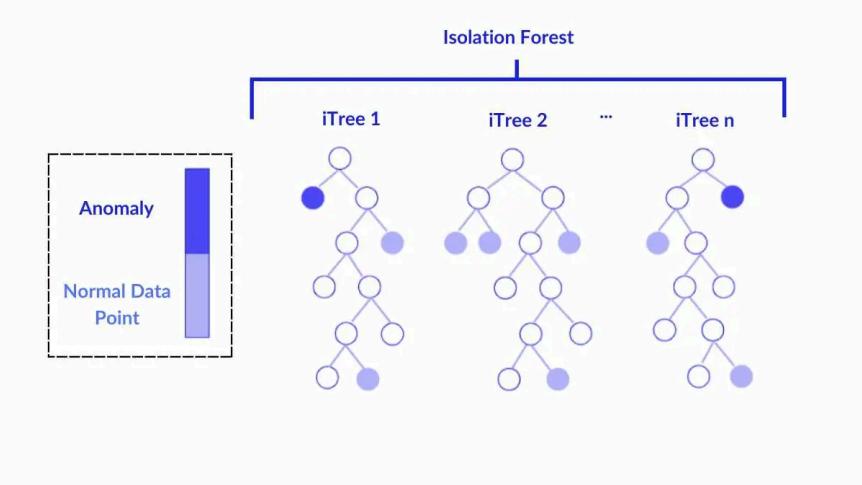


Figure 4. Illustration of Isolation Forest Algorithm. Created by Wrench. (Medium).

3. Project Flow

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
    a. The system shall process and classify a corpus of 20 texts within 1.5 hours.
2. Accuracy and Reliability
    a. The system shall produce consistent results when run on the same dataset under identical conditions.
    b. The system shall achieve at least 90% classification accuracy in distinguishing between impostor text segments.
3. Usability
    a. The system shall execute the entire process with the press of a single button.
    b. The system shall provide a summary of classification results in the format of visualizations (e.g., cluster plots, DTW graphs) to support human interpretation.
4. Security
    a. The system shall restrict codebase access to authorized users only.
    b. All data handled by the system shall be stored in a secure environment (Google Drive).
5. Maintainability
    a. The system shall be modular, allowing easy updates or replacement of the embedding model (e.g., swapping BERT with Word2Vec).
    b. The system codebase shall include documentation and unit tests for all major components.
6. Portability
    a. The system shall be deployable on Linux, macOS, and Windows operating systems.
7. Testability
    a. At least 85% 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.

3.3. Description of Research

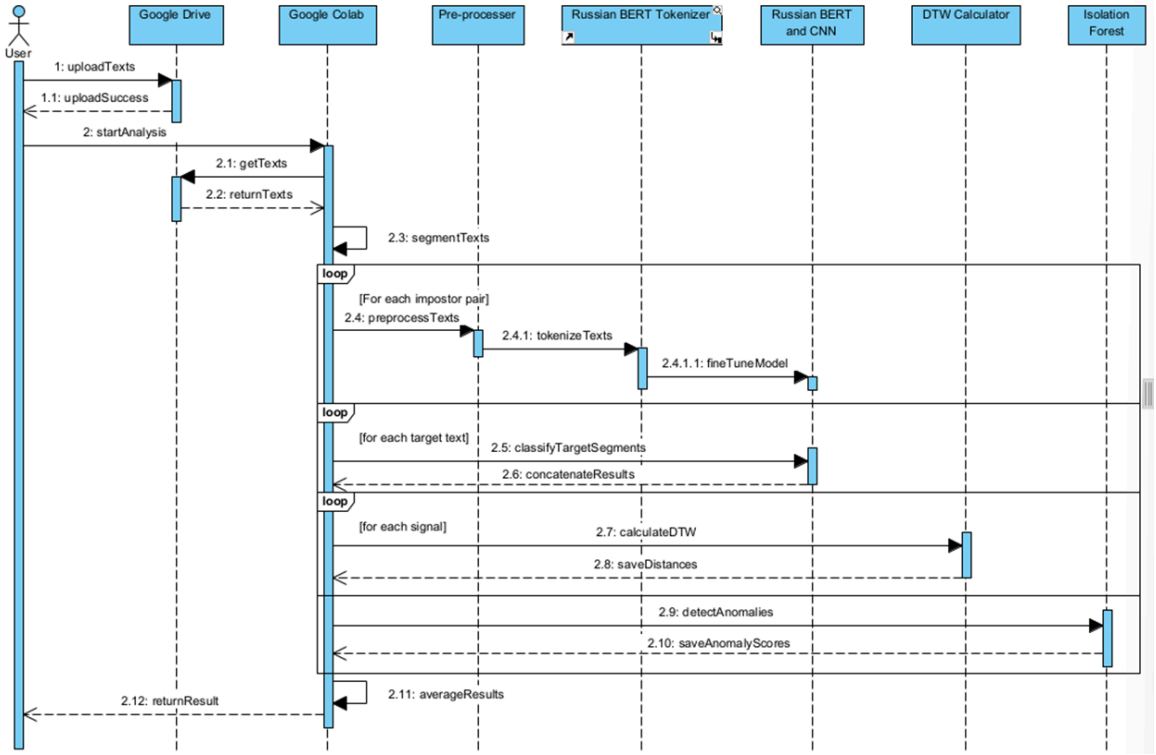
The analysis centers on texts attributed to M. Sholokhov, referred to as the target texts throughout the study. These texts are examined by comparing their linguistic features to those of other writers, referred to as impostors, in order to isolate stylistic patterns unique to Sholokhov.

