

**Capstone Project Phase B**

**Pre-Trained Authorship Representation Transformer**

**23-1- R-18**

Meitar El-Ezra 318608742 [meitar56789@gmail.com](mailto:meitar56789@gmail.com)

Tuval Zitelbach 316219633 [ballnumber18@gmail.com](mailto:ballnumber18@gmail.com)

**Supervisors**:

Pr. Zeev Volkovich ( [vlvolkov@braude.ac.il](mailto:vlvolkov@braude.ac.il) )

Dr. Renata Avros ( [Ravros@braude.ac.il](mailto:Ravros@braude.ac.il) )

***Abstract***

  Authorship attribution is the computational task of identifying the author of a text based on a set of possible candidates. Authors imprint intentional and unintentional traces in the form of linguistic features such as punctuation, registry and semantics which can be used to profile authorship. Previous works attempted to encode the semantic features. Those approaches led to poor results on open-set authors (authors that were not included in the training phase). The presented approach, nonetheless, attempts to encode features from both context and semantics, focusing more on style rather than semantics alone creating **authorship embeddings**. The proposed architecture named PART consists of a contrastively Pre-Trained Authorship Transformer. It uses zero shot generalization capabilities in authorship attribution to compute authorship embeddings with the assistance of the state-of-the-art contextual based models. Analyzing those attributions can contribute to several fields such as forensics, plagiarism detection, social networks analysis, identifying fake news or profiles and more. For the final part of this research, application of test methods in the plagiarism detection problem, will be tested specially for anonymous texts suspected to have been written by William Shakespeare or originated in the Old Testament.

Code Repository:[*https://github.com/TuvalZit/Capstone-Project-23-1-R-18.git*](https://github.com/TuvalZit/Capstone-Project-23-1-R-18.git)

***Table of Contents***

***1. Introduction*** *4-5*

*1.1 Paper's Structure 5*

***2. Background and Related Work*** *5*

*2.1 Authorship Attribution 5-7*

*2.2 Transformer 7*

*2.2.1 Positional Embedding 7-8*

*2.2.2 Encoder-Decoder Stacks 8*

*2.2.3 Scaled Dot-Product Attention 8-9*

*2.2.4 Multi-Head Attention 10*

*2.3 Encoder Transformer 10-11*

*2.4 Transfer Learning 11-12*

*2.5 LSTM 12-14*

*2.5.1 BiLSTM 14*

*2.6 Authorship Embedding 14-15*

*2.7 Contrastive Learning 15-16*

*2.8 Plagiarism Detection 16*

***3. Expected Achievements*** *17*

*3.1 Objectives 17*

*3.2 Outcomes 17*

*3.3 Success Criteria 17-18*

***4. PART Architecture – The Product*** *19*

*4.1 Pre-Processing 20*

*4.2 Frozen Transformer 20*

*4.3 BiLSTM 20*

***5 Project Review and Process Description*** *21*

*5.1 Research Process 21-23*

*5.1.1 Use Case Diagram* [*23*](#_4.2.1_Pre-Processing)

*5.1.2 Evaluation/Verification Plan 23-24*

*5.2 Results and Conclusions 24-26*

*5.2.1 Future Research 26*

***6 User Documentaion*** *26*

*6.1 General Description 26-29*

*6.2 Maintenance Guide 29*

*6.2.1 Package Diagram 29*

*6.2.2 Deployment Diagram 30*

*6.3 Database Structures 30*

***7 References*** *30-32*

# 1. Introduction

Authorship is the source of a piece of writing, music, code, art, etc. Authorship attribution is the task of identifying the author of a text. This task is part of the stylometry field - the study of style, which was originally applied to handwritten text, approximately in the late 19 century. However, nowadays, most textual data is digital, thus recent works applied stylometry to digital texts and computer codes.

Typically, the input for authorship attribution models is a set of candidate authors, set of text samples of known authorship covering all the candidate authors (training set) and a set of text samples of unknown authorship (test set). Feature measurement is a key stage in the production of authorship attribution model. Each authorship attribution task analyzes different features or combinations of features to complete it. These features are lexical, character, syntactic, semantic, bag of words, etc.

The main obstacle in authorship attribution is the extraction of relevant features characterizing the author's style. Previous works and studies in the authorship attribution field addressed this task with supervised approaches, on in-domain authors, that used hand-crafted features or classification task to train their authorship models. Those approaches relied on a large amount of feature engineering in order to reflect both the content and the style of the author. Recent works in this field, found those techniques performed less reliably in terms of accuracy and computation time when applied to real-word application on out-of-domain authors like in forensic investigation.

This project's underlying assumption is that authors imprint intentional and unintentional traces in the form of linguistic features such as punctuation, registry and semantics which can be used to profile authorship. Whether those features are intentional or not, they can be detected and analyzed.

In this paper a better approach to the task of authorship attribution is suggested - learning stylometric representation from both context and semantics, focusing more on style rather than semantics alone. The proposed authorship attribution model is **PART –** a semi-supervised contrastively trained model fit to learn **Authorship Embedding**. It uses zero shot generalization capabilities in authorship attribution to compute authorship embeddings with the assistance of the state-of-the-art contextual based models. In other words, a pre-trained Transformer with a Bi-LSTM head is trained with the contrastive training methods.

In general, authorship attribution has great potential to be used in a wide range of applications: in history and literary science for determining the author of a disputed or anonymous document, in forensic investigation for identifying authors in anonymous or phishing emails, in law to serve as an evidence, in online social network analysis for bot, spam or fake news detection or even stopping disinformation spreading , in plagiarism detection for detecting collaboration in the documents or to identify a document was stolen. The model will be used for the task of plagiarism detection as a part of the testing phase of this project, specifically with works written by William Shakespeare.

## 1.1 Paper's Structure

A short introduction to authorship attribution, previous works in the field, the proposed approach and the range of application are given in Chapter 1. Background and related work will be introduced in Chapter 2. Chapter 3 will cover the expected achievements of the project. The research process will be reported in Chapter 4. Chapter 5 will delve into the proposed PART architecture and all of its components. The GUI and a relevant UML diagram will be presented in Chapter 6.

# 2. Background and Related Work

Firstly, the three forms of authorship attribution are introduced. Additionally, this chapter includes explanation about PART's key components – the Transformer and BiLSTM head. After explaining the Transformer architecture, the focus shifts towards the state-of-the-art Transformer models as an example of previous or current works on the task of authorship attributing. The use of pre-trained or fine-tuned Transformer illustrates the process of transfer learning. The distinction between LSTM and BiLSTM is added to this chapter. The authorship embedding and the contrastive learning method is mentioned in also.

## 2.1 Authorship Attribution

Authorship profiling is the analysis of a given set of texts in an attempt to uncover various characteristics of the author based on stylistic - and content-based features, or to give identification to the author. Authorship attribution, on the other hand, refers to those identifiable features of a text and aims to determine the writer of the text according to them.

Authorship attribution is divided into three forms. First, closed-set attribution, where the true author of a given text must be included in the set of suspects. Second, open-set attribution, where the true author of a given text could be excluded from the set of candidates. The third approach, the author verification where there is only one candidate author.

In real life scenarios, the training and test sets of texts may differ in the topic and genre. Those situations are examined in the Cross-Domain Authorship attribution. The key goal is to exclusively focus on the stylistic properties of texts regarding to the personal style of the author, while avoiding using extra information about the topic or genre of the text.

An automated model which performs accurate authorship determination would be a beneficial model for these areas to use. Content and style are the two typical features which are being taken under consideration when trying to perform authorship attribution. Stylometric features, namely word choice and frequency, punctuation or sentence length among others, are something that can easily be located and identified by computers and transformed into numerical features. Later on, machine learning approach can train model and learn from it.

In machine-learning, trained models have a hard time adapting to out-of-train domains and frequently fail to shift to new datasets, a fact highlighted by Murauer et al. in their work in [5]. When solving the task of authorship attribution, it is essential to train the model on multiple datasets, each varying in its contents, whether it being documents’ size, languages, genres or style. Generalization is a concern that often worries developers of machine learning models. Data bias in machine learning is a type of error in which certain elements of a dataset are more heavily weighted and/or represented than others. A biased dataset does not accurately represent a model’s use case, resulting in skewed outcomes, low accuracy levels, and analytical errors, which is why it is another concern in the field of machine learning [6]. These effects can be lessened by implementing augmentation techniques like word removal.

One more problem with models which are dedicated to the attribution task, is that these models have proved to not be secure when it comes to adversarial attacks.  Simko et al. [7] had studied the influence of adversaries on models that perform program authorship attribution on code files. As a result, he concluded that if a fake style is inserted to the model, the attribution may be inaccurate. Even though the models are not meant for natural language, it is still a reason for concern.

