Spring 2023

ML-Based User Authentication Through Mouse Dynamics

Sai Kiran Davuluri
San Jose State University

ML-Based User Authentication Through Mouse Dynamics

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sai Kiran Davuluri

May 2023

ML-Based User Authentication Through Mouse Dynamics

By

Sai Kiran Davuluri

Approved for the Department of Computer Science

San José State University

May 2023

Dr. Fabio Di Troia	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Pranav Cherukupalli	Software Engineer 3, Juniper Networks

ABSTRACT

ML-Based User Authentication Through Mouse Dynamics

Sai Kiran Davuluri

Increasing reliance on digital services and the limitations of traditional authentication methods have necessitated the development of more advanced and secure user authentication methods. For user authentication and intrusion detection, mouse dynamics, a form of behavioral biometrics, offers a promising and non-invasive method. This paper presents a comprehensive study on ML-Based User Authentication Through Mouse Dynamics.

This project proposes a novel framework integrating sophisticated techniques such as embeddings extraction using Transformer models with cutting-edge machine learning algorithms such as Recurrent Neural Networks (RNN). The project aims to accurately identify users based on their distinct mouse behavior and detect unauthorized access by utilizing the hybrid models. Using a mouse dynamics dataset, the proposed framework's performance is evaluated, demonstrating its efficacy in accurately identifying users and detecting intrusions.

In addition, a comparative analysis with existing methodologies is provided, highlighting the enhancements made by the proposed framework. This paper contributes to the development of more secure, reliable, and user-friendly authentication systems that leverage the power of machine learning and behavioral biometrics, ultimately augmenting the privacy and security of digital services and resources.

Index terms - Behavioral Biometrics, Mouse Dynamics, Transformer models, Recurrent Neural Networks

ACKNOWLEDGEMENTS

I extend my profound gratitude to my project advisor, Dr. Fabio Di Troia, for his unwavering support and encouragement throughout my time in graduate school. His expertise, guidance, and inspiration were crucial to the successful completion of this project.

I am also grateful to my committee members, Dr. Robert Chun and Pranav Cherukupalli, for their invaluable feedback and support during the development of this project.

I would like to express my heartfelt gratitude to my caring family that consistently motivates me to chase my dreams. Finally, I want to thank everyone who contributed to my unforgettable experience at San José State University. Your impact on my journey will be cherished for a lifetime.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The lightning-fast expansion of digital services combined with an ever-increasing reliance on the Internet for day-to-day activities has made it more important than ever before to safeguard access to sensitive information and online resources. The studies [1] and [2] describe how traditional authentication mechanisms such as passwords and tokens are vulnerable to a broad variety of cyberattacks, such as brute force, keylogging, and phishing. As a consequence of this, there is a requirement for authentication methods that are more cutting-edge and reliable to strengthen security and secure the data of users. The use of behavioral biometrics, which focuses on the distinctive patterns and traits of an individual's behavior, is one method that shows promise as a solution. In this group, mouse dynamics have distinguished themselves as a user authentication and intrusion detection technology that is both efficient and unobtrusive.

A detailed study on ML-Based User Authentication Using Mouse Dynamics for Intrusion Detection is described in this paper. The research aims to develop a robust and accurate machine-learning algorithm that not only identifies users based on their unique mouse behavior but also detects unauthorized access or malicious activities.

All of the tests in this study were carried out with the use of deep learning algorithms. The mouse dynamics data is preprocessed and made into embeddings using the transformer models. The paper is broken up into sections and begins by providing an overview of the motivation behind exploring mouse dynamics for user authentication and intrusion detection. It

then goes on to highlight the limitations of traditional authentication methods as well as the advantages of employing advanced machine learning algorithms for authentication.

A comprehensive review of the existing literature in the field of user authentication and intrusion detection using mouse movements is provided in Section 2. Followed by the introduction of a novel ML-based framework that leverages advanced algorithms like transformer models in combination with recurrent neural network models to accurately identify users and detect intrusions.

Section 3 provides a detailed overview of niche machine learning and deep learning techniques employed throughout this project. In Section 4, we detail the experimental dataset and discuss the experimental setting that was utilized to evaluate the proposed framework's effectiveness. Figure 1 below shows the overall setup for the experiments used in this investigation. The data is initially cleaned and prepared for usage. The initial trials are run using the preprocessed data on RNN models. The second series of tests involves trying out RNN models with embeddings derived from the transformer models using the preprocessed data. Section 4 concludes with visual representations of the experiment findings, which prove that our proposed system is capable of reliably identifying users based on their mouse dynamics and detecting intrusions.

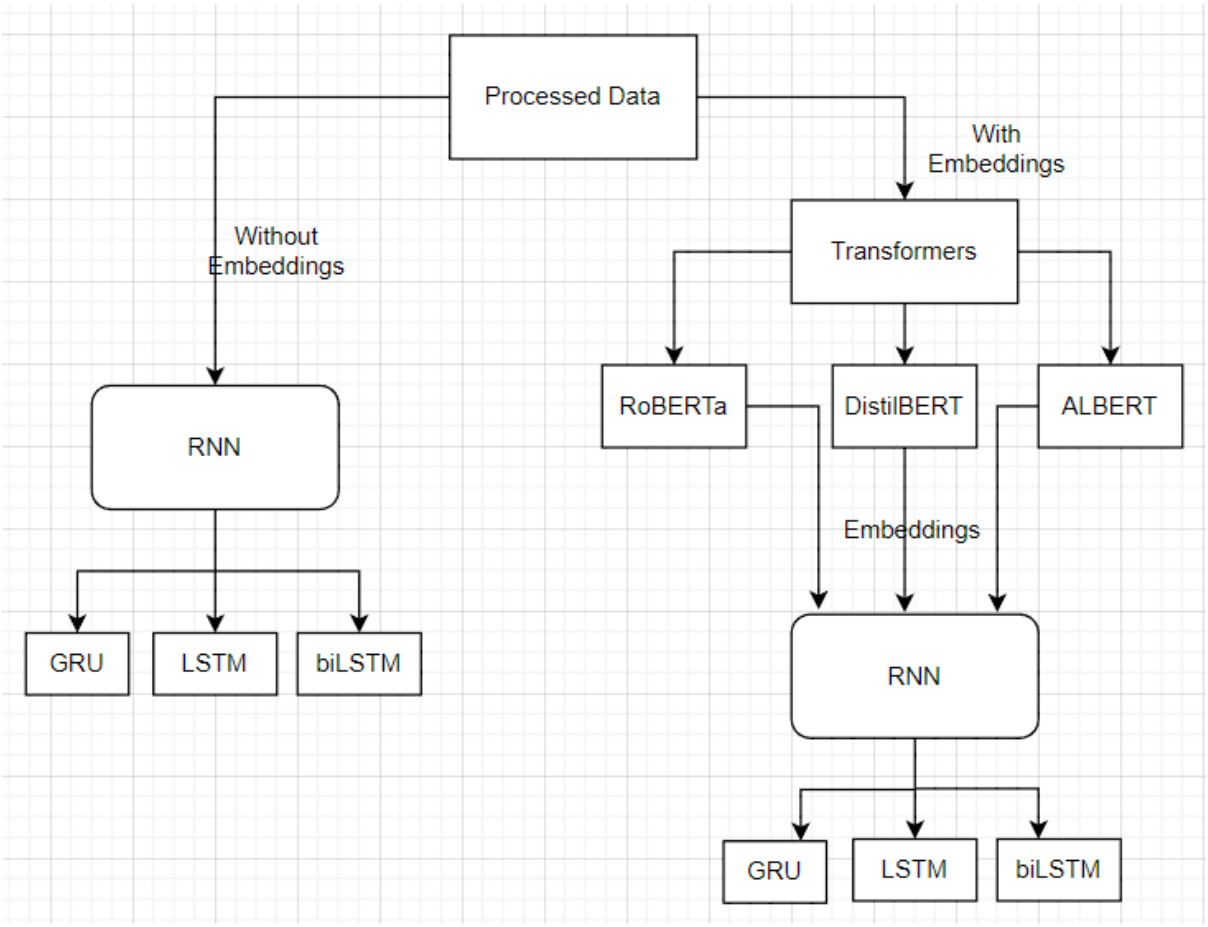

Figure 1: Layout of the Experiments

The paper concludes by providing a comparative analysis of other existing approaches. Highlighting the improvements achieved by our framework and summarizing the findings and contributions to the field of ML-Based User Authentication Through Mouse Dynamics for Intrusion Detection. It also emphasizes the significance of the work in advancing the study in this domain in section 5, and it suggests possible directions for future research in this area.

The purpose of this study is to contribute to the development of authentication systems that are more secure, reliable, and user-friendly. To that end, these systems will make use of machine learning and behavioral biometrics to bolster the safety and privacy of online services and data.

CHAPTER 2

Related Work

This section provides a summary of the research on User Authentication Using Mouse Dynamics that has been conducted in the field of machine learning so far. The security environment is always shifting, which means that there is a growing demand for reliable authentication systems, that do not violate users' privacy and are easy to use. The application of mouse dynamics, a sort of behavioral biometrics, has emerged as a potential solution for user authentication. This method takes advantage of the distinctive patterns in an individual's mouse usage to differentiate between real users and impostors.

A comprehensive review of the introduction to biometric recognition is provided in [3], discussing various biometric modalities, including physiological and behavioral biometrics. The authors also explore the role of machine learning algorithms in biometric recognition systems and their applications in security and user authentication.

Then, Gamboa and Fred in [4] describe a behavioral biometric system that makes use of human-computer interaction. More specifically, they center their attention on mouse motions and keystroke dynamics. The authors propose a method that employs machine learning techniques to analyze user conduct and gauge the system's efficacy. According to their findings, the combination of mouse dynamics and keystroke analysis may be able to give a solution for user authentication that is both robust and trustworthy.

Using mouse movements for authentication as a behavioral biometric is explored in the literature review presented in [5]. The authors conduct an analysis of mouse movement patterns using machine learning techniques, which allows them to evaluate the effectiveness of

their strategy. They highlight the advantages of employing mouse dynamics for authentication and make recommendations for further study on the topic.