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.

Pseudo-Code:

1. Divide the texts into segments of L words (batches).
2. Loop on impostors pairs:
    a. Preprocess texts.
        i. Lowercase text
        ii. Remove special characters to keep only Russian letters.
        iii. Tokenize texts using Russian BERT tokenizer.
    b. Train the model to distinguish between the two current impostor texts.
    c. 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).
    d. Every k batches are concatenated into a chunk with averaging the batch scores.
    e. The chunk scores are concatenated to create a signal.
    f. For each target creation the Dynamic Time Warping distance is calculated from the other target texts.
    g. Anomaly score is detected by Isolation Forest Alg. for each one of the texts.
    h. 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 Sequence Diagram



3.3.2. 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. (Created with draw.io)

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.

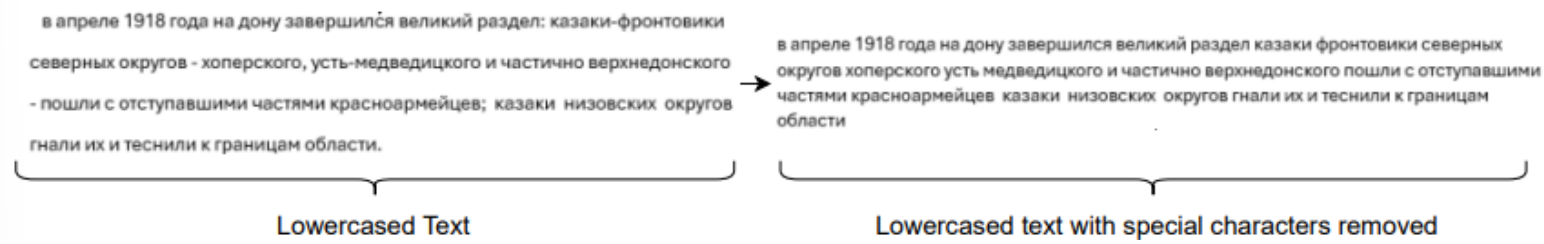


Figure 6. Illustration of removal of special characters. (Created with draw.io)

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.