These are not all the problems that might occur while trying to attribute authorship. The size of a document might be shorter than normal (i.e., a tweet) which could make the task more difficult, or the orthogonality between authors and the subject they choose to write may be troublesome for models when trying to assign an author to a text with a topic not as documented. Co-authoring, when speaking of larger bodies of texts could also present new problems, when the model struggles to detect as writing style changes, amongst other complex issues [8].

Most of the previous works regarding this task focused on the closed-set Authorship attribution. Nevertheless, the authorship embedding, that is presented in this paper, is calculated with zero-shot generalization capabilities in authorship attribution, namely, open-set authorship attribution form.

In addition, several methods of machine learning were used and compared, from classical to most recent models, all of which fed with features with the desire to attribute authorship of Russian literature pieces. [4]

**Recent related works:**

1. **PolitiBETO:** A BETO model, which isa BERT (Bidirectional Encoder Representations from Transformers) model trained on a big Spanish corpus, trained on a collection of political language and an ensemble approach profile authors in the same domain [9]. Other authors have also used BETO, this time in combination with the model MarIA [10].
2. **Sentence-BERT:** A transformer-based model dedicated for identifying multilingual spreaders of hate speech. [11], [12].
3. **Authorship attribution of social media messages** is another challenge that is treated by applying the task of authorship attribution to Convolutional Neural Networks architecture. [13].
4. Studies focusing on zero-shot classification which use entailment-based models for author profiling and similar tasks, with a small training data set [14]. [15] [16].

After examining the literature, a conclusion can be made that research focused on authorship is bounded by the data used to train the model. Models built with these techniques are successful close-set domains but fail at others. Accordingly, using authored data, zero-shot classification of authors can be performed if, instead of using supervised classification, a semi-supervised classification approach is used.

There are two keys to success in this regard, first is finding a suitable method for robust embedding. Second is a training scheme that allows for finding similarities between the texts of authors.

## 2.2 Transformer

Transformers play a prominent role in the field of NLP, computer vision and image generation. Transformers are the building blocks of state-of-the-art models like GPT-3, BERT, RoBERTa, DALL-E, etc.

There are two important keys that set transformers apart from other sequence-to-sequence models like Recurrent Neural Networks (RNN), Long Short-Term Memory Neural Networks (LSTM) or Convolutional Neural Networks (CNN). First, unlike RNN and LSTM, transformers process an entire input all at once, making them convenient models to train extremely large inputs. Second, transformers cleverly use the concept of **Attention**.

### 2.2.1 Positional Embedding

In NLP tasks, the transformer’s input data is a text that is parsed into tokens. Each token is then converted into a vector using word embedding. Then, because the transformer does not contain neither recurrence nor convolution, positional embedding solves this issue in order for the model to utilize the order of the sequence. Those embeddings hold information about the relatives or absolute position of each token in the sequence. In the paper “Attention is all you need” in which the concept and architecture of transformers was first introduced, the authors used sine and cosine functions of different frequencies to create the positional embedding:

*PE* stands for positional embedding, *pos* for the position and *i* for the dimension.

### 2.2.2 Encoder-Decoder Stacks

The transformer has an encoder-decoder structure. The encoder is built from a stack of N identical layers. Each layer consists of two sub-layers. First, is the multi-head attention and the second is a simple, position wise fully connected feed-forward network. A residual connection is employed around each of the two sub-layers, followed by layer normalization. In other words, the output of each sub-layer is the sum of the input plus the function implemented by the sub-layer.

Similar to the encoder, the decoder is built from a stack of N identical layers However, in addition to the aforementioned sub-layers, the decoder has inserted a third sub-layer between them, which performs multi-head attention over the output of the encoder stacks. Like in the encoder, a residual connection is employed around each of the two sub-layers, followed by layer normalization. The first sub-layer of the decoder, the multi-head attention is modified to be masked. Multi-head attention prevents positions from attending to subsequent positions.

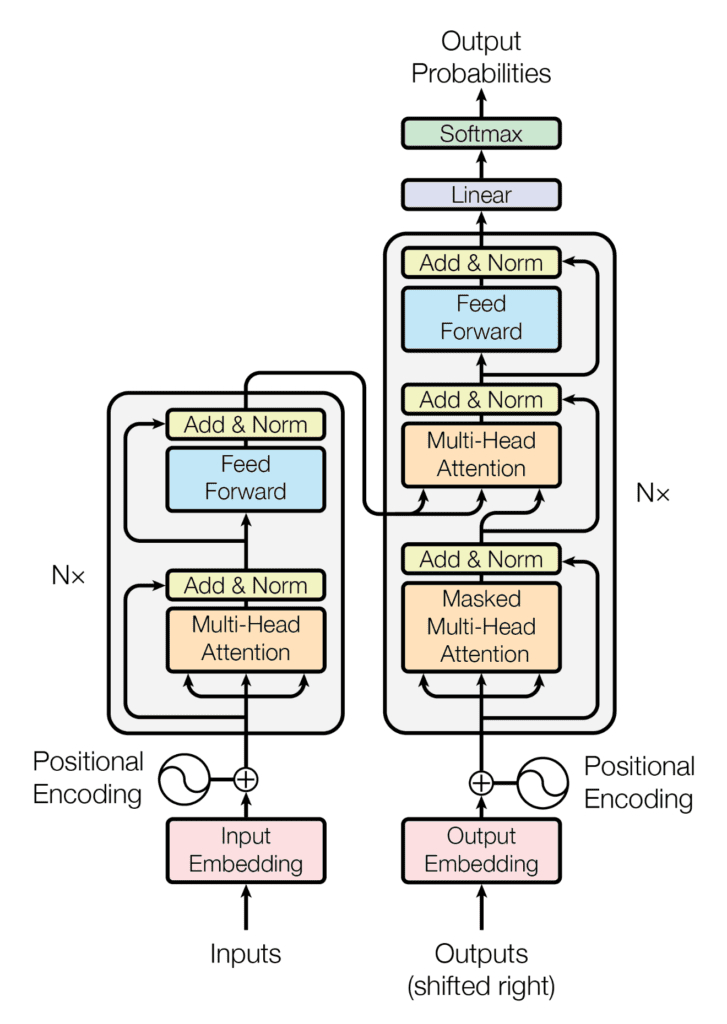


Figure 1: Transformer Encoder-Decoder Architecture of The Transformer (taken from "Attention Is All You Need", page 3[22])

### 2.2.3 Scaled Dot-Product Attention

Attention can be described as a technique that is meant to mimic cognitive attention. This mechanism enhances some parts of the input data while diminishing other parts. In other words, it tries to focus on the important parts of the data. Transformer is the first transduction model relying entirely on **self-attention** to compute representations of its input and output without using sequence aligned RNNs or convolution.

Multi-head attention is a transformer model of attention mechanism. When the attention module repeats its computations over several iterations, each computation forms parallel layers known as attention heads. Each separate head independently passes the input sequence and corresponding output sequence element through a separate head. A final attention score is produced by combining attention scores at each head so that every nuance of the input sequence is taken into consideration.

The Transformer implements a scaled dot-product attention which follows the procedure of the general attention mechanism. First a dot product is computed for each query, *q*, with all of the keys, *k*. It subsequently divides each result by square root of and proceeds to apply a SoftMax function. Thus, it obtains the weights that are used to scale the values, *v*.

This attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

In practice, the prior mentioned computations can be applied to the entire set of queries simultaneously. For that cause, the matrices – *Q*, *K* and *V* serve as inputs to the attention function.