According to Chen et al. in [6], a functional real-time authentication system that employs mouse dynamics for current user identification tracking has demonstrated efficiency in predicting an authentic user with precision. This success was demonstrated by the system's ability to track the identity of the user in real time. During the five seconds that the verification took place, it obtained a False Rejection Rate (FRR) of 2.86% and a False Acceptance Rate (FAR) of 4.00%, where FAR is the percentage of unauthorized persons are wrongly accepted (fraud rate), and FRR is the percentage of authorized individuals are wrongly denied (insult rate).

Although many studies have been conducted on the topic of intrusion detection using mouse dynamics, these efforts have been hampered by the lack of accessible general-purpose datasets that include unrestricted mouse usage data. This issue was resolved in 2016 when the Balabit data set [7] was made accessible in preparation for a data science competition. Despite the small volume of users who were included in this study, this data set is regarded to be the first appropriate one that was made publicly available.

Using the Balabit data set as a basis, Hu et al [8]'s proposal to employ the CNN algorithm to authenticate users through the use of mouse dynamics was presented. Using a predetermined set of guidelines, the creators of this method transformed mouse actions into JPEG pictures. They were able to attain a maximum accuracy of 81% with a FAR of 2.96%, and an FRR of 2.27%, and both of those results in just seven seconds. The main advantage of this approach is that no model feature extraction is required, and no user activity data is lost in the

process. In addition, it is not necessary to make use of any additional techniques to extract features from the dataset.

The study described above was expanded upon by Lee et al., who published their findings in [9]. In this paper, the authors provide an innovative strategy for user authentication using Natural Language Processing (NLP) approaches in combination with mouse dynamics analysis. The authors propose a method that combines features from user-specific mouse movement patterns and NLP-based text analysis with recurrent neural networks to create a robust, multi-modal authentication system. By doing so they were able to achieve maximum accuracy of 95% with the biLSTM model.

The work that was done in [10], which presents a machine learning-based strategy to identify malware by employing contextualized vector embeddings, has served as a source of motivation for this research. The author employs pre-trained language models like BERT, DistilBERT, ALBERT, and RoBERTa to generate embeddings of textual properties extracted from malware samples. The meaning and structure of the characteristics are captured by these embeddings in the context of the full sample. This project's goal is to improve upon the existing models found in [9].

Using pre-trained language model embeddings, this study explains how recurrent neural network models function. The experiments conducted in this current research evaluate the performance and accuracy of recurrent neural network models when used in tandem with embedding creation utilizing different transformer models.

CHAPTER 3

Background

Within this section, the RNN and transformer models that are utilized in this research are broken down into their parts in greater depth. It provides a condensed description of the transformer architecture as well as the models that were utilized for this project. The GRU, LSTM, and biLSTM are the RNN models that are being explored in this article. Meanwhile, the BERT, RoBERTa, DistilBERT, and ALBERT transformer models are being brought up in this discussion. The experiments used to confirm the findings will be discussed in the next section.

3.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) [11] is a network that takes sequential data or input that has been accumulated over a period of time. Apps such as Siri, voice search, and Google Translate all make use of these deep learning algorithms. These techniques may also be found in Google. These are used rather commonly for solving issues involving numbers or times, such as translating languages, applying natural language processing (NLP), speech recognition, and picture captioning. These methods may be utilized to handle a wide range of problems, including those pertaining to order and time, amongst others. Training data is used to help recurrent neural networks grow, much like it is used to help convolutional neural networks (CNNs) and feedforward neural networks. They are different from other systems that are comparable in that they have a "memory," which gives them the ability to change both the input and the output at any given moment by drawing on knowledge from earlier entries in the system. This sets them apart from other systems that are comparable. RNNs depend on the variables that came before them in the sequence for their outcome. This is in contrast to the

traditional deep neural networks, which consider both the inputs and outputs to be independent of one another. Despite the fact that these events could influence a particular sequence's outcome, unidirectional RNNs are unable to include them in their forecasts.

Let's take an expression that's often used to describe how someone feels when they're sick—for example, "feeling under the weather"—and use it to help explain RNNs [12]. It is necessary to express the idiom in precisely that order for it to make any kind of sense. As a consequence of this, recurrent neural networks are required to take into consideration the location of each word within the idiom, and they use this knowledge to make predictions for the following word in the chain.

Recurrent networks are distinguished from other types of networks in part because their parameters are shared between all of the network's layers. Unlike feedforward neural networks, recurrent neural networks maintain a constant weight parameter throughout all network layers. The weights at each node in feedforward neural networks are not uniform. However, these weights are still refined via backpropagation and gradient descent in order to provide reinforcement learning. Standard RNNs, seen in Figure 2 below [11] illustrate the RNN's repeating module.

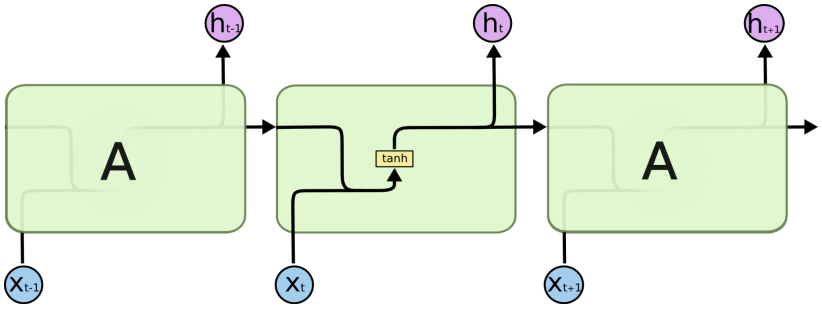

Figure 2: The Repeating Module in a Standard RNN

Backpropagation through Time (BPTT) [13] is a technique that is used to compute the gradients in recurrent neural networks. This approach is somewhat distinct from regular backpropagation in that it is designed to work exclusively with sequence data. The fundamentals of BPTT are identical to those of traditional backpropagation, in which the model educates itself by comparing the errors found in its output layer to the faults found in its input layer. Because of these calculations, we can modify the model's parameters so that it better fits the data. Since feedforward networks do not exchange parameters between layers, the conventional approach is distinct from the BPTT approach, which totals errors at each time step.

At this point in the process, RNNs typically face two concerns that are referred to as bursting gradients and disappearing gradients. Both of these issues can be problematic. The nature of these issues may be described as being determined by the size of the gradient, which can also be thought of as the slope of the loss function along the error curve. When the gradient is already too low, it will continue to decline, which will cause an update to the weight parameters until they are so low that they are no longer meaningful; that is, they will hit 0. When the gradient is already too low, it will continue to fall. This will cause an update to the weight parameters. When this occurs, the algorithm is prevented from acquiring new knowledge. Exploding gradients are possible to form when the gradient is very big, and this can lead to the model being unstable. In this particular case, the model weights will continue to expand until they become unmanageably enormous and are eventually represented as NaN. After reaching this point, the model weights will be rendered as NaN. One potential solution to these issues would be to simplify the RNN model by removing some of its more involved

components. One method for achieving this goal is to reduce the number of hidden layers that are present inside the neural network.

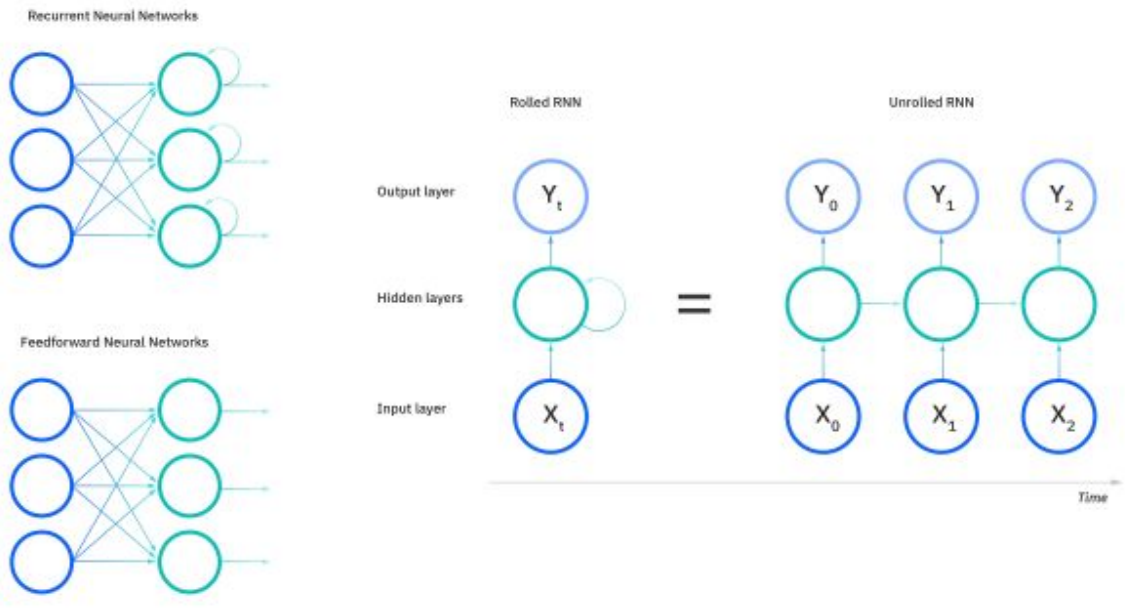

Figure 3: Recurrent Neural Networks

Different varieties of neural networks are depicted in Figure 3 [11]. To combat the vanishing gradient problem that might arise while training RNNs on long sequences, various variants of RNNs have been developed, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) [14]. When RNNs are trained on longer and longer sequences, this issue may appear.

3.1.1 LSTM

Commonly abbreviated as "LSTM," the Long Short-Term Memory architecture was initially defined by Sepp Hochreiter and Jurgen Schmidhuber in 1997 [15]. Common RNNs cannot learn dependencies over lengthy periods of time because of a vanishing gradient

problem that arises during training on such sequences [16]. Long short-term memories (LSTMs) were created as a workaround for this issue. The long short-term memory (LSTM) model has been widely adopted due to its impressive performance on several sequence modeling tasks [17]. NLP, voice recognition, and time series prediction are all examples of such activities.