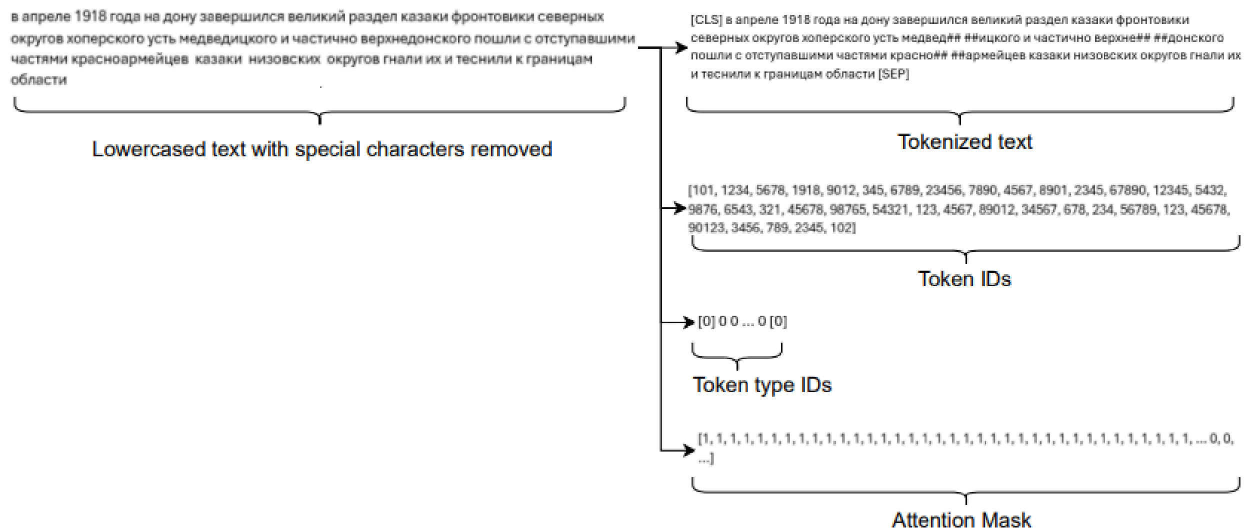


Figure 7. Illustration of Russian BERT Tokenizer outputs. (Created with draw.io)

3.3.3. 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 CNN. 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 CNN. 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.