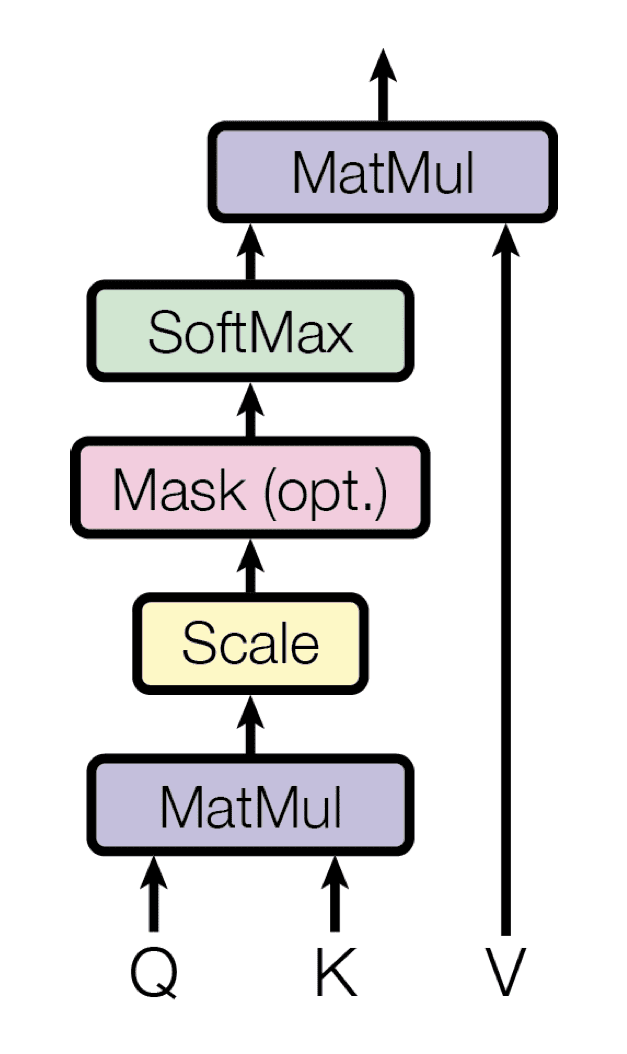


Figure 2: Scaled Dot-Product Attention (taken from "Attention Is All You Need", page 4[22])

### 2.2.4 Multi-Head Attention

Transformer architecture include multi-head attention layers. This attention mechanism linearly projects the queries, keys, and values ℎ times, using a different learned projection each time. The single scaled dot-product attention mechanism is then applied to each of these ℎ projections in parallel to produce ℎ outputs, which, in turn, are concatenated and projected again to produce a final result.

Multi-head attention layer allows the attention function to extract information from different representation subspaces, in contrast to a single scaled dot-product attention head that cannot do it. The following formula explains the computation of the attention:

*Q*, *K* and *V* denoting matrices packing together sets of queries, keys and values, respectively. denoting a projection matrix for the multi-head output. Each  implements a scaled dot-product attention characterized by its own learned projection matrices:

denoting projection matrices that are used in generating different subspace representations of the query, key and value matrices.

## 

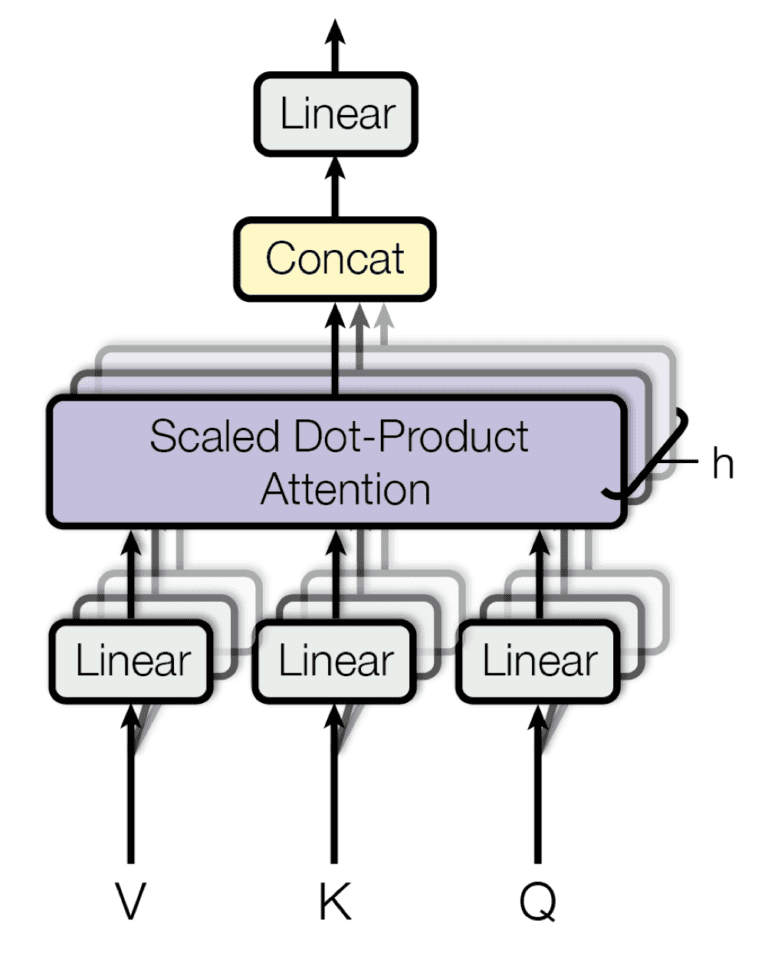


Figure 3: Multi-Head Attention (taken from "Attention Is All You Need", page 4[22])

## 2.3 Encoder Transformer

In recent years, plenty of transformers models and architectures have been trained to understand natural language. Those pretrained models are capable of interpreting semantic features of speech and rarely stylistic understandings. The goal of this project is to output authorship embeddings for given texts, meaning, to use a stylistically oriented transformer, centered on authorship with small differences from pre-trained procedures that focused on semantics.

Previous works that focus on style mainly involved content generation instead of analysis, consequently they include a style vector into the system to be represented as the Styleformer. Transformers that are style-oriented are scarce for NLP, although in the field of audio, some works gave partial focus to style but without concern of authorship. In a similar domain to text, code writing, some interest has been in an authorship transformer, however it did not reach to the text field.

To produce authorship embedding, an encoder transformer is required, such as RoBerta. In addition, with comparison to BERT, ELMo and GPT-2 encoders to their decoder-style equivalents, the encoder-styles models generated better representations and produced more contextualized representation.

The closest transformers for the task of authorship embedding are the BertAA and RoBERTa. BertAA is a fine-tuning of a pre-trained BERT language model with an additional dense layer and a SoftMax activation to perform authorship classification.

The use of embeddings can help take authorship to a more complex level, enabling cross-domain scenarios with pre-trained models and a normalized corpus. RoBERTa can be used to generate embeddings, its representations are stronger than the BERT generated embeddings. A particular scenario is the case for microtext from social media, which can be particularly challenging.

## 2.4 Transfer Learning

Many machine-learning methods performed well if they only took into consideration that the training and test data are drawn from the same feature space and the same distribution. As soon as the distribution changes, most of the models need to be rebuilt from scratch using newly collected training data. In real life applications, recollect new data and rebuilding the model is expensive or even unfeasible. For this reason, knowledge transfer or transfer learning between source and target domains and tasks would be helpful.

Sinno Jialin Pan and Qiang Yang divided transfer learning into 3 different types: Transductive transfer learning, Inductive transfer learning and Unsupervised transfer learning. An Overview of Different Setting of Transfer Learning is presented in Figure 4.

Inthe **transductive transfer learning** settings, the source and target tasks are the same, although the source and target domains (the data distribution on which a model is trained) are different but related. This type can be divided into two subtypes: **Domain adaptation** and **cross-lingual learning**. Domain Adaptation is a technique to improve the performance of a model on a target domain containing insufficient annotated data by using the knowledge learned by the model from another related domain with adequate labeled data. Cross-lingual learning is a paradigm for transferring knowledge from one natural language to another.

In the **inductive transfer learning settings**, the target task is different from the source task, regardless of if the source and target domains are the same or not. This type can be divided into **multi-task transfer** and **sequential transfer.** In multi-task transfer learning, several tasks are learned concurrently, and common knowledge is shared between the tasks. In sequential transfer learning, the source data’s general knowledge is transferred to only one task. Sequential inductive transfer learning largely consists of two steps: pretraining and adaptation. First the model is pretrained with the source task and later the model is adapted to the target task. The following step is adoption. There are two options in this phase: keeping the weights (embedding or feature extraction) of the pretrained model or modifying them to the target task (fine tuning).

 In feature extraction, the weights do not change because those pretrained features are useful for the target task. However, the last layer of the pretrained model is specific to the source task, therefore a new layer needs to be added on top of the pretrained model and needs to be trained from scratch.

Nevertheless, in fine-tuning, the weights are modified on a specific task, meaning unfreeze a few of the top layers of a frozen model base and jointly train both the newly added classifier layers and the last layers of the base model. This allows us to fine-tune the higher-order feature representations in the base model in order to make them more relevant for the specific task. On one hand this is much more flexible and no need for further adjustments. On the other hand, this can lead the model to forget pre-trained knowledge abruptly and drastically upon learning for the new task. This is called catastrophic forgetting.