The LSTM cell is the most important part of the LSTM architecture which is depicted in Figure 4 [18]. This cell is the one that is in charge of preserving and updating the hidden state, which is also referred to as the cell state. The LSTM cell is made up of a few gates that regulate the flow of information within the cell, and they are as follows:

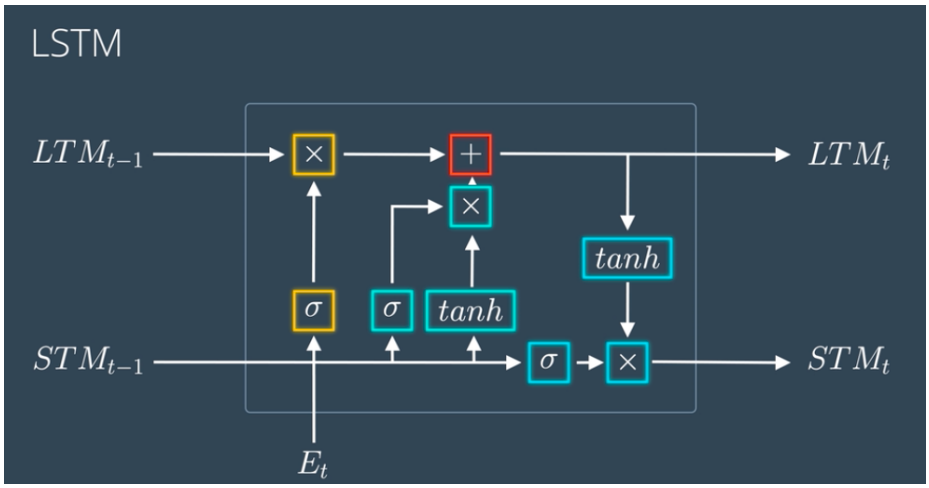

Figure 4: LSTM Architecture

- Input gate (i): This gate controls how much the newly received data is integrated into the cell's present state. The sigmoid activation function is used by the input gate to produce a value between zero and one. The current input (x_t) and the previous hidden state (h_{t-1}) are used to calculate this value. Having a number near 1 implies that the input is useful and should be used; having a value close to 0 indicates that the input is irrelevant and should be ignored.

- Forget gate (f): This gate controls how much the newly received data is integrated into the cell's present state. The sigmoid activation function is used by the input gate to produce a value between zero and one. The current input (x_t) and the previous hidden state (h_{t-1}) are used to calculate this value. Having a number near 1 implies that the input is useful and should be used; having a value close to 0 indicates that the input is irrelevant and should be ignored.
- Cell state update: The cell's current state (c_t) is the sum of the outputs of the input gate, the forget gate, and the prior state of the cell. First, we utilize a tanh activation function to determine a candidate cell state (c'_t) based on the current input (x_t) and the previous hidden state (h_{t-1}). Next, the candidate cell state receives a one-by-one application of the input gate's (i_t) output. The output (f_t) of the forget gate is then multiplied, one element at a time, by the previous cell state (c_{t-1}). Finally, by adding the results of these two multiplications, we get the cell's current state (c_t).
- Output gate (o): During each cycle, this gate determines which parts of the current cell state (c_t) will be replaced by the hidden state (h_t). The output gate uses a sigmoid activation function to compute a value between 0 and 1 based on the current input (x_t), the previous hidden state (h_{t-1}), and the current cell state (c_t). Applying the tanh activation function on the current cell state (c_t) and then multiplying the result by the output of the output gate (o_t) generates the new hidden state (h_t), one element at a time.

The cell's unique architecture in LSTM allows it to learn long-term dependencies by selectively incorporating, updating, and outputting information through its gates. This makes

LSTMs particularly suitable for tasks that involve long sequences [17] or requires capturing dependencies over long time horizons [20].

3.1.2 GRU

Gated Recurrent Unit (GRU) architecture for RNNs was introduced in 2014 by Cho et al. in [21]. As a simpler alternative to LSTM networks, GRUs were created to address the vanishing gradient problem and successfully capture long-term relationships in sequence data. Natural language processing, audio recognition, and time series prediction are just some of the areas [22] where GRUs have been shown to achieve comparable performance to LSTMs, although often requiring fewer parameters and lower CPU resources.

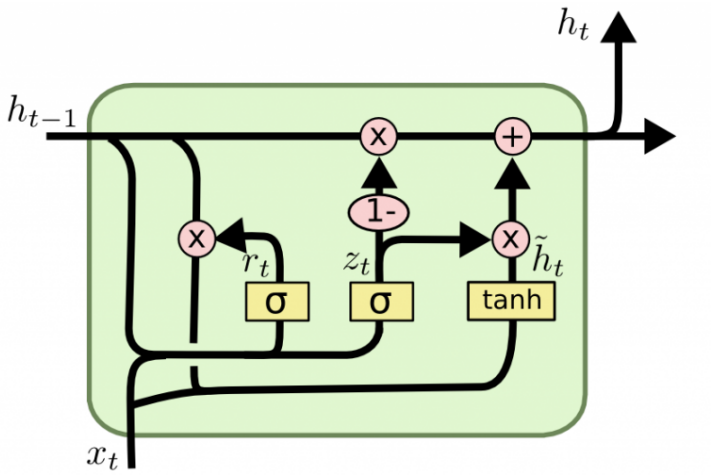

Figure 6: GRU Architecture

The GRU cell, which stores and refreshes the secret state, is at the heart of the GRU architecture seen in Figure 6 [23]. There are two gates in a GRU cell that regulate the flow of data:

- Update gate (z): This gate controls the proportional addition of the previous hidden state (h_{t-1}) to the present input. Update gates use a sigmoid activation function applied to the

current input (xt) and the previous hidden state (h_{t-1}) to output a value between 0 and 1. The closer the value is to 1, the less weight the new input information should have on the hidden state, whereas the closer it is to 0, the more weight the previous hidden state should be kept.

- Reset gate (r): This gate determines how much of the previous hidden state (h_{t-1}) should be used in the calculation of the candidate hidden state (h'_t). Like the update gate, the sigmoid activation function takes the current input (x_t) and the previous hidden state (h_{t-1}) and outputs a value between 0 and 1. When the value is close to zero, the candidate hidden state derived from the new input information is used, and when the value is close to one, the prior hidden state is used.
- Candidate hidden state (h'_t): The candidate hidden state is derived by fusing the current input (x_t) with a modified version of the prior hidden state (h_{t-1}). After element-wise multiplying the output of the reset gate (r_t) by the previous hidden state, the tanh activation function is applied to the current input. This candidate hidden state is an alternate hidden state that incorporates the updated input data and the baseline hidden state.
- Hidden state update (h_t): The final hidden state (h_t) is a product of the output (z_t) of the update gate and the candidate hidden state (h'_t). To be more specific, the update gate multiplies the previous hidden state, whereas the candidate hidden state is multiplied by ($1 - z_t$). The resulting sum is fed into a formula to get the most recent hidden state (h_t).

The GRU cell's architecture allows it to learn long-term dependencies by selectively incorporating, updating, and outputting information through its gates. This makes GRUs

particularly suitable for tasks that involve long sequences or require capturing dependencies over long time horizons. Compared to LSTMs, GRUs have a simpler structure, with fewer gates and no separate cell state, which can lead to faster training and reduced computational complexity in some cases. However, the choice between GRUs and LSTMs [24] depends on the specific task and dataset, as both architectures are effective in different scenarios.

3.1.3 BiLSTM

BBidirectional Long Short-Term Memory (BiLSTM) [25] networks are a variant of the conventional LSTM architecture that is designed to capture both forward and backward dependencies in sequence data. Natural language processing and speech recognition are two examples of activities that require transitioning from one sequence to the next. Accurate predictions may be made in many of these tasks with the assistance of context from both the past and the future. BiiLSTM is a technique that allows one to make use of the context's bidirectional nature by processing the input sequence in both directions and integrating the information gained from forward and backward passes [26]. This allows one to take advantage of the fact that the context is bidirectional.

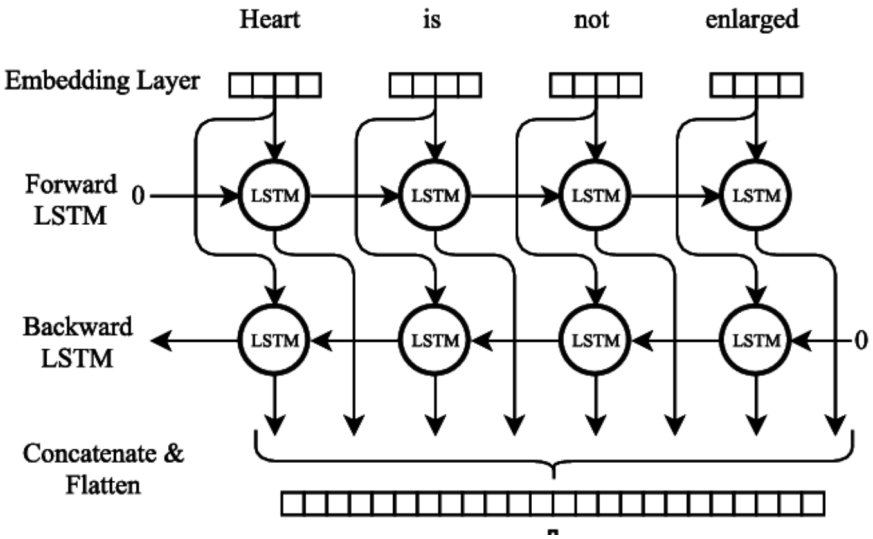

Figure 7: BiLSTM Architecture

A BiLSTM is made up of two independent LSTM layers that operate in tandem with one another. These layers are referred to as the forward LSTM layer and the backward LSTM layer, respectively. The input sequence is processed by the forward LSTM from the beginning all the way through to the end, whereas the backward LSTM processes the sequence in the opposite direction, from the end all the way through to the beginning. The output of the BiLSTM layer is created by first combining the hidden states of both LSTMs at each time step, which is commonly accomplished by concatenating the results of the two combinations. The BiLSTM Architecture is seen up top in Figure 7, which may be found on page [27].