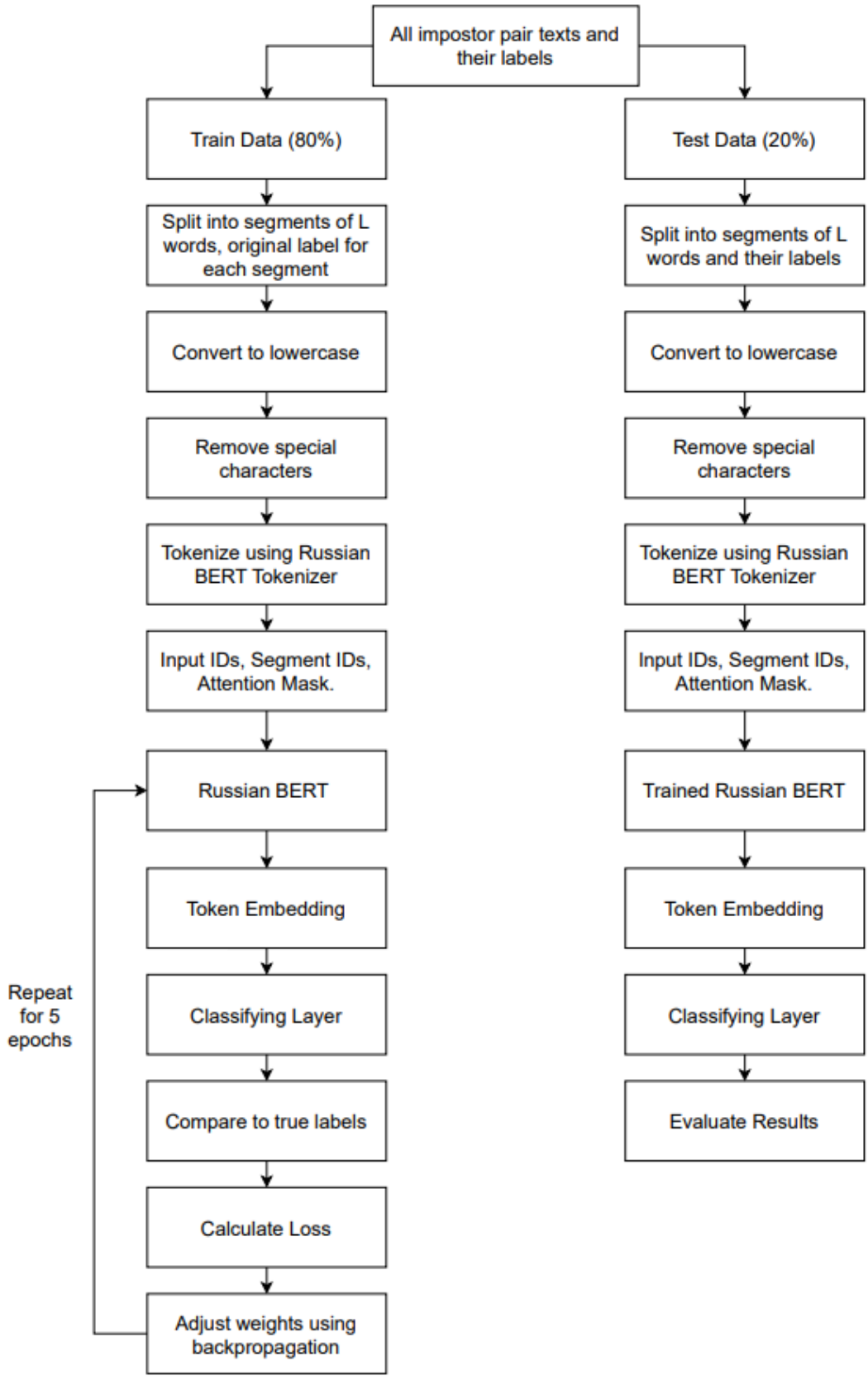


Figure 8. Flowchart of BERT Fine Tuning Process. (Created with draw.io)

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.

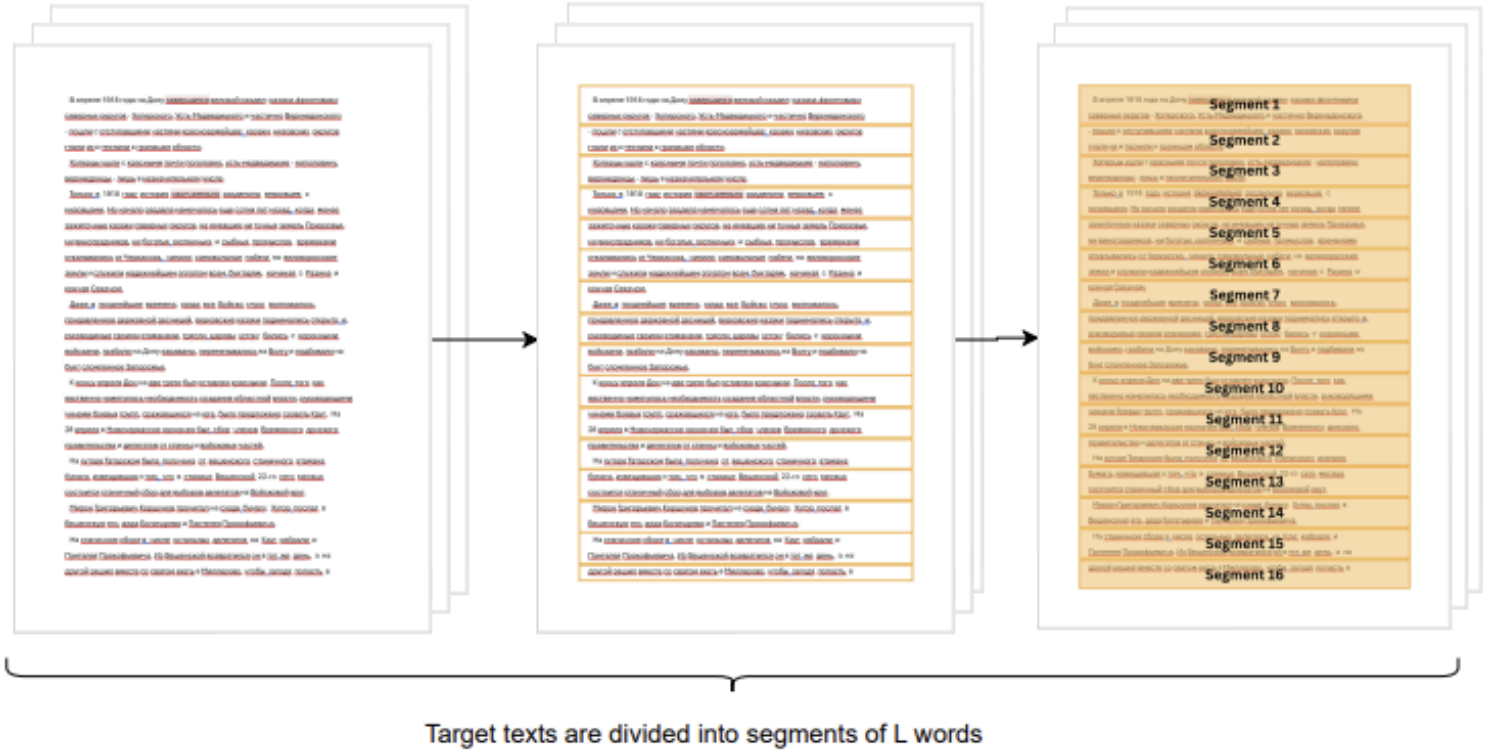


Figure 7. Illustration of text division into segments. (Created with draw.io)

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.