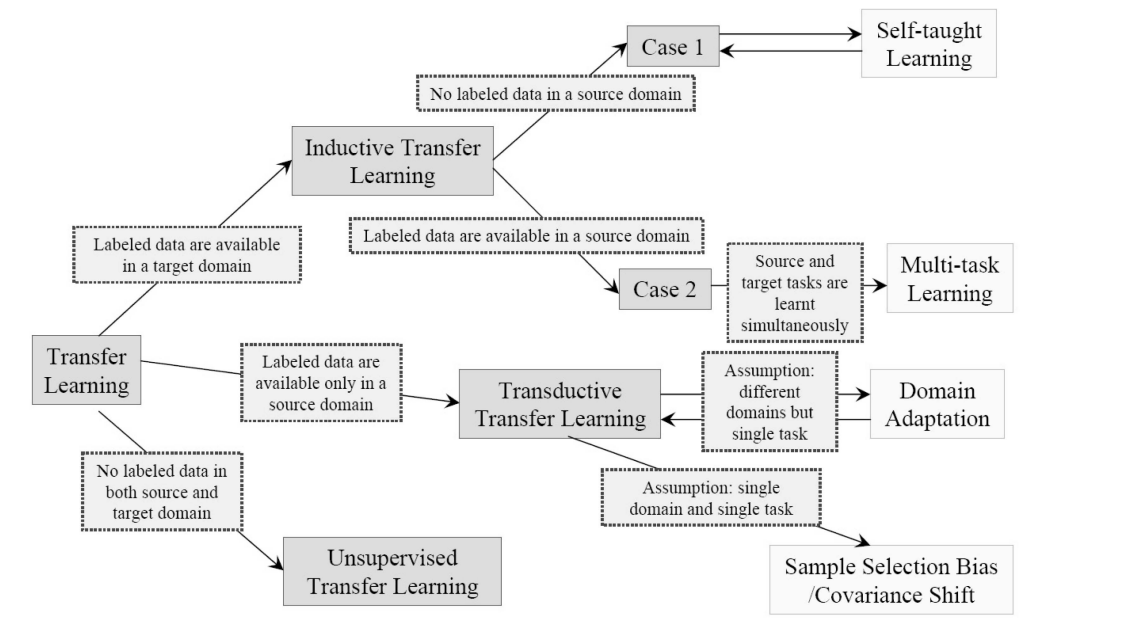


Figure 4: An Overview of Different Setting of Transfer Learning – (Taken From "A Survey on Transfer Learning", page 5[21])

## 2.5 LSTM

In a regular Recurrent Neural Network (RNN), the problems of vanishing and exploding gradients frequently occurs when connecting previous information to new information. The Vanishing Gradient problem occurs when the information about the input or gradient passes through a lot of layers, it will vanish by the time when it reaches to the end or beginning layer. This problem makes it hard for RNNs to capture the long-term dependencies. Beside that, the exploding gradient problem, which rarely happens, refers to the scenarios, similar to the first problem, in which information about the input or gradient passes through a lot of layers, however the result is different – the gradient is very large when it reaches to the end or beginning layer. This problem makes RNNs hard to train

In 1997, Horchreiter and Schmidhuber introduced a special kind of RNN - the Long Short Term Memory networks (LSTMs) as a solution to those problems. The major difference between LSTM and RNN that hidden layer updates in RNN are swapped with memory cells. The LSTM cell structure is presented in Figure 5.

LSTMs were introduced to avoid the long-term dependency problem of vanishing and exploding gradients while remembering information for longer periods of time is their default behavior. LSTM is unidirectional network where the network stores only the forward information. This type of network is better for maintaining long-range connections, recognizing the relationship between values at the beginning and end of a sequence.

The robustness of LSTM is achieved due to its complex structure, which is formally defined by the following equations:

Where and are the LSTM model parameters, and are the biases. The input gate, forget gate and the output gate denoted as , and respectively. The output that is passed to the next layer of the network is . Lastly, is the state that is passed to the next step at time .

As mentioned above, LSTM solves the Vanishing gradient problem. Basically, this happens in the computation of equation 5. The sum prevents the gradient from vanishing and the extent to which it grows is controlled by the model's parameters through the input and forget gates' parameter learning.

Considering stylometry, LSTMs are able to extract temporal information in terms of their internal timeline. Furthermore, the inner properties of the model are adequate to characterize global temporal dependencies but not the temporal nuances associated to relevant stylometric features

### 

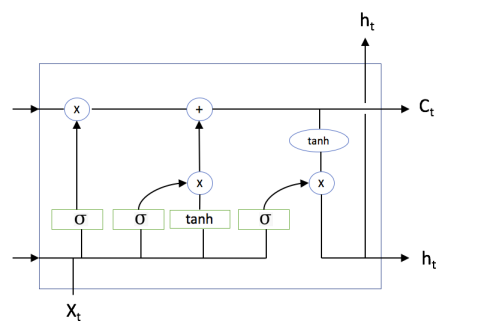


Figure 5: LSTM Cell Structure (taken from "Utilizing Recursive Neural Networks for Contextual Text Analysis, page 13 [23]

### 2.5.1 BiLSTM

Bidirectional long-short term memory (Bi-LSTM) is a recurrent neural network used primarily on NLP problems. In contrast to standard LSTM, in BiLSTM the sequence input flows in both directions backwards (future to past) and forward (past to future). It is done in order to preserve the future and the past information on and utilize it. It is a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence.

In summary, BiLSTM adds one more LSTM layer, which reverses the direction of information flow. Briefly, it means that the input sequence flows backward in the additional LSTM layer. Then there are several ways to combine the outputs from both LSTM layers such as average, sum, multiplication, or concatenation. Every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions.

## 2.6 Authorship Embedding

A reminder to the project's premise is that authors imprint intentional and unintentional traces in the form of linguistic features such as punctuation, registry and semantics which can be used to profile authorship. Whether those features are intentional or not, they can be detected and analyzed. Analyzing abstract or subjective properties such as rhythm, punctuation, flow, registry and so on is a challenging task.

Therefore, the majority of recent works point to the requirement to adopt advanced language models to extract fine-grained details from the text and revel characteristics aligned with the author. Briefly, by comparing text and maximizing the similarity of their embedding when they belong to the same author, in turn minimizing similarity with unaligned text. If the premise is true, the model will be able to extract meaningful and representative features from various of texts that belong to the same author.

Those representation of writing style are called authorship embedding. The semantic embedding generated by transformers can detect contextual and semantic feature from a text, focusing on the content of the text. Whereas authorship embedding, primarily objective, is to encode properties from context and semantics too from the author of the text, shifting the focus of the transformer slightly toward style.

## 2.7 Contrastive Learning

The objective of contrastive learning approaches can be accomplished with the help of contrastive loss function. In general, loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.

In the research process, several contrastive loss function have been studied such as Contrastive Loss, Triplet Loss, NCE Loss and InfoNCE loss.

Contrastive learning is a machine learning technique used to learn low-dimensional representation of a data by contrasting between similar and dissimilar samples. Essentially, when training, the objective is to minimize the distance between the embedding of similar samples and maximize the distance between dissimilar samples using Euclidean Distance. The notion of contrastive learning can be applied in both supervised and unsupervised methods.

In supervised methods, during training, all the samples' labels are available. With this approach, positive and negative pairs or even triplets can be generated with ease by just looking into the labels. Worth to mention, that this process requires a long-drawn out computational time and mass resource. In addition, in every dataset, there are many negative pairs or triplets that already satisfy the contrastive training objective and give a zero-loss. Hence, the model will slowly reach convergence (model converges when extra training will not improve it). To sum up, deep supervised learning methods rely heavily on manual labels and suffers from generalization error, spurious correlations and adversarial attacks.

To solve this issue, the model must generate hard pairs and hard triplets, which means that their loss value is high. Namely, similar pairs that are far apart and dissimilar pairs that are very close.

While supervised learning methods assumes that the entire training dataset is labeled, reality may not always be like it. In such cases, self-supervised learning methos is used. Accordingly, features of the data are extracted and exploited to generate pseudo-labels, which then used in downstream tasks.

Recent works, such as the models T5 and GPT-3 which were pre-trained with Casual Language Modeling achieve state of the art results in out-of-domain benchmarks, which depicts how impressive generalization self-supervised learning methods can obtain with big enough amounts of data, when dealing with zero-shot tasks. These transformer-based models are powerful; however, semi supervised training methods are semantically oriented. Contrastive Learning is a good approach to take for learning representations directly from the dataset, without masking or using pseudo-labels.

Works which have applied semi-supervised contrastive approach to different domains, to learn representations of text and other modalities:

1. **CLIP**: A neural network which efficiently learns visual concepts from natural language supervision. CLIP can be applied to any visual classification benchmark by simply providing the names of the visual categories to be recognized, similar to the “zero-shot” capabilities of GPT-2 and GPT-3. It has achieved exceptional performance in the common benchmarks. [17]
2. **SimCSE**: A simple contrastive learning framework that greatly advances state-of-the-art sentence embeddings. The framework employs contrastive learning to sentence similarity tasks. [18]

In order to successfully attribute authorship, there must be available authorship. Thus, labels can be utilized for learning, too. Contrastive learning can be supervised, as well, and as shown in [19] it can improve the quality of representations. A good illustration for that is natural language inference as described in [20]. It is shown there how supervised contrastive learning produces improved results which can later be labeled.