Here's a step-by-step explanation of the BiLSTM process:

- Input sequence: Given an input sequence (x_1, x_2, ..., x_T), the BiLSTM processes it in both forward and backward directions.
- Forward LSTM layer: The input sequence, from the first item (x_1) to the last item (x_N), is processed by the forward LSTM layer (x_T). The hidden state (hf_t) is calculated at each

time step t using the current input (x_t) and the prior hidden state (hf_{t-1}). This LSTM layer stores information about the future.

- Backward LSTM layer: The input sequence is reverse-processed by the backward LSTM layer, which moves from the final element (x_T) to the first (x_1). At each time step t, it takes the current input (x_t) and the previously hidden state (hb_{t+1}) and calculates the hidden state (hb_t). The historical context is captured by this LSTM layer.
- Combining hidden states: The output (y_t) of a BiLSTM is the result of a combination of the hidden states from the forward (hf_t) and backward (hb_t) LSTM layers at each time step t. The hidden states can be combined in a variety of ways, such as by concatenation ($y_t = [hf_t ; hb_t]$), addition ($y_t = hf_t + hb_t$), or even by running them through a feedforward neural network.
- Output: The final output of the BiLSTM is a sequence of combined hidden states (y_1, y_2, ..., y_T), which can be used as input for the next layer or for generating predictions.

BiLSTMs are useful for a variety of sequence-to-sequence applications, such as speech recognition [28], natural language processing, and machine translation. They provide a richer representation of the input sequence since they capture both forward and backward dependencies, which can lead to increased performance compared to that of unidirectional LSTMs. BiLSTMs, on the other hand, need a greater amount of computational resources and memory than ordinary LSTMs do because they comprise two LSTM layers that operate in tandem.

3.2 Transformer Models

In the publication "Attention is All You Need" (2017) [29], Vaswani et al. introduced a sort of neural network architecture known as transformer models. These models are meant to perform sequence-to-sequence tasks, such as machine translation and text summarization. The BERT model, along with many others that are considered state-of-the-art in the field of natural language processing, has their foundations in transformers.

The self-attention mechanism, which enables Transformer models to determine the relative significance of various input elements by analyzing their connections to one another, is the most important new feature introduced by these models. This method makes it possible for Transformers to efficiently capture long-range dependencies in sequences, which is something that standard RNNs and LSTMs may find difficult to do.

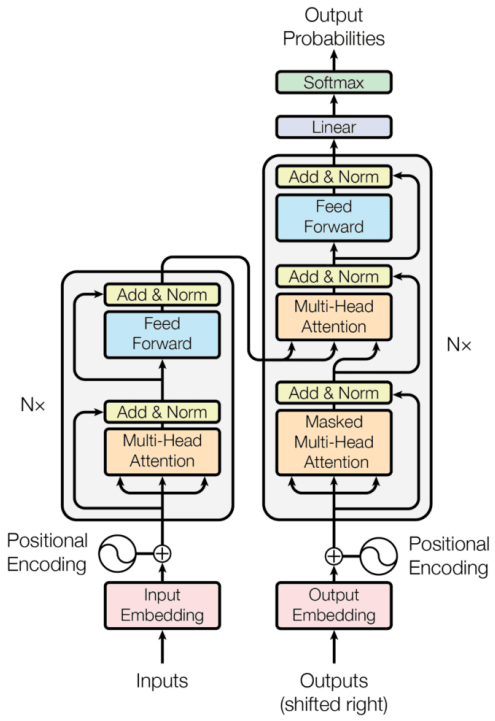

Figure 8: The Encoder-Decoder Structure of the Transformer Architecture

As can be seen in Figure 8 [30], a Transformer model is made up of two primary components, which are referred to as the encoder and the decoder. Each component is made up of a stacked arrangement of layers that are all the same. Let's study each part in detail:

- Encoder: Encoders take a sequence as input and provide a continuous representation for each element. It has multiple layers, each of which houses a position-wise feedforward neural network and a multi-head self-attention mechanism. Layer normalization and residual connections link these layers together.
    - Multi-head self-attention: This technique calculates attention scores for each pair of components in the input sequence, where a higher score indicates more weight should be given to the first element in the pair when representing the second. To generate an output sequence of the same length as the input, we use the attention scores to compute a weighted sum of the input items. The model can simultaneously learn several attention patterns thanks to the multi-head mechanism, thus capturing a wider range of interactions between items.
    - Position-wise feedforward neural network: This section applies a feedforward neural network to each individual sequence element, enabling the model to discover intricate nonlinear connections between inputs.
- Decoder: The output sequence is created by the decoder from the continuous representation produced by the encoder. Like the encoder, it is multi-layered and has position-wise feedforward neural networks, multi-head self-attention mechanisms, and encoder-decoder attention mechanisms at each level.

- Multi-head self-attention: This method is analogous to that of the encoder, except that it modifies the output sequence already generated by the decoder. Figure 9 below shows the multi-head attention module in the decoder [30]

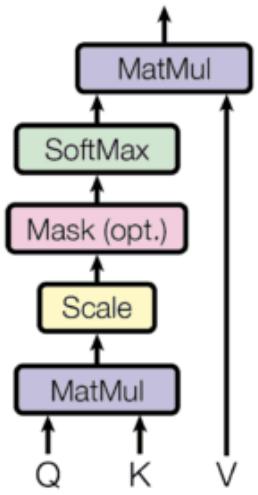

Figure 9: The Multi-Head Attention in the Decoder

- Encoder-decoder attention: By calculating attention scores between the decoder's output sequence and the encoder's continuous representation, this method establishes a bridge between the two components. Because of this, the decoder is only going to generate output for the important bits of the input sequence.
- Position-wise feedforward neural network: In the same vein as the encoder, this part individually applies a feedforward neural network to each output sequence piece.

Positional encoding is another component of the Transformer design. This feature injects information into the model regarding the order in which the pieces of the sequence are

presented. This is extremely important because the self-attention mechanism is permutation invariant and does not contain any inherent sense of position.

In general, Transformer models have enjoyed a great deal of success in a variety of sequence-to-sequence tasks [31]. This is largely attributable to their capacity to effectively capture long-range dependencies, their high parallelizability during training, and their scalability to large amounts of data. These have become the basis for numerous models in natural language processing that are considered to be state-of-the-art, such as BERT, GPT [32], and T5 [33].

3.2.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained deep learning system for interpreting natural language. It was introduced in the work that Devlin and colleagues published in 2018 [34]. The full meaning of the acronym BERT is "Bidirectional Encoder Representations from Transformers." This study's full name is "Pre-training of Deep Bidirectional Transformers for Language Understanding," and it is titled after that phrase. The efficiency of BERT is at the forefront of contemporary research and progress in a variety of NLP applications [35]. These applications include named entity recognition, sentiment analysis, and question answering.

The Transformer design, on which BERT is based, processes input sequences via self-attention techniques rather than recurrent or convolutional layers. These layers are typically used in traditional neural network architectures. The most important contribution that BERT made was the development of its bidirectional training method. This method enables the model to acquire context from both the right and left sides of a token in a particular sequence.

Because of this bidirectional context, BERT can better understand the meaning of words within their context, which ultimately leads to improvements in its performance on NLP tasks [36]. Two stages are included in the BERT model's training process:

- Pre-training: During this unsupervised phase of the training process, BERT is trained on a large corpus of text (Wikipedia or the BooksCorpus) through the use of two tasks:
    - Masked Language Modeling (MLM): BERT is taught to use the context of surrounding tokens for predicting randomly masked tokens in a sequence. Because of this, the model is pushed to acquire linguistic context in both directions.
    - Next Sentence Prediction (NSP): BERT may also determine if two sentences are sequentially related to one another in the source text. With this information, the model can better understand sentence structure and discover new correlations between sentences.
- Fine-tuning: After the initial phase of "pre-training," in which a large unlabeled dataset is used, BERT is fine-tuned on a targeted supervised NLP job. As part of the fine-tuning process, the model is modified to include new layers tailored to the target task and its weights are adjusted with data collected for that specific endeavor. This kind of tuning helps BERT swiftly adjust to new NLP tasks and attain optimal performance.

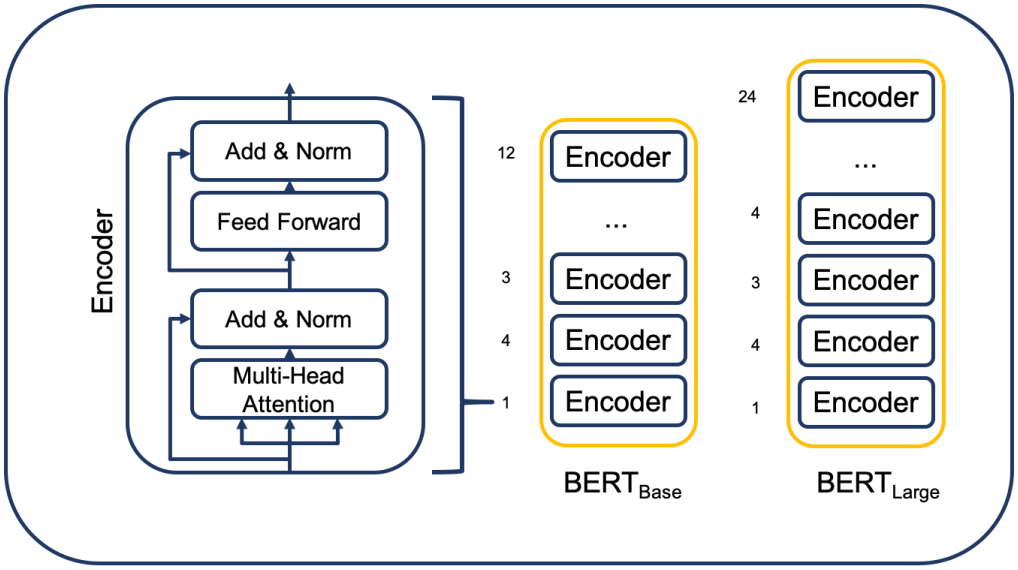

Figure 10: BERT Architecture

BERT as seen in Figure 10 [37], may be implemented in many distinct ways, each of which varies the model size as well as the quantity of training data. BERT-base (which has 12 layers, 768 hidden units, and 12 attention heads) and BERT-large (which has 24 layers, 1,024 hidden units, and 16 attention heads) are two of the most frequent variations of the BERT algorithm. In general, larger models produce better results, but they demand more computer resources for training and inference.