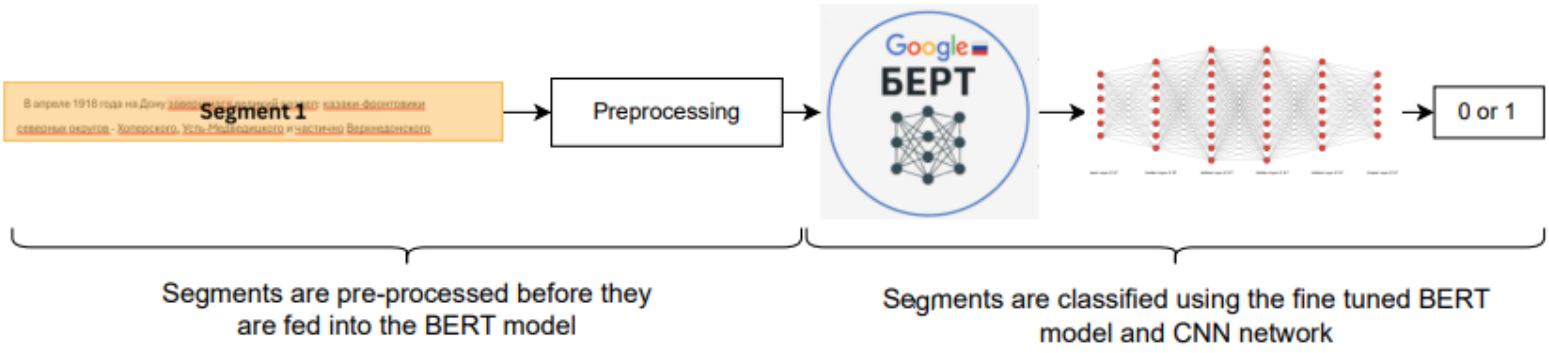


Figure 8. Illustration of segment to classification process. (Created with draw.io)

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.