## 2.8 Plagiarism Detection

The Internet has paved the way for both the spreading of anonymous texts as well as easy ‘‘borrowing’’ of ideas and works of others. Plagiarism is one of the growing global problems experienced by the publishers, researches and educational institutions which is generally defined as literary theft and academic dishonesty in the literature. In other words, plagiarism is the act of taking ideas, documents, codes, images, etc. of another person and presenting them as their own works. and it really has to be prevented and stick to the ethical principles. This proves an act of dishonesty in academics and literature and hence it has to be prevented.

Plagiarism has raised a number of important questions regarding authorship. Can we anonymous author be identified by comparing his text with the works of known authors? Can a text, or parts of it, be plagiarized? Such questions are clearly of both academic and commercial importance.

The plagiarism problem can be divided into two types: **extrinsic analysis** and **intrinsic analysis**. In the extrinsic case, the objective is to detect plagiarism by finding near-matches to a text in a database of texts. In intrinsic detection, we wish to show that different parts of a presumably single-author text could not have been written by the same author. Extrinsic plagiarism analysis is more closely related to algorithmic issues than attribution problems because it involves approximate pattern matching. Intrinsic plagiarism analysis, however, is very tightly tied to the problem of authorship verification since stylistic inconsistencies within a text indicate parts written by different authors.

# 3. Expected Achievements

This chapter consist of the expected achievements of the project attached to the expected outcomes. In addition, a list of success criteria to examine whether the project succeed or not.

## 3.1 Objectives

The primary objective of the project is to advance the field of authorship analysis by representing the PART model, which would obtain style characterization capabilities to include features from the author's unique style. Those capabilities will help generate a numerical vector which would be a solid representation of the style of an author. This numerical vector is called the authorship embedding that is a projection of the author's style into a content-aware hyper space. The expectation from the model is to be able to generate embeddings for words, sentences and documents, as well.

Furthermore, another objective of the project is to test PART model for the plagiarism detection task, especially with works written by William Shakespeare or originated from the Old Testament.

## 3.2 Outcomes

1. A new semi-supervised contrastive learning approach to address zero-shot authorship attribution and profiling, using state-of-the-art contextual-based models.
2. A model that is able to generate numerical stylometric embeddings which can project words, sentences and documents, surmounting the bottleneck of text length.
3. Author embedding computation with zero-shot generalization capacities in authorship identification.
4. Experiments which testify to the concept described in (3), using the architecture from (2).

## 3.3 Success Criteria

The aspiration is creating such a representation that would be aligned to the author and not the content of the text they wrote. It will be done by contrasting the embeddings of different documents written by the same author. A conceptual representation of the approach:

As for the plagiarism detection ability, the expectation from the model is to produce authorship embedding with high similarity value for texts that are suspected to be written by a specific author in comparison with texts that are known to have been written by them.

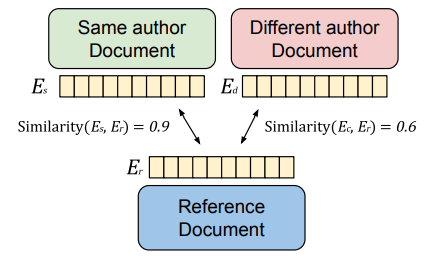


Figure 6: "Example of a comparison of authorship embeddings. Er, Es and Ed represent authorship embeddings from a referenced document, a document whose author is the same as the reference and a text from a completely different author. When compared with a similarity function, the related document similarity should be higher than an unrelated text."(Taken From "PART: Pre-trained Authorship Representation Transformer", page 2[1])

## 4. PART Architecture – The Product

The architecture is based on three important phases. First, the data is divided into chunks, which are later tokenized. The pre-tokenized chunks comprise of L tokens each. Second, the chunks are sent to a frozen transformer. The pre-trained frozen transformer produces semantic word embeddings in the form of a matrix with dimension (L, K) where L is the sequence length and K is the number of features. Lastly, the semantic word embeddings are fed to a Bidirectional LSTM (BiLSTM). The output of the network is an Authorship Embedding vector sized K. The proposed architecture is presented :

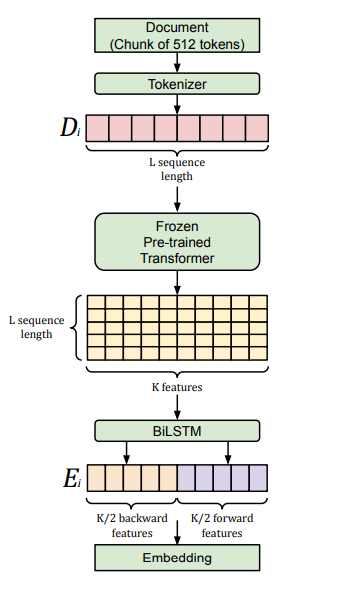


Figure 7: Topology of the PART used to extract embedding from documents. (Taken From "PART: Pre-trained Authorship Representation Transformer", page 2[1])

### 4.1 Pre-Processing

Before using the data for analysis or prediction, processing the data is important. In order to prepare the text data for the model building text preprocessing is performed. Some of the preprocessing steps are removing punctuations, removing URLs, removing Stop words, lower casing, tokenization, stemming, lemmatization.

#### 4.1.1 Chunking

Each author documents-set is merged with a separator token in-between, resulting in a large pre-tokenized text written in its entirety by this author. The pre-tokenized data is being split in chunks of L tokens. If an author ends up with two chunks or more to his name, then each chunk is considered separately for training. Otherwise, the author is being excluded. Like most NLP models, the data is split into a training set and a testing set.

#### 4.1.2 Tokenization

After cleaning the data, the next phase of the pre-processing of the data is Tokenization. In this phase the text is broken down to smaller pieces such as sentences or words, those pieces are called tokens. The objective of tokenization is to try to comprehend the meaning of the text by analyzing it in smaller sections and units. Popular tokenization methods are: White Space Tokenization, Dictionary based Tokenization, Ruled-Based Tokenization, Regular Expression Tokenization, Penn Treebank Tokenization, Spacy Tokenization, Moses Tokenization and Subword Tokenization and more. After tokenization, the L tokens are transferred to the next level in the architecture.

### 4.2 Frozen Transformer

The main goal of the frozen transformer is to extract features for training the model, with speed, from the L tokens it is given. Therefore, while training the model, the layers of the pre-trained transformer are being frozen, meaning the weights of the transformer are kept unmodified. Those features are called semantic word embedding, they are a matrix with dimension of LxK, containing the L semantic word embeddings sized K; a word embedding of K features for each input token.

### 4.3 BiLSTM

In order to interpret the L word embeddings, they are sent as L sequences to a BiLSTM, where each K sized word embedding passes through the forward LSTM pass and through the backward LSTM. The passes output 2 features vectors sized K/2, which we then concatenate into one K sized final Embedding.

## 5 Project Review and Process Description

## 5.1 Research Process

The starting point of the research process was a profound understanding of the paper "PART – Pre-trained Authorship Representation Transformer". This paper displayed the suggested architecture to approach the Authorship Attribution task in a semi-supervised contrastive learning method. For this reason, the next step of the research process was to study the fields of Authorship Attribution, Contrastive Learning and a deep understanding of the PART architecture's components. One of the key components in PART is the pre-trained frozen transformer.

Before heading for training and testing the model, we contacted the original paper's writers and consulted with them about how we should approach the task of training the model for our purposes. They informed us that the model's weights and biases were not published with the source code to GitHub, hence the model needs to be trained prior to finetuning the model to field of Plagiarism detection, especially with works written by William Shakespeare.

In the conversation we had with them, we told them that our goal is to take the task of authorship embedding to the field of plagiarism detection. Therefore, they recommended us to omit the mails datasets. Hence, the datasets that we agreed that are still relevant to our task are the Gutenberg and the blogs datasets.

They mentioned in the paper that are not enough datasets that were created for authorship attribution and profiling. The Gutenberg and the blogs datasets are the closest datasets to those tasks, that is why original paper's writers chose to use them and we chose to follow them also. Another reason why he did not choose to replace those datasets was because our project's goal – finetune the model to plagiarism detection. Different datasets that fed into the model lead to different results. It is known that different runs on the same datasets can output various results. However, we anticipate that different datasets would output more distant results in comparison to the result that have been shown in the original paper.

They instructed us how to download those datasets in order to start training the model. Worth mentioning their willingness to help us with many problems that we faced during the research process, from the silliest problem up to the more complex one.

Following the recommendation mentioned above, we began our journey by first downloading the datasets. Downloading the blogs dataset directly from the Kaggle website was fairly easy, it had a size of around 780 MB. In contrast, the process of downloading the Gutenberg datasets proved significantly more intricate. In order to download the dataset, we had to clone the GitHub repository of the "Standardized Project Gutenberg Corpus". The instructions were very easy, but the setback was the downloading time, mainly due to its size – over 70,000 books. In addition, we took only the books that are written in English. The next step was to pick the first 500 books out of the Gutenberg corpus and continuing with them alone, due to the insufficient means we had, which were not able to process this large amount of data.