In a nutshell, BERT is a robust pre-trained model for natural language comprehension that is founded on the Transformer architecture. Its state-of-the-art performance may be achieved on a broad variety of NLP tasks thanks to its bidirectional training strategy and two-step training procedure (pre-training and fine-tuning). Since its inception, the BERT model has been the impetus for a plethora of iterations and expansions, such as RoBERTa, DistilBERT, and ALBERT, which have contributed to the overall development of the NLP discipline.

3.2.2 RoBERTa

A new variation of the BERT model was presented by Liu et al. in their 2019 work [38], under the acronym RoBERTa (which stands for "Robustly optimized BERT method"). RoBERTa is an approach to BERT pretraining that is robustly optimized. The purpose of RoBERTa is to increase the performance of BERT by improving the pre-training procedure, employing larger training datasets, and optimizing hyperparameters. When compared to the original BERT model, RoBERTa features some key distinctions as well as enhancements.

- Dynamic Masking: IDuring the pre-training phase of the masked language modeling (MLM) task that BERT utilizes, a predetermined proportion of the tokens in the input sequence are switched out for [MASK] tokens. This masking is static, meaning that the tokens it covers up don't change across different rounds of training. For better training results, RoBERTa introduces dynamic masking, where the tokens to be hidden vary throughout iterations of the training process. By masking new tokens at each training iteration, the model is nudged toward learning more abstract representations of the input text rather than memorizing precise token placements.
- Byte Pair Encoding: RoBERTa departs from BERT's WordPiece tokenization by instead using byte pair encoding (BPE). It is possible to effectively express an open vocabulary using BPE, a subword tokenization method. RoBERTa is better equipped than BERT to deal with uncommon and out-of-vocabulary terms thanks to this strategy.
- Removal of Next Sentence Prediction (NSP): To determine if two sentences followed one another in the original text, BERT underwent a next sentence prediction challenge during its pre-training phase. The goal of this exercise was to improve BERT's grasp of

sentence structure by exposing it to and learning from connections between sentences. RoBERTa's creators, however, discovered that greater results might be achieved on subsequent tasks by omitting this step from the pre-training phase and instead relying solely on the masked language modeling target. This result hints that the NSP task might not be as useful as originally thought for acquiring representations of context.

- Larger Training Dataset: The BooksCorpus (used in BERT) and the English version of the WebText dataset are both included in RoBERTa's pre-training dataset. With roughly 16GB of text, the merged dataset represents a more robust language-learning resource for RoBERTa. Increased natural language processing tests' performance is attributed to the larger dataset's ability to capture more complex linguistic patterns and correlations.
- Longer Training Time: In comparison to the original BERT model, the creators of RoBERTa spent more time training their model and performed more training iterations. Training RoBERTa for longer allows it to pick up on more subtle linguistic nuances, which in turn boosts its efficiency on downstream tasks.
- Larger Batch Sizes: To boost model stability and convergence, RoBERTa pre-trains using bigger batch sizes. For training, higher batch sizes assist reduce noise in the gradient estimates, resulting in more precise updates.
- Optimized Hyperparameters: To identify the best possible setup for the model, the RoBERTa authors experimented with a wide range of hyperparameter variables, including learning rates, training schedules, and the number of training iterations. By fine-tuning these hyperparameters, the authors improved the model's performance relative to the baseline BERT model on downstream tasks.

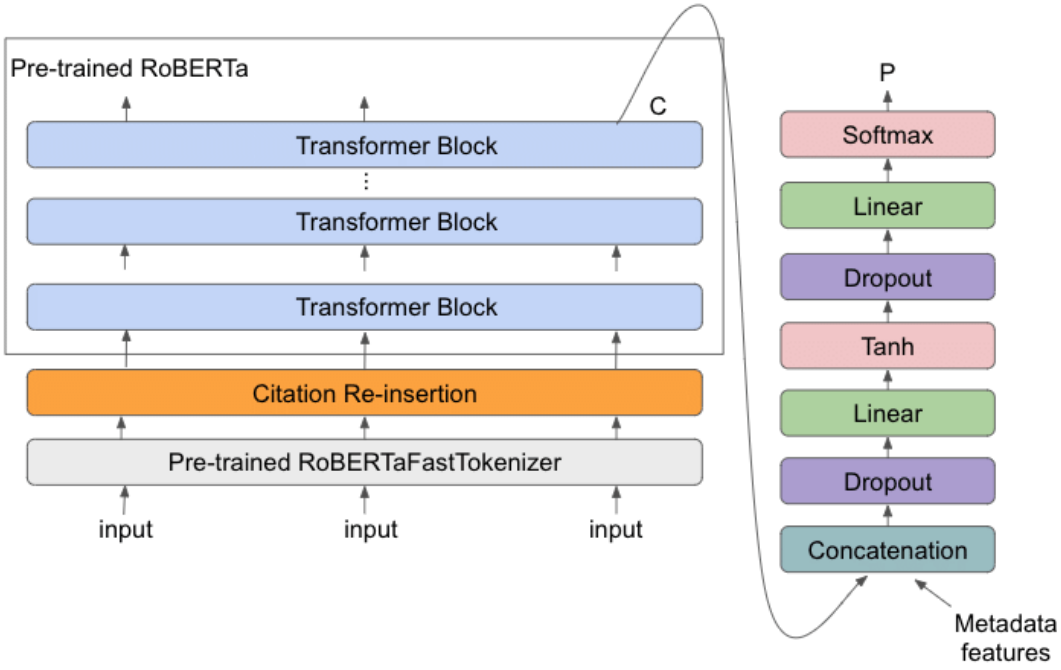

Figure 11: RoBERTa Architecture

The above figure depicts RoBERTa architecture [39]. The significance of refining pre-training strategies and maximizing the effectiveness of the training setup is highlighted by the advancements that RoBERTa makes over the basic BERT model. Its state-of-the-art performance on various benchmark datasets, such as GLUE, SuperGLUE [40], and SQuAD, has made RoBERTa a popular choice for transfer learning in NLP tasks and spurred further research into improving pre-training methods for Transformer-based models. GLUE, SuperGLUE [40], and SQuAD are just a few examples of these benchmark datasets.

3.2.3 DistilBERT

In their 2019 work titled "DistilBERT, distilled form of BERT: smaller, faster, cheaper, and lighter," [41] Sanh et al. introduced the "DistilBERT" model, which is a variant of the BERT model that is more compact, quicker, and lighter. DistilBERT's primary objective is to preserve a significant portion of BERT's performance while simultaneously shrinking its footprint and

lowering the amount of processing it calls for. This will make it more suitable for use in contexts with limited resources as well as on-edge devices.

Knowledge distillation [42] is the procedure that is utilized to construct DistilBERT. During this process, a larger, more sophisticated model (referred to as the "teacher" model in this scenario, which is BERT) is used to direct the training of a smaller, simpler model (the "student" model, which is DistilBERT). This representation can be seen in Figure 12 [43] below. The primary objective of the knowledge distillation process is to effectively transfer knowledge from the instructor model to the student model with as little drop-off in performance as possible.

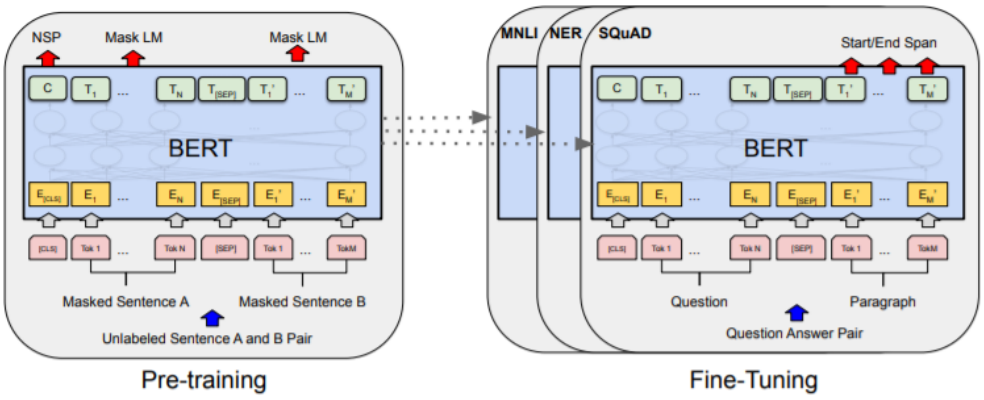

Figure 12: DistilBert Architecture in Student Model

Here is a more detailed explanation of the key aspects of DistilBERT and the knowledge distillation process:

- Architecture: DistilBERT shares the same basic structure as BERT but is significantly smaller. DistilBERT lacks token-type embeddings and a next-sentence prediction job in its pre-training, and it has fewer layers (6 for DistilBERT-base as opposed to BERT-12). base's As a result, the resulting model has about 40% fewer parameters and is roughly half the size of BERT.

- Knowledge Distillation: While its architecture is comparable to BERT, DistilBERT is significantly smaller. During pre-training, DistilBERT does not use token-type embeddings or a next-sentence prediction task, and it has fewer layers (6 for DistilBERT-base versus 12 for BERT-base). The resulting model is around 40% smaller in size than BERT's and requires half as many parameters.
- Soft Target Probabilities: Knowledge distillation entails adjusting the student model so that its predictions are consistent with the instructor model's soft target probability. To get "softer" probability distributions that capture more information about the relationships between classes, the instructor model's output logits are temperature-scaled. The pupil model learns to better imitate the teacher model by matching its behavior on these softer probabilities.
- Distillation Loss: Distillation loss functions are used to quantify the degree to which student model output probabilities deviate from those predicted by the instructor model. A common method for determining distillation loss is the Kullback-Leibler (KL) divergence between the probability distributions of the instructor and the trainee. The probability of the instructor model's output is approximated by the student model by minimizing this loss function.
- Training Process: The distillation loss and the regular masked language modeling (MLM) loss are used in DistilBERT's pre-training. The distillation loss promotes the student model's imitation of the teacher model, while the MLM loss facilitates the student model's acquisition of knowledge from the unprocessed training data. In the end, the training goal is a combination of the two loss functions.