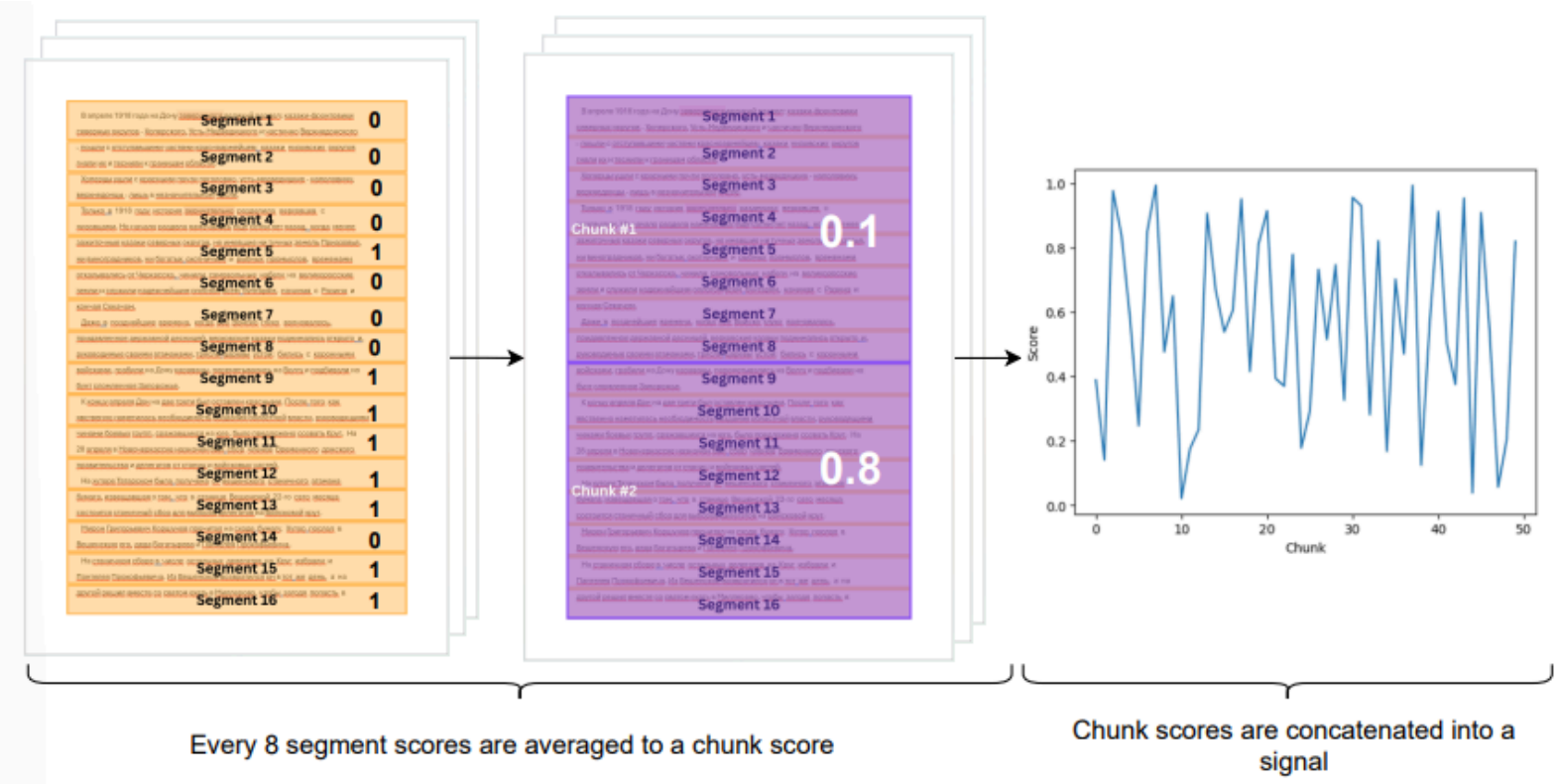


Figure 9. Illustration of segment score averaging to chunks and concatenation of chunk scores to a signal. (Created with draw.io)

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.