For the task of plagiarism detection, especially with works written by William Shakespeare, our supervisors gave us two datasets of books. The first database is a folder full of Shakespeare novels, part of them have been taken from the Gutenberg dataset. The second one is the imposters dataset. This dataset is a folder that consists of many Authors folders of texts they have written.

After having all the datasets, all that is left to do in order to start training the model was to clear the data and to pre-process it to fit the model input structure. We followed the exact steps that were published in the GitHub repository for cleaning and preprocessing the Gutenberg and the blogs datasets. In the same manner, we cleaned and pre-processed the Shakespeare and the imposters datasets with slight modifications. Lastly, we split each one of the datasets to training and testing sets.

Following the steps above we started the step of training the model. As mentioned before, the model was published without its weights and biases, therefore we had to train it from scratch. We have opted to use the transformer model that is used in the original paper – RoBERTa Large.

We encountered difficulties while trying to run the training and testing processes of the model, when following the paper's hyperparameters. The means we had in our possession were the key issue – they were not sufficient. Our personal computers' specifications are for general use and not able to handle the complexities of the transformer and the LSTM head that the model consists of.

In an attempt to overcome this problem, we used Google Collab's resources instead. Google Collab has 3 main plans for users to choose from: free, Collab Pro and Collab Pro+. At first, we tried running the model with the default and free plan. We first used the hyperparameters that the original paper's writers recommended us to use, but the program collapsed. Before reaching out to the premiums plans that Google Collab offers, we tried feeding the model with lighter hyperparameters, unfortunately leading to the same results.

After utilizing the free package to the fullest, we decided to try the Collab pro plan. This plan gives faster GPUs, more memory and 100 compute units. As we did in the default plan, we started to train the model with the original hyperparameters and later started to mitigate them. In the end, we were able to run the model with only one dataset out of the 3 used in the original paper– the books dataset, with smaller hyperparameters in comparison to the starting point. Running the model did not last long enough to get decent results and we utilized all of the plan resources we paid for. We were obligated to make drastic changes to the hyperparameters, reducing the datasets and training time of the model, which affected the results the model was able to achieve.

We tried to get the best results possible, that is why we bought this plan 3 times, for the rounds of training and testing. We got similar results every time we trained the model. We did not have the financial means or the computational means to extend our research further than that. We were not able to run the model on different transformers or with the max values of the hyperparameters to get the desirable results, so we decided to continue with the weights and biased that we have managed to get and started testing the model.

Later, we moved on to attacking the plagiarism detection problem with anonymous texts that are suspected to have been written by William Shakespeare. We used texts which are known to have been written by Shakespeare and the imposters texts, provided to us by our supervisors.

## 5.1.1 Use Case Diagram

In order to study the use of PART for plagiarism detection, a system is suggested with the option test the model. The system allows the user to select a text to extract the authorship embedding for and later on to compare it to the embedding of Shakespeare's writing style, to detect plagiarism. To illustrate this, a Use Case diagram of the system is shown in the following figure.

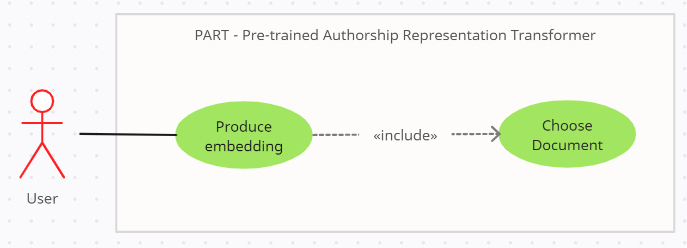


Figure 8: The Suggest GUI for The System

# 5.1.2 Evaluation/Verification Plan

To test the aforementioned systems, the functionality testing table is represented in the following table.

|  |  |  |
| --- | --- | --- |
| **Test Number** | **Test Subject** | **Expected Result** |
| 1 | Press “Choose Document” | The window “Choosing Document” appears with “Create Embedding” button. |
| 2 | Press "Create Embedding" | “No Selection” error window opens. |
| 3 | Press "Ok" | “No Selection” error window closes. |
| 4 | Select a document from the dropdown lists | Document is selected |
| 5 | Press "Create Embedding" | “Loading Embedding Document” window appears.  When finishes, loads window “Document’s authorship embedding display” |

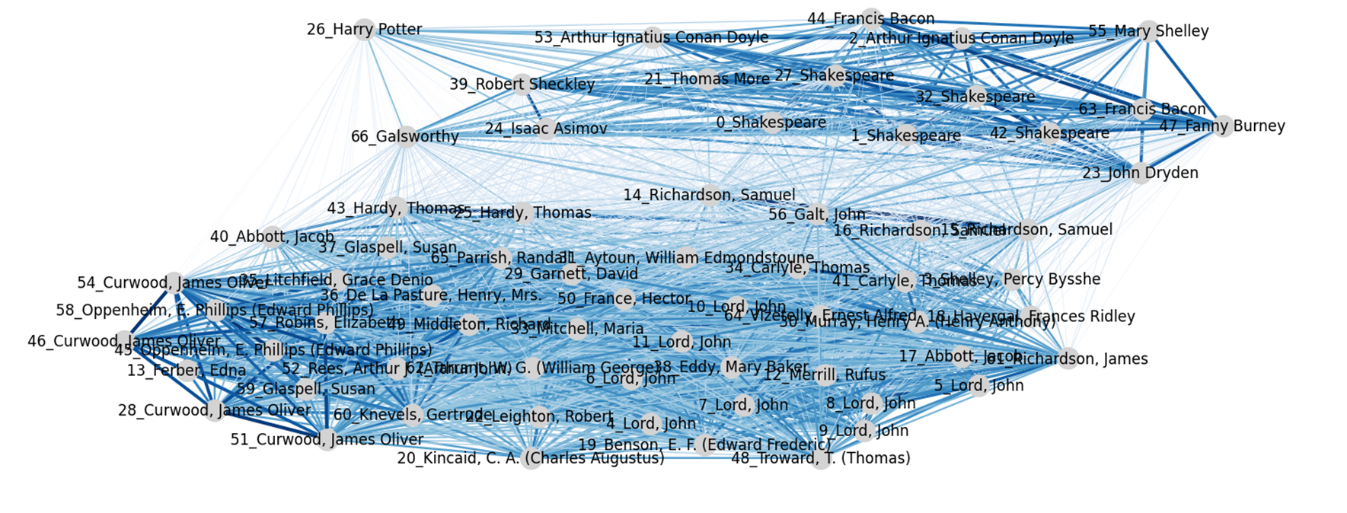
## 5.2 Results and Conclusions

The goals of the project were achieved partly. We were able to implement the discussed architecture. The training itself had to be under limitations due to shortcomings in the resources we had. In the end we achieved results, but the results could be better, given better environmental conditions, including better hardware, more time etc.

Effort wise, we did our best to extract the maximum out of this project. We have consulted with many figures in order to achieve the best results we could with the means we had on our hands. In retrospect, we could have tried running this model in the first part of our project, to alert our supervisors about the possible delay problems when dealing with the downloading of the datasets and the high resources necessary to run the model much earlier.

As mentioned before, the model is supposed to produce authorship embeddings. Instead of checking the raw embeddings, we chose to display the results in a more intuitive way. A figure that has been used in the original paper to display the results represents the cosine similarity between each embedding in the testing sets. We made 3 test rounds as was planned in part A of this project.

**Result of the 1st round:**

****



We can see the books written by Shakespeare very close to each other, and the imposter texts in a bulk around them as well. This indicates how the model predicted the closeness in style of the author between Shakespeare himself and the texts copied from him, from the testing set. In a different "cluster" the rest of the books from the Gutenberg corpus which aren’t suspected to have been written by Shakespeare are placed in different distances from one another.