- Performance: DistilBERT is far more compact than BERT while retaining much of its performance on many natural language processing applications. DistilBERT can attain almost 97% of BERT's performance on the GLUE benchmark with only 60% of its parameters. DistilBERT is a viable option for use cases where computing resources or model size are limited due to the aforementioned performance trade-off.

In conclusion, DistilBERT is a more compact and efficient variant of BERT that makes use of knowledge distillation to keep performance at a high level despite a drastic reduction in data and processing needs. Given the importance of speed in real-time applications and the limited resources available on mobile devices, DistilBERT is a promising choice for delivering BERT-based models.

3.2.4 ALBERT

ALBERT, or "A Lite BERT," is a variant of the BERT model that tries to scale down the model and lower the processing demands while preserving efficiency. In their 2019 publication titled "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations" [45], Lan et al. proposed it. The basic BERT design is altered in a variety of ways by ALBERT, which helps to lower the total number of variables and lower memory and processing demands. Figure 13 below illustrates the innovative architecture of ALBERT [46]

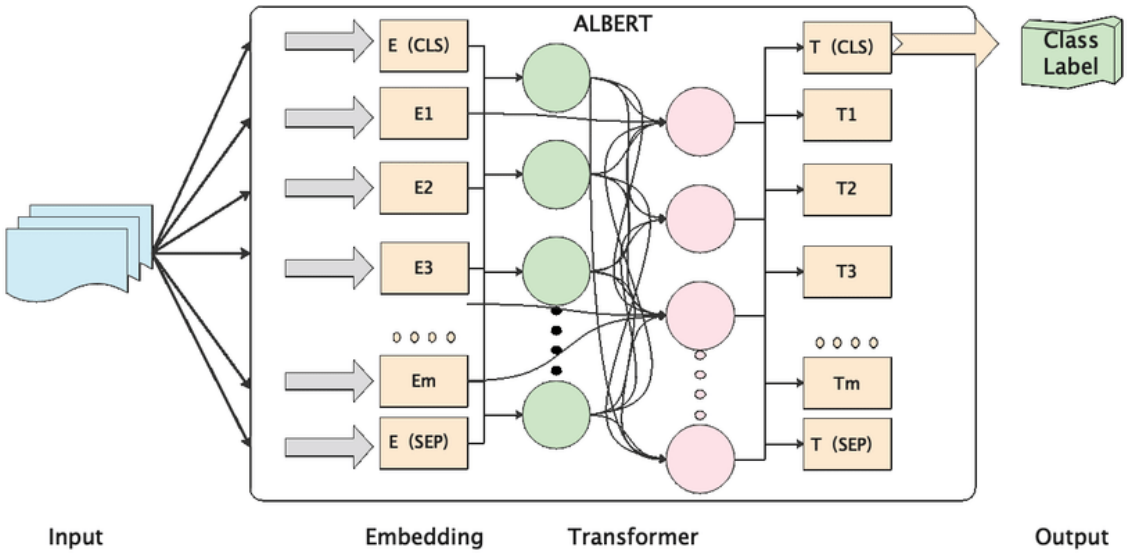

Figure 13: ALBERT Architecture

The key innovations of ALBERT are as follows:

- Factorized Embedding Parameterization: One of the main contributions of ALBERT is the factorization of the embedding layer into two smaller matrices. In the original BERT model, the embedding matrix is of size V x H, where V is the vocabulary size and H is the hidden size. In ALBERT, the matrix is factorized into two smaller matrices of size V x E and E x H, where E is a smaller embedding size. By doing this, the embedding layer's component count is decreased without affecting the model's ability to record more data. The model's memory consumption and processing needs are reduced thanks to the adjusted embedding parameterization.
- Cross-layer Parameter Sharing: Another innovation in ALBERT is the sharing of parameters across the layers of the model. In the original BERT model, each layer has its own set of parameters. The parameters, however, can be used by every level in ALBERT. The model becomes simpler and has fewer parameters, which speeds up training and

facilitates deployment. While training on smaller datasets, cross-layer pooling of parameters significantly assists minimize the danger of overfitting.

- Sentence Order Prediction (SOP) Task: Sentence Order Prediction (SOP), a novel pre-training assignment that ALBERT proposes as an alternative to the Next Sentence Prediction (NSP) task employed in the initial BERT model. To determine if two input phrases are in the original sequence or have been switched around, the model undergoes training for the SOP problem. It has been demonstrated that this new task, which is aimed at inspiring the model to learn better inter-sentence coherence, is more efficient for downstream tasks than the NSP task.
- Model Variants: ALBERT offers several model variants with different hidden layer sizes, numbers of attention heads, and numbers of layers. This allows users to choose the model that best fits their specific requirements in terms of memory, computational resources, and performance. The base ALBERT model (ALBERT-base) has 12 layers and 768 hidden units, while larger models like ALBERT-large, ALBERT-xlarge, and ALBERT-xxlarge have more layers and hidden units.
- Performance: Despite its reduced size and computational requirements, ALBERT demonstrates competitive performance on many NLP tasks. On the GLUE benchmark, ALBERT achieves marvelous results, outperforming BERT and other Transformer-based models. ALBERT is also effective on the SQuAD and RACE benchmarks, which involve question-answering and reading comprehension tasks, respectively.

In summary, ALBERT is a more efficient and lighter version of BERT that retains much of its performance while reducing the model size and computational requirements. This is accomplished by introducing the Sentence Order Prediction challenge, factorized embedding parameterization, and cross-layer sharing of variables. ALBERT could be used in resource-constrained contexts where the original BERT model may be overly complex or intensive to compute. This is appropriate for a variety of NLP jobs.

CHAPTER 4

Dataset and Experiments

For the experiments in this project, the dataset referred to as the Balabit Mouse Dynamics Challenge [7] was utilized. A cybersecurity business called Balabit (which is now a part of One Identity) made this dataset available to the public and made it is available on their website. In 2016, the dataset was made available for download in preparation for a competition that was organized on Kaggle. The objective of the competition was to construct models that can identify people based on the patterns of mouse movement that those users exhibit. These models may subsequently be used as a way of continuous authentication or behavioral biometrics.

The collection is made up of aggregated and anonymized information regarding mouse movements from 10 different users. The data collection took place for two months, during which time the participants carried out their normal day-to-day work activities. The dataset contains a total of 5,500 sessions, with an average of around 55 sessions per user every day.

The data provided is a collection of CSV files, where each file represents a single session for a user. The columns in each CSV file include the following information:

- record timestamp (time): The timestamp of the recorded mouse event.
- client timestamp (client timestamp): The client-side timestamp of the recorded mouse event.
- button (button): The state of the mouse button during the event (e.g., 'Left', 'Right', 'None').
- state (state): The state of the mouse event (e.g., 'Pressed', 'Released', 'Move').

- x coordinate (x): The x-coordinate of the mouse pointer on the screen.
- y coordinate (y): The y-coordinate of the mouse pointer on the screen.

The goal of the competition was to use this data to train models that can distinguish between the 10 different users based on their mouse movement patterns. Participants in the competition used various machine learning techniques, including deep learning, feature engineering, and ensemble methods, to develop their models.

Though the Balabit Mouse Dynamics Challenge has concluded, the dataset is still available and can be used for research purposes or to develop novel techniques for user authentication based on mouse dynamics.

4.1 Experiments

This particular sub-section gives a detailed description of the experiments conducted on the dataset and their results. The first set of experiments deals with using only the RNN models, followed by experiments on hybrid models created by using the embeddings generated by transformer models and RNN models.

4.1.1 RNN model experiments

Using the implementation of LSTM, GRU, and BiLSTM models described [9], the same models were recreated and then tested to see the accuracies of the predictions, and the layer information was given the same as the previous work. The experiments resulted in fetching similar maximum accuracies. Figure 14 below shows the accuracies of the three models in the form bar graph.

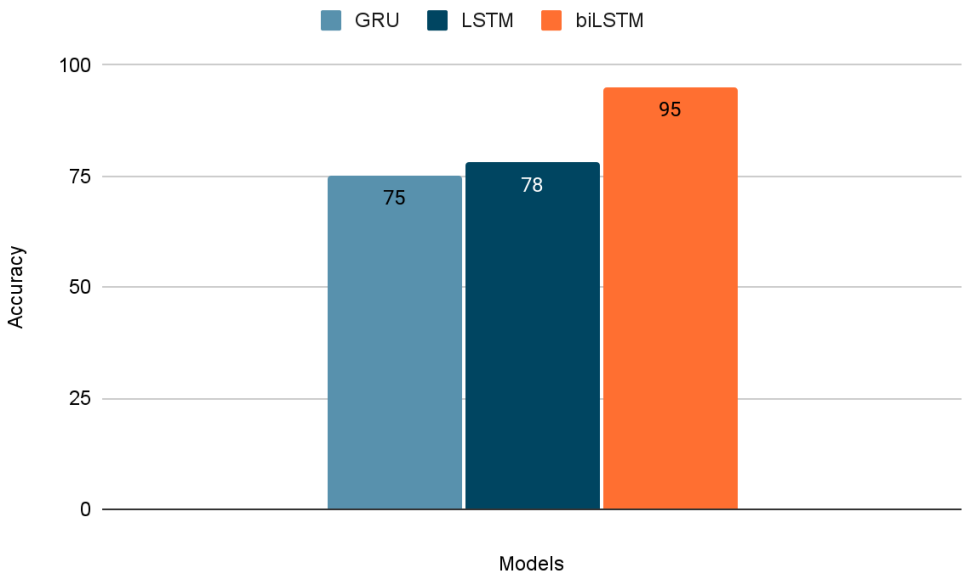

Figure 14: Maximum Accuracy of RNN models without embeddings

4.1.2 RoBERTa-based Experiments

This section shows the experiments done with a custom-made deep learning model. The preprocessed data is tokenized using the RobertaTokenizer. The RobertaTokenizer is initialized using the 'roberta-large' model described in section 3.3.3(change later). The batch_encode_plus method returns the embeddings in the form of TensorFlow tensors, which are then split into training and testing data. The pre-trained RoBERTa model is loaded using the 'TFRobertaModel.from_pretrained' method and then we combine this with the RNN models GRU, LSTM, and biLSTM. The embeddings are provided to the RNN models, and the RoBERTa model is incorporated into the hybrid model as an additional layer following the input level. The models' layers and parameters are displayed in the table below.