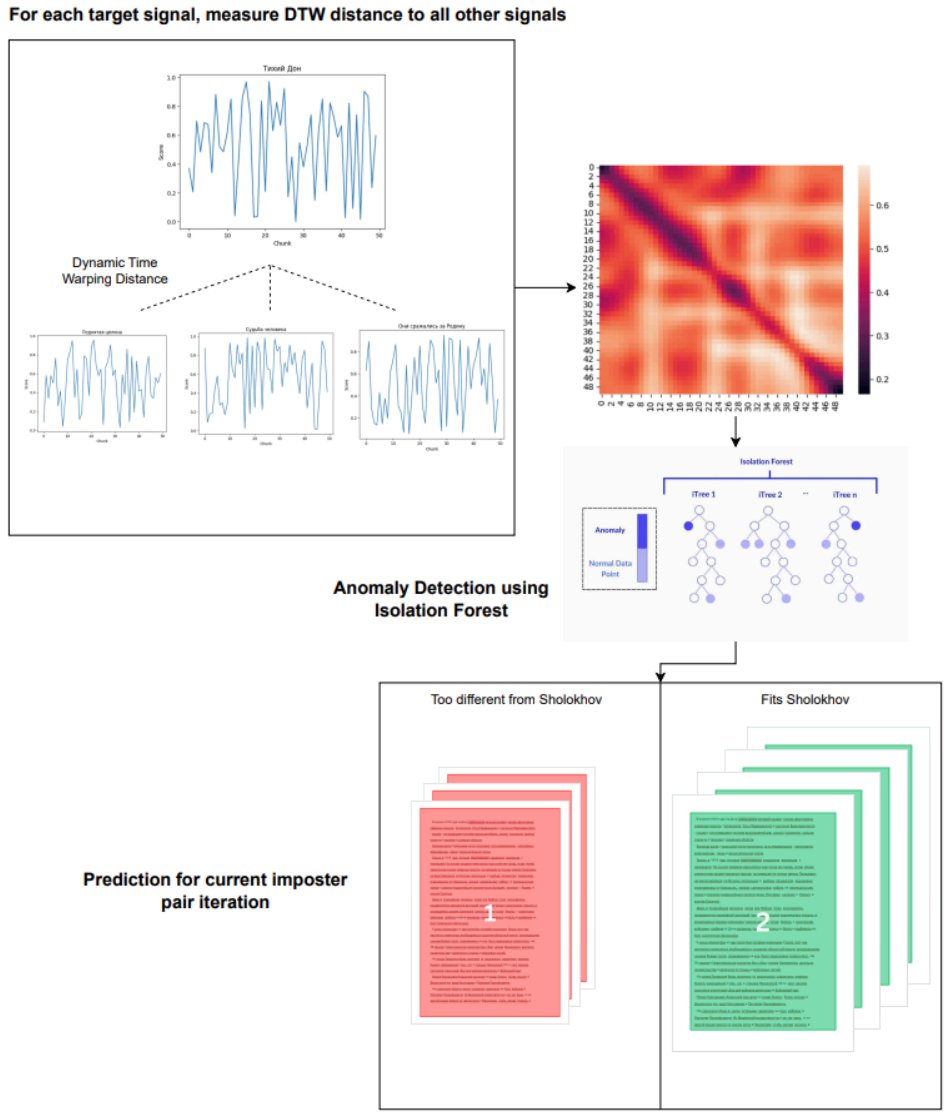


Figure 10. Illustration of the target text classification process in each impostor pair iteration. (Created with draw.io)

3.3.6. Results

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.