**Result of the 2nd round:**

תמונה שמכילה טקסט, שרטוט, ציור, תרשים

התיאור נוצר באופן אוטומטי



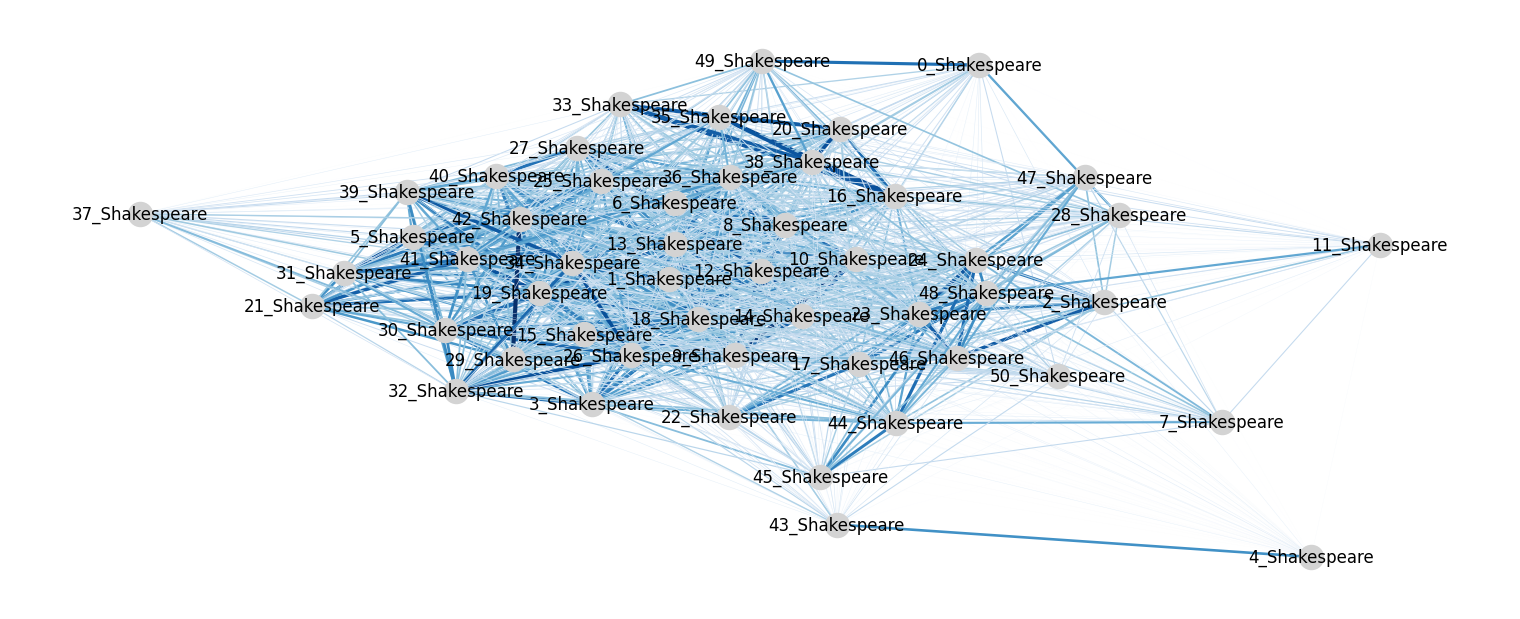
**Result of the 3rd round:**

תמונה שמכילה טקסט, ציור, שרטוט, צילום מסך

התיאור נוצר באופן אוטומטי



The results are consistent and as expected, Shakespeare and the suspected imposters are translated by the model to embeddings which are close, while other texts by other authors are found in different areas.

Another test we made was we tested the model on the entirety of the Shakespearean texts' dataset. This was the resulted cosine-similarity diagram:  




We can see there is a **kernel** of texts, meaning they are all close to each other in the author’s writing style. There are **3 texts** which are significantly **distanced from the group**. Those texts are **attributed to Shakespeare**, however, there is **a chance they were wrongly attributed**, and have been written by somebody else.

# 5.2.1 Future Research

In conclusion, we found the model is able to answer the plagiarism detection task with authorship embeddings. We recommend first training the model in its original hyperparameters with the addition of the 2 datasets of Shakespeare and the Imposters datasets to get better results.

In addition, an idea for expanding the research plagiarism detection is to try finetune the model to deal with programmers' codes. A designated modifications needs to be done to that dataset in order to send it into the model. The plagiarism detection in codes, in our modest opinion, could be game changer solution.

# 6. User documentation

This section consists of a user's guide-operating instructions with gives instructions to the user how to run the GUI. In addition, it is also displaying maintenance guide with 2 UML diagrams.

# 6.1 General Description

The GUI was built for the user to be able to find the value of the cosine similarity between the authorship embeddings of texts to the embedding of William Shakespeare, with the intention to detect plagiarism.

Within the GUI the user can choose a document from a selection of texts which are suspected to have been written by Shakespeare. The user can produce an embedding for the chosen text's author and get a percentual comparison between that embedding and William Shakespeare's.

This allows the user to assess the probability that the text they chose was actually written by Shakespeare.

Audiences that may benefit from such a network are researchers from the NLP field, people who work with detecting plagiarism. The network can be reused for different purposes, it can be utilized for hate speech detecting, forensics, social networks analysis, identifying fake news and more.

The model was trained on 3 datasets: part of the Gutenberg corpus (500 English books written by different authors), Shakespeare dataset (50 texts signed by William Shakespeare) and an imposters' dataset (120 texts which are suspected to be originally written by Shakespeare).

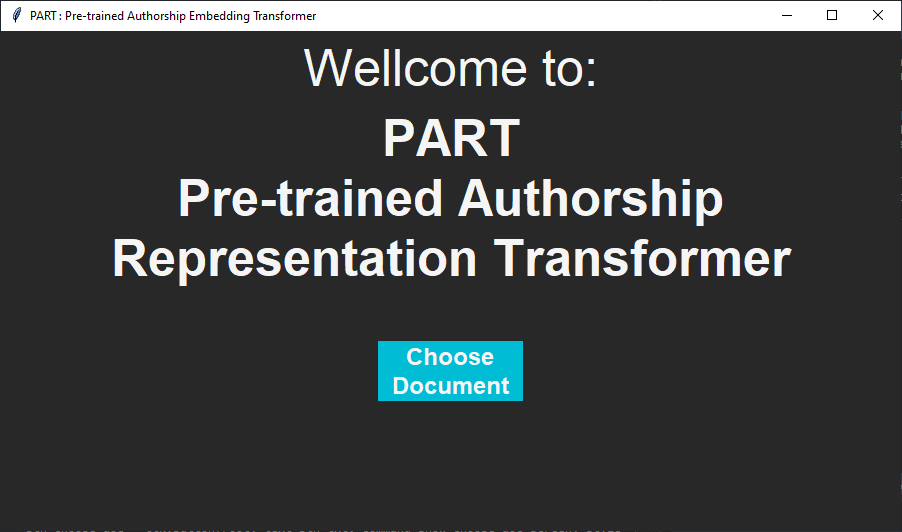
The network relies on a pre-trained PART model.In order to use the GUI, the user must first train the model. The instructions for the process are detailed in the following Google Colab notebook:

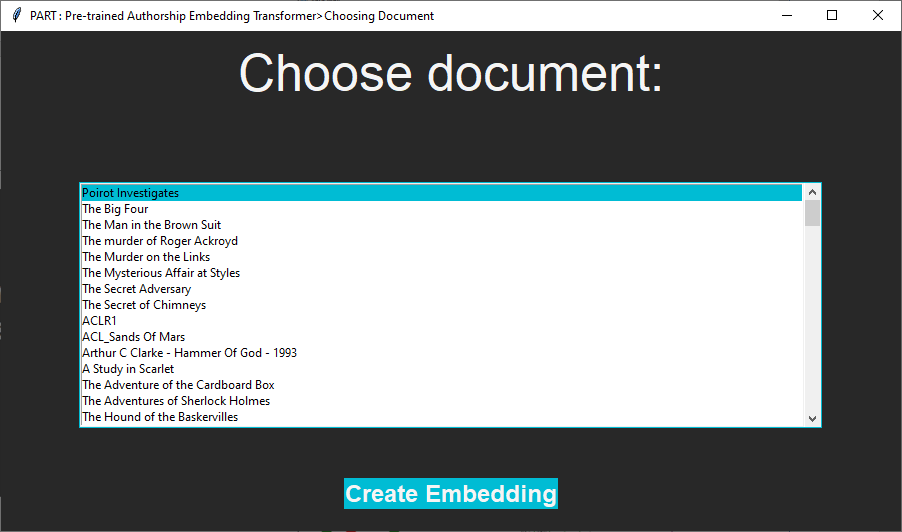
<https://github.com/TuvalZit/Capstone-Project-23-1-R-18/blob/main/PART_All_in_one_notebook.ipynb>

After the training, the user can choose different datasets for the testing process in the network, download them to their local folder, and run them with the provided GUI. The user can also produce measuring results for their testing data.