Layer	Output Shape	Parameters
Input	[(None, 512)]	0
TFRobertaModel	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	124645632
GRU	(None, 128)	344832
Dense	(None, 1)	129

Table 1: Model Layers and Parameters for GRU with RoBERTa

Layer	Output Shape	Parameters
Input	[(None, 512)]	0
TFRobertaModel	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	124645632
LSTM	(None, 128)	459264
Dense	(None, 1)	129

Table 2: Model Layers and Parameters for LSTM with RoBERTa

Layer	Output Shape	Parameters
Input	[(None, 512)]	0
TFRobertaModel	TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	124645632
SlicingOpLambda	(None, 768)	0
Bidirectional	(None, 128)	426496
Concatenate	(None, 896)	0
Dense	(None, 128)	114816
Dropout	(None, 128)	0
Dense	(None, 1)	129

Table 3: Model Layers and Parameters for biLSTM with RoBERTa

When the hybrid models were trained and validated, it resulted in maximum accuracies as depicted below.

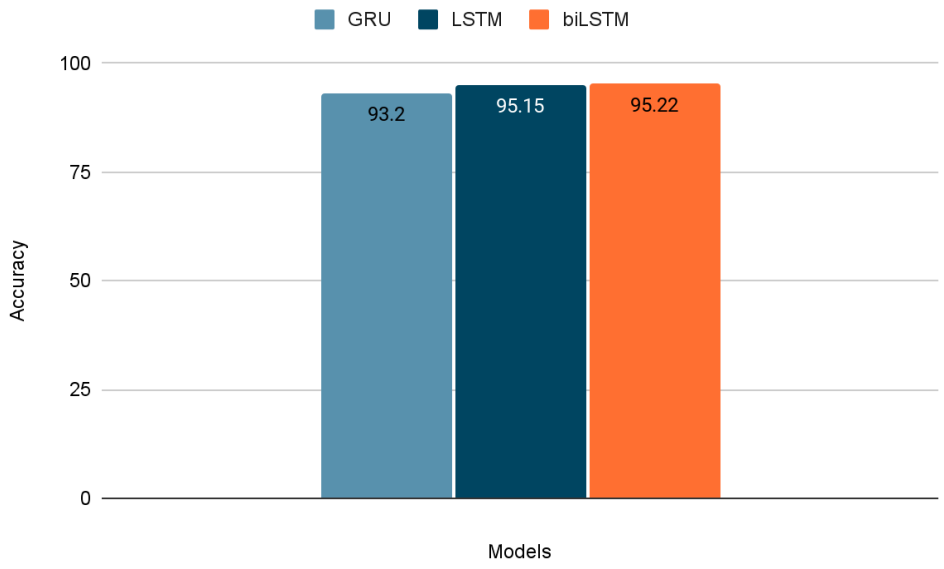

Figure 15: Maximum Accuracy of RNN Models with RoBERTa Embeddings

4.1.3 DistilBERT-based Experiments

Here, the experiments that deal with distilBERT embeddings in combination with RNN models are discussed. The preprocessed data is taken and then tokenized using the DistilBertTokenizer which is initialized using the 'distilbert-base-uncased' model. The batch_encode_plus method of the tokenizer processes the data and returns the embeddings. The embeddings were fed as input layers to the RNN models and the pre-trained distilBERT model is also placed as an intermediate layer in the deep learning models. The tables below describe the layers and the parameters of these models.

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFDistilBertModel	TFBaseModelOutput(last_hidden_state=(None, 512, 768), hidden_states=None, attentions=None)	66362880
GRU	(None, 32)	76992
Dense	(None, 1)	33

Table 4: Model Layers and Parameters for GRU with DistilBERT

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFDistilBertModel	TFBaseModelOutput(last_hidden_state=(None, 512, 768), hidden_states=None, attentions=None)	66362880
LSTM	(None, 128)	459264
Dense	(None, 1)	129

Table 5: Model Layers and Parameters for LSTM with DistilBERT

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFDistilBertModel	TFBaseModelOutput(last_hidden_state=(None, 512, 768), hidden_states=None, attentions=None)	66362880
SlicingOpLambda	(None, 768)	0
Bidirectional	(None, 128)	426496
Concatenate	(None, 896)	0
Dense	(None, 128)	114816
Dropout	(None, 128)	0
Dense	(None, 1)	129

Table 6: Model Layers and Parameters for biLSTM with DistilBERT

Upon validating the models, the maximum accuracies of the models were observed to be

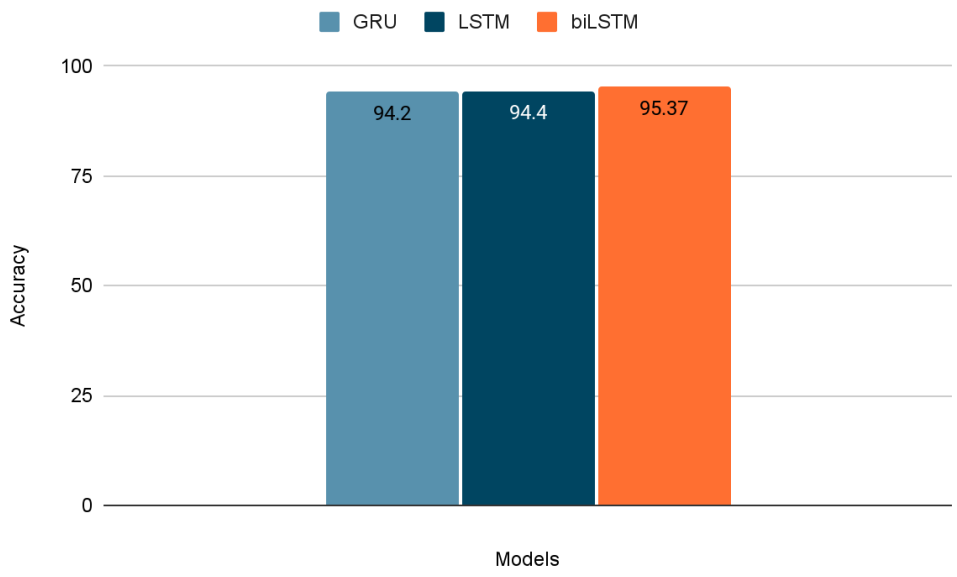

Figure 16: Maximum Accuracy of RNN Models with DistilBERT Embeddings

4.1.4 ALBERT-based Experiments

This section deals with the experiments carried out using ALBERT embeddings in combination with RNN models. Similar to the previous experiments, the preprocessed data is taken and then tokenized using the AutoTokenizer which is initialized using the 'albert-base-v2' model. The embeddings from the tokenizer were fed as input layers to the RNN models and the pre-trained ALBERT model is also placed as an intermediate layer in the deep learning models. The tables below describe the layers and the parameters of these models.

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFAlbertModel	TFBaseModelOutputWithPooling(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), hidden_states=None, attentions=None)	11683584
GRU	(None, 32)	76992
Dense	(None, 1)	33

Table 7: Model Layers and Parameters for GRU with ALBERT

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFAlbertModel	TFBaseModelOutputWithPooling(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), hidden_states=None, attentions=None)	11683584
LSTM	(None, 128)	459264
Dense	(None, 1)	129

Table 8: Model Layers and Parameters for LSTM with ALBERT

Layer	Output Shape	Parameters
mask_layer (Input)	[(None, 512)]	0
input_layer (Input)	[(None, 512)]	0
TFAlbertModel	TFBaseModelOutputWithPooling(last_hidden_state=(None, 512, 768), pooler_output=(None, 768), hidden_states=None, attentions=None)	11683584
SlicingOpLambda	(None, 768)	0
Bidirectional	(None, 128)	426496
Concatenate	(None, 896)	0
Dense	(None, 128)	114816
Dropout	(None, 128)	0
Dense	(None, 1)	129

Table 9: Model Layers and Parameters for BiLSTM with ALBERT

After validating the models, the maximum accuracies were observed to be

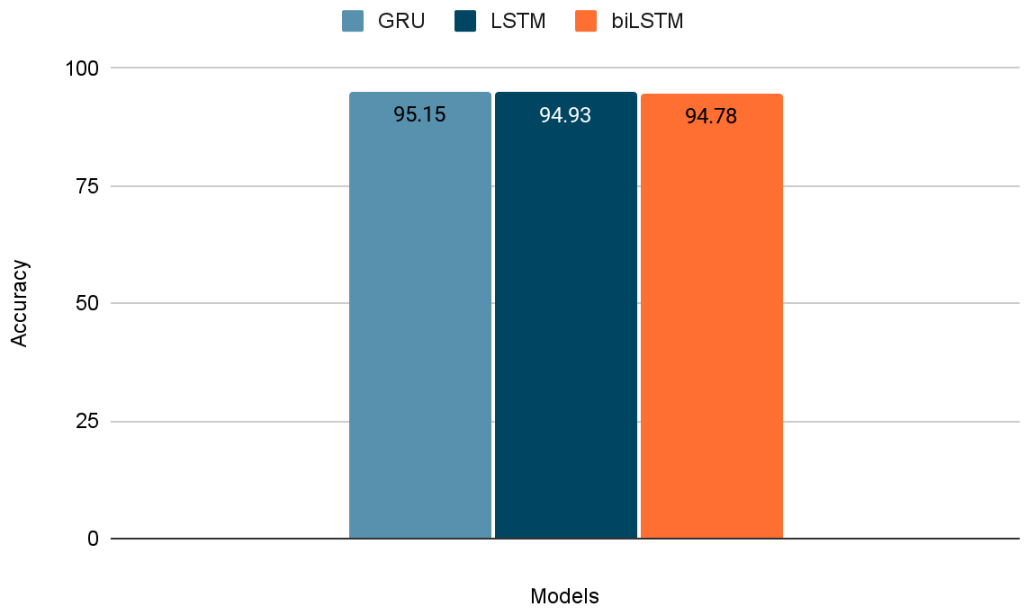

Figure 17: Maximum Accuracy of RNN Models with ALBERT Embeddings

CHAPTER 5

Conclusion and Future Work

In conclusion, this paper revealed that it is effective to use embeddings derived from pre-trained transformer models in conjunction with RNN models for intrusion detection based on Mouse Dynamics. Our strategy makes use of the sophisticated language understanding capabilities of transformer models, which capture contextualized representations of the input data. At the same time, the RNN layers model the temporal relationships in the sequences of mouse movements.