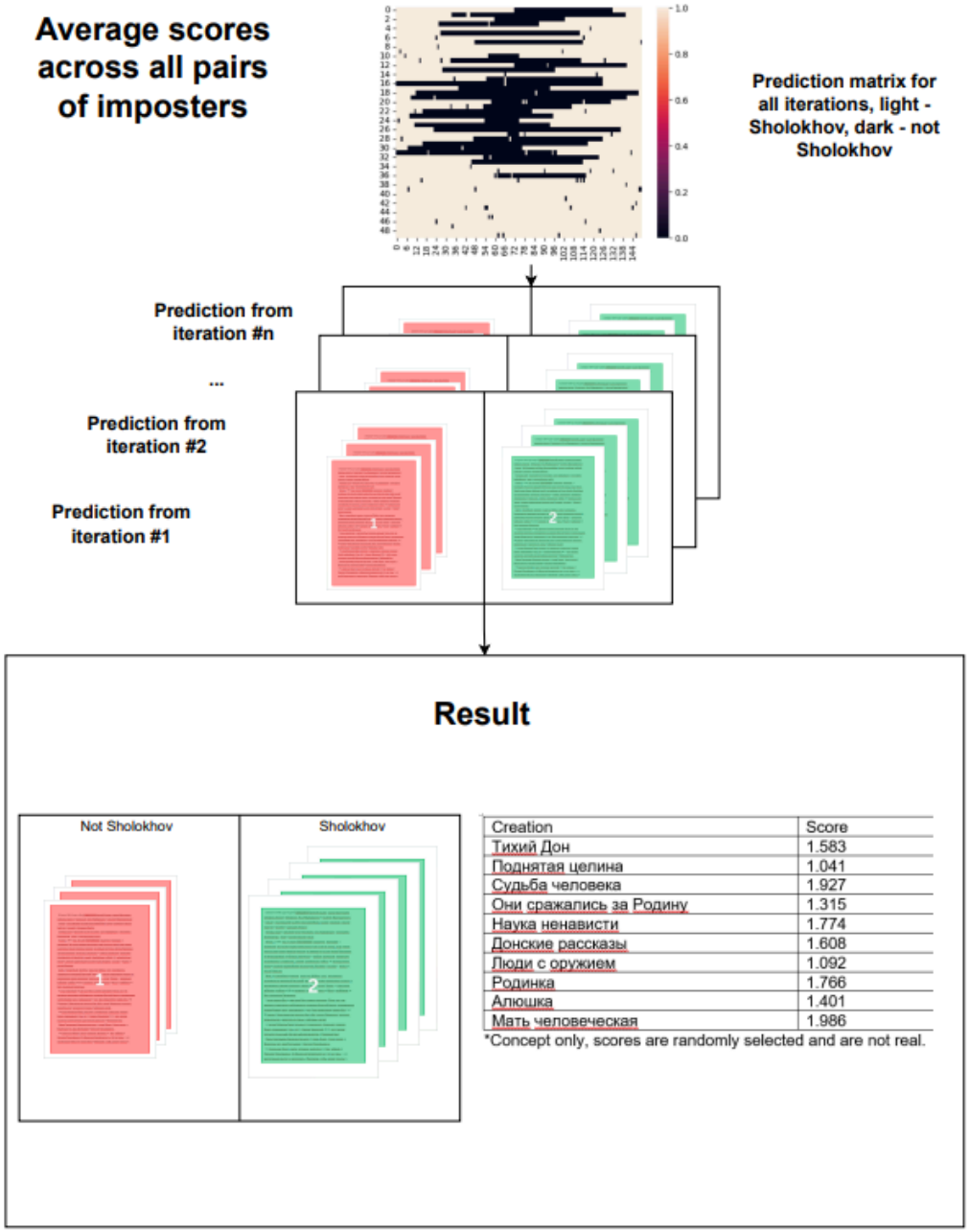


Figure 11. Illustration of results–extraction for each text after all iterations. (Created with draw.io)

3.4. Expected Challenges

The challenges we expect to face are:

- Locating the entire corpus of M.Sholokhov - Older texts may not have been well digitized. Available digital versions might be incomplete or contain errors from poor scanning or locked behind paywalls and institutional access.
- Converting all different work files into usable text - Even with digital versions, some works might be handwritten, which will require more specialized tools to convert the work into usable format.
- Integration of existing Russian BERT tokenizer and model into the project - With the use of old Russian language, many terms might not be well represented in BERT's vocabulary, meaning there would be no embedding to those words.

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 90% 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 85% code coverage.

3.8. Success Indicators

The indicators chosen for success are:

1. Above 90% 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

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

#	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.

Classification Module:

#	Test Description	Expected Outcome
1.	Enter preprocessed Russian text into the Classification module.	Module returns a label, 0 or 1.
2.	Enter unprocessed Russian text into the Classification module.	Module throws an exception asking only to enter preprocessed Russian text.
3.	Enter an empty string into the Classification module.	Module throws an exception asking only to not enter empty strings.
4.	Enter an integer into the classification module.	Module throws an exception asking only to enter preprocessed Russian text.
5.	Enter preprocessed Hebrew text into the Classification module.	Module throws an exception asking only to enter preprocessed Russian text.
6.	Enter 2 Impostor different impostor names into the classification module.	Module trains the model to distinguish between the two impostors and returns a positive message.
7.	Enter the same impostor name into the classification module	Module throws an exception asking to use different impostors.
8.	Enter an impostor name whose texts are not in the system into the classification module.	Module throws an exception saying it could not get the impostor texts.

DTW Distance Calculation Module:

#	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 into the DTW Distance	Module throws an exception mentioning that an entered signal has an invalid score.

	Calculation Module.	
4.	Enter 2 strings into the DTW Distance Calculation Module	Module throws an exception asking to only enter valid signals.

Anomaly Detection Module:

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

Result Calculation Module:

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

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

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

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. 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://www.lamda.nju.edu.cn/publication/icdm08b.pdf

4. O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv. https://arxiv.org/abs/1511.08458

5. Ostrowski, D. (2020). Who wrote that? Authorship controversies from Moses to Sholokhov. Cornell University Press.

6. Senin, P. (2008). *Dynamic time warping algorithm review*. University of Hawaii at Manoa, Department of Information and Computer Science. https://csdl.ics.hawaii.edu/techreports/2008/08-04/08-04.pdf

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

8. Suthaharan, S. (2021). A review on word embedding techniques for text classification. ResearchGate. https://www.researchgate.net/publication/348989232

9. 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

10. Zmitrovich, D., Abramov, A., Kalmykov, A., Tikhonova, M., Taktasheva, E., Astafurov, D., Baushenko, M., Snegirev, A., Kadulin, V., Markov, S., Shavrina, T., Mikhailov, V., & Fenogenova, A. (2023). A family of pretrained transformer language models for Russian. arXiv. https://arxiv.org/abs/2309.10931