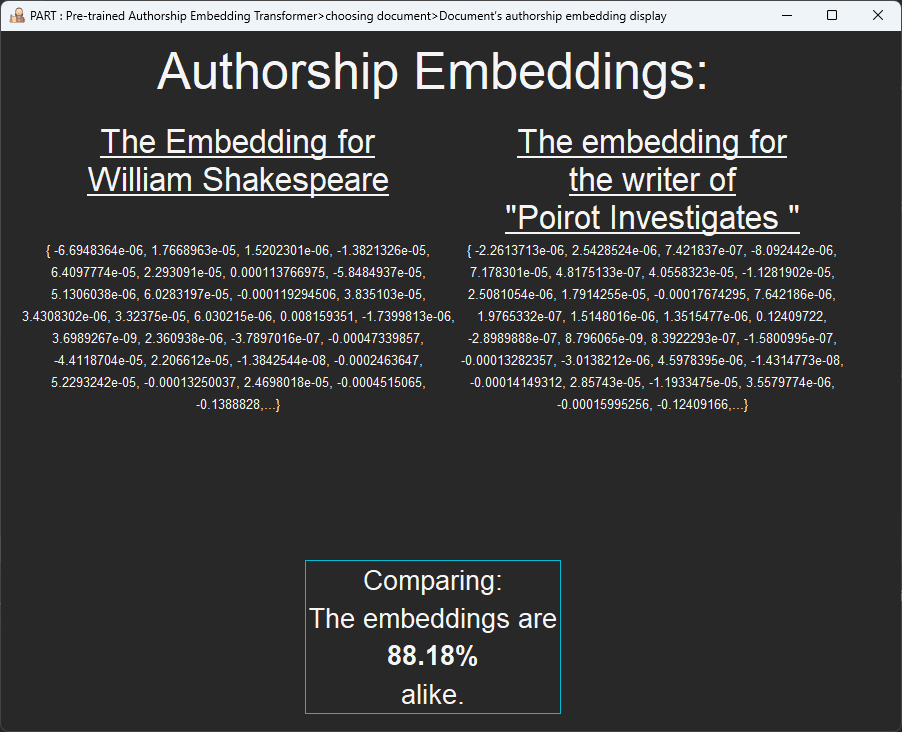
In the program folder should be the following folders: data\_base – which consists of imposters folder, which is the folder of the collection to choose documents from and a pickle document: "total\_embedding.pickle" that holds all the documents's embeddings and their information for the program to read from. Backend – Consists of "GetEmbedding.py". These python documents should be in it: "MainWindow.py", "ChooseDocuments.py", "CreateEmbeddings.py", "Common.py", "Results.py".

**Here are the instructions for using the GUI**

The opening screen: in this screen the user clicks on the "Choose Document" button.  


Choosing document screen: in this screen the user can scroll down a list of documents out of the imposters' dataset, click on a certain row, then click on the "Create Embedding" button in order to continue to the results screen.  


Results screen: in this screen the user can see a comparison between their chosen document's produced embedding by the model, and the embedding produced for Shakespeare. The user can also find a percentage of similarity between the two, and decide whether plagiarism indeed took place.



# 6.2 Maintenance Guide

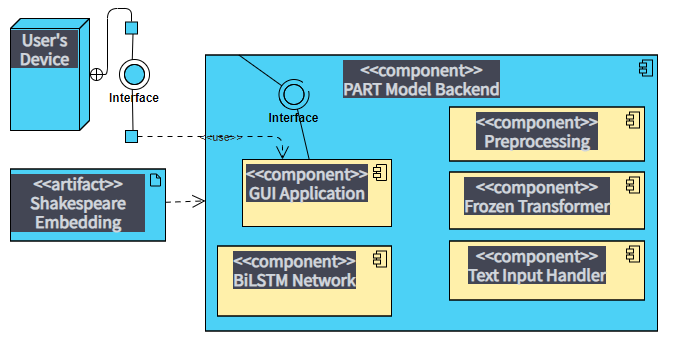
In this section 2 UML diagram will be shown – the package and deployment diagrams.

## 6.2.1 Package Diagram

To illustrate the program, a Package diagram of the system is shown in the following figure. System's GUI is presented in the following figure.

## 4.3.2 Deployment Diagram

## 6.2.2 Deployment Diagram

The project works with the latest version on Python. It works with RoBERTa large, but other encoder-transformer models which are more sophisticated can and should be tested for results to see if better can be achieved.

## 6.3 Databases Structures

When wanting to train the model, one must have a decent GPU processor, above 16GB RAM. If using Google Collab, it is recommended to buy the Google Collab Pro package that opens GPU accelerators and more RAM memory.

When wanting to work with the datasets of the model, one must follow the steps in the Google Collab notebook that is attached to the project GitHub repository. The actions that are open to do with the datasets are cleaning the data, gathering it and preprocessing it before sending it to the model.

When following the Google Collab network to train the model, there are specific code cells that are designated to install the required libraries. When trying to run the GUI, to compare authorship embeddings, all is needed is to watch the imports and to make sure that all of the libraries and files are in the GUI directory.

# 7. References

[1] J.Huertas-Tato, A.Martín, A.Huertas-García & D.Camacho. PART: Pre-Trained Authorship Representation Transformer. arXiv preprint *arXiv:2209.15373*, 2022.

[2] Efstathios Stamatatos. A survey of modern authorship attribution methods. Journal of the American Society for Information Science and Technology, 60(3):538–556, 2009.

[3] Zhou Ai, Zhang Yijia, Wei Hao, and Lu Mingyu. Lda-transformer model in chinese poetry authorship attribution. In China Conference on Information Retrieval, pages 59–73. Springer, 2021.

[4] Anastasia Fedotova, Aleksandr Romanov, Anna Kurtukova, and Alexander Shelupanov. Authorship attribution of social media and literary russian-language texts using machine learning methods and feature selection. Future Internet, 14(1):4, 2022.

[5] Benjamin Murauer and Günther Specht. Developing a benchmark for reducing data bias in authorship attribution. In Proceedings of the 2nd Workshop on Evaluation and Comparison of NLP Systems, pages 179–188, 2021.

[6] Julian Hitschler, Esther van den Berg, and Ines Rehbein. Authorship Attribution with Convolutional Neural Networks and POS-Eliding. ACL Anthology, pages 53–58, September 2017.

[7] Lucy Simko, Luke Zettlemoyer, and Tadayoshi Kohno. Recognizing and imitating programmer style: Adversaries in program authorship attribution. Proc. Priv. Enhancing Technol., 2018(1):127–144, 2018.

[8] Robert Deibel and Denise Löfflad. Style change detection on real-world data using an lstm-powered attribution algorithm. In CLEF (Working Notes), pages 1899–1909, 2021.

[9] Emilio Villa-Cueva, Ivan González-Franco, Fernando Sanchez-Vega, and Adrián Pastor López-Monroy. Nlp-cimat at politices 2022: Politibeto, a domain-adapted transformer for multi-class political author profiling. In Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2022). CEUR Workshop Proceedings, CEUR-WS, A Coruna, Spain, 2022.

[10] Asier Gutiérrez Fandiño, Jordi Armengol Estapé, Marc Pàmies, Joan Llop Palao, Joaquin Silveira Ocampo, Casimiro Pio Carrino, Carme Armentano Oller, Carlos Rodriguez Penagos, Aitor Gonzalez Agirre, and Marta Villegas. Maria: Spanish language models. Procesamiento del Lenguaje Natural, 68, 2022.

[11] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084, 2019.

[12]  Ipek Baris Schlicht and Angel Felipe Magnossão de Paula. Unified and multilingual author profiling for detecting haters. arXiv preprint arXiv:2109.09233, 2021.

[13] Antonio Theophilo, Romain Giot, and Anderson Rocha. Authorship attribution of social media messages. IEEE Transactions on Computational Social Systems, 2021.

[14]  Mara Chinea-Rios, Thomas Müller, Gretel Liz De la Peña Sarracén, Francisco Rangel, and Marc Franco-Salvador. Zero and few-shot learning for author profiling. In International Conference on Applications of Natural Language to Information Systems, pages 333–344. Springer, 2022.

[15] Sinong Wang, Han Fang, Madian Khabsa, Hanzi Mao, and Hao Ma. Entailment as few-shot learner. arXiv preprint arXiv:2104.14690, 2021.

[16] Wenpeng Yin, Nazneen Fatema Rajani, Dragomir Radev, Richard Socher, and Caiming Xiong. Universal natural language processing with limited annotations: Try few-shot textual entailment as a start. arXiv preprint arXiv:2010.02584, 2020.

[17] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. arXiv preprint arXiv:2103.00020, 2021.

[18] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. arXiv preprint arXiv:2104.08821, 2021.

[19] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. Advances in Neural Information Processing Systems, 33:18661–18673, 2020.

[20] Xuming Hu, Li Lin, Lijie Wen, et al. Pair-level supervised contrastive learning for natural language inference. In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 8237–8241. IEEE, 2022.

[21] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering* 22.10 (2009): 1345-1359.

[22] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).‏

[23] Makarenkov, Victor. Utilizing Recursive Neural Networks for Contextual Text Analysis. Diss. Ben-Gurion University OF THE Negev, 2019.‏