The outcomes of the study confirmed that the recommended models exhibited promising performance in identifying intrusions. They outperformed baseline models and previously described methods in the literature to achieve this goal. It was only possible for the model to capture high-level and fine-grained patterns in mouse movements thanks to the combination of transformers and RNNs, which is crucial for accurate intrusion detection.

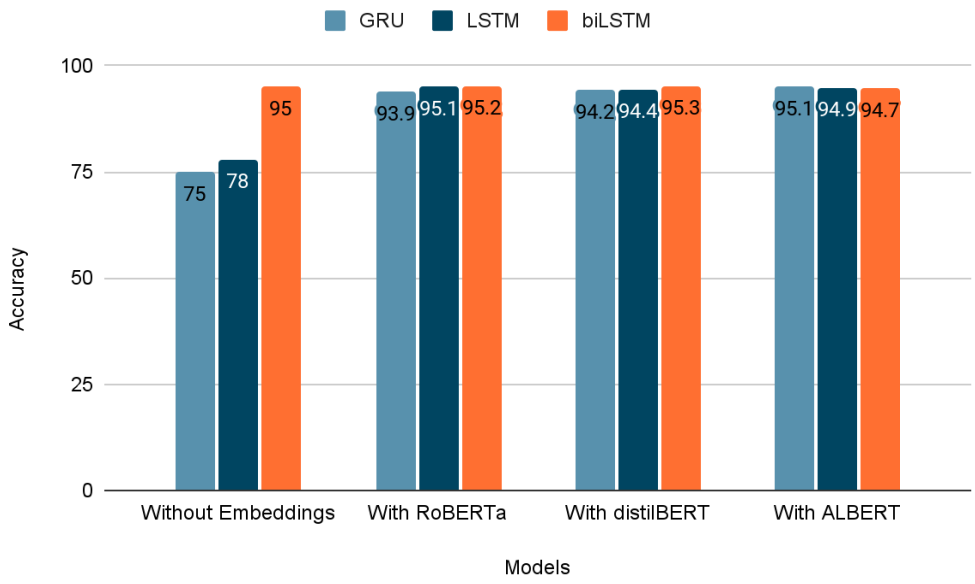

Figure 18: Maximum Accuracy of the Experiments for Comparison

From the column chart above we can observe that the usage of embeddings is making the GRU and LSTM more stable and closer in accuracy to biLSTM. The maximum accuracy (95.1 %) for GRU is observed with ALBERT, whereas for LSTM (95.1) with RoBERTa and biLSTM (95.3%) with distilBERT embeddings. The best accuracy for the combination model is achieved when distilBERT embeddings are used with the biLSTM model. It can be concluded that the embeddings are helping the GRU and LSTM but BiLSTM remains the best choice with or without the embeddings.

Future studies could go into several different areas. First, exploring other transformer models, such as BERT or GPT, and other models like CNN, could provide additional insights into the best model architecture for this task. Second, integrating attention mechanisms or other advanced techniques may further enhance the model's ability to focus on important features in the Mouse Dynamics data. Finally, the proposed approach could be implemented in other types of data and domains, such as keystroke dynamics, network traffic analysis, or user behavior analysis, to assess its generalizability and potential for broader application in intrusion detection.

By successfully demonstrating the utility of combining transformer-based embeddings with RNN models for detecting intrusions based on Mouse Dynamics, this paper contributes to advancing deep learning techniques for cybersecurity. It functions as an entry point for any additional research in this field of study.

REFERENCES

[1] A. Adams and M. A. Sasse, “Users are not the enemy,” *Commun. ACM*, vol. 42, no. 12, pp. 40–46, Dec. 1999.

[2] M. A. Sasse, S. Brostoff, and D. Weirich, “Transforming the ‘weakest link’—a human/computer interaction approach to usable and effective security,” *BT Technol. J.*, vol. 19, no. 3, pp. 122–131, 2001.

[3] A. K. Jain, A. Ross, and S. Prabhakar, “An introduction to biometric recognition,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 1, pp. 4–20, Jan. 2004.

[4] H. Gamboa and A. Fred, “A behavioral biometric system based on human-computer interaction,” *Biometric Technology for Human*, 2004, [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/5404/0000/A-behavioral-biometric-system-based-on-human-computer-interaction/10.1117/12.542625.short

[5] Z. Jorgensen and T. Yu, “On mouse dynamics as a behavioral biometric for authentication,” in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, Hong Kong, China, Mar. 2011, pp. 476–482.

[6] X.-J. Chen, F. Xu, R. Xu, S. M. Yiu, and J.-Q. Shi, “A practical real-time authentication system with Identity Tracking based on mouse dynamics,” in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 121–122.

[7] Á. Fülöp, L. Kovács, T. Kurics, and E. Windhager-Pokol, “Balabit mouse dynamics challenge data set,” *Accessed on May*.

[8] T. Hu, W. Niu, X. Zhang, X. Liu, J. Lu, and Y. Liu, "An Insider Threat Detection Approach Based on Mouse Dynamics and Deep Learning," *Security and Communication Networks*, vol. 2019, Feb. 2019, doi: 10.1155/2019/3898951.

[9] H. A. Lee, N. Prathapani, R. Paturi, and S. Parmaksiz, "NLP-based User Authentication through Mouse Dynamics," *ICISSP*.

[10] V. Pandya, "Contextualized Vector Embeddings for Malware Detection," San Jose State University, 2022. doi: 10.31979/etd.rjra-9c8m.

[11] https://www.ibm.com/topics/recurrent-neural-networks

[12] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and Understanding Recurrent Networks," *arXiv [cs.LG]*, Jun. 05, 2015. [Online]. Available: http://arxiv.org/abs/1506.02078.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[16] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," *arXiv [cs.LG]*, May 29, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, Accessed: Apr. 15, 2023. [Online].

Available:

https://proceedings.neurips.cc/paper/5346-sequence-to-sequence-learning-with-neural-

[18] https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/

[19] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: continual prediction with LSTM,” *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000.

[20] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, “Convolutional LSTM network: A machine learning approach for precipitation nowcasting,” *arXiv [cs.CV]*, Jun. 12, 2015. Accessed: Apr. 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/07563a3fe3bbe7e3ba84431ad9d055af-Abstract.html

[21] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” *arXiv [cs.CL]*, Jun. 03, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *arXiv [cs.NE]*, Dec. 11, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[23] https://www.nomidl.com/deep-learning/what-is-the-difference-between-lstm-and-gru/

[24] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An Empirical Exploration of Recurrent Network Architectures,” in *Proceedings of the 32nd International Conference on Machine Learning*, 07--09 Jul 2015, vol. 37, pp. 2342–2350.

[25] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Montreal, QC, Canada, 2005, pp. 2047-2052 vol. 4, doi: 10.1109/IJCNN.2005.1556215.

[26] J. Zhou and W. Xu, "End-to-end learning of semantic role labeling using recurrent neural networks," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Jul. 2015, pp. 1127–1137.

[27] https://paperswithcode.com/method/bilstm

[28] B. Plank, A. Søgaard, and Y. Goldberg, "Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss," *arXiv [cs.CL]*, Apr. 19, 2016. [Online]. Available: http://arxiv.org/abs/1604.05529

[29] A. Vaswani *et al.*, "Attention is all you need," *arXiv [cs.CL]*, Jun. 12, 2017. Accessed: Apr. 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/7181-attention-is-all

[30] https://machinelearningmastery.com/the-transformer-model/

[31] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive Attention Span in Transformers," *arXiv [cs.LG]*, May 19, 2019. [Online]. Available: http://arxiv.org/abs/1905.07799

[32] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training." https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf (accessed Apr. 15, 2023).

[33] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 5485–5551, Jan. 2020.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv [cs.CL]*, Oct. 11, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[35] C. Sun, L. Huang, and X. Qiu, “Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence,” *arXiv [cs.CL]*, Mar. 22, 2019. [Online]. Available: http://arxiv.org/abs/1903.09588

[36] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” *arXiv [cs.CL]*, Jun. 19, 2019. Accessed: Apr. 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html

[37]https://www.analyticsvidhya.com/blog/2021/12/fine-tune-bert-model-for-sentiment-analysis-in-google-colab/

[38] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *arXiv [cs.CL]*, Jul. 26, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[39] https://www.researchgate.net/figure/The-RoBERTa-model-architecture_fig2_352642553

[40] A. Wang *et al.*, “SuperGLUE: A stickier benchmark for general-purpose language understanding systems,” *arXiv [cs.CL]*, May 01, 2019. Accessed: Apr. 15, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/4496bf24afe7fab6f046bf4923da8de6-Abstract.html

[41] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv [cs.CL]*, Oct. 02, 2019. [Online]. Available: http://arxiv.org/abs/1910.01108

[42] S.-C. Lin, J.-H. Yang, and J. Lin, “Distilling Dense Representations for Ranking using Tightly-Coupled Teachers,” *arXiv [cs.IR]*, Oct. 22, 2020. [Online]. Available: http://arxiv.org/abs/2010.11386

[43] https://www.analyticsvidhya.com/blog/2022/11/introduction-to-distilbert-in-student-model/

[44] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,” *arXiv [cs.CL]*, Sep. 26, 2019. [Online]. Available: http://arxiv.org/abs/1909.11942

[45] https://www.researchgate.net/figure/The-architecture-of-the-ALBERT-model-in-our-task_fig1_355434928