Wrocław University of Science and Technology
Faculty of Fundamental Problems of Technology

Field of study: **Big Data Analytics**

MASTER THESIS

GAN-based methods of user authentication

Ivan Shamilov

Supervisor
Dr. Inż. Piotr Syga

Keywords: Biometrics, User Authentication, Keystroke Dynamics, GAN

WROCŁAW 2023

ABSTRACT

This paper aims to develop a generative deep learning approach for continuous authentication based on keystroking dynamics. The work incorporates Conditional Generative Adversarial Network (cGAN) to generate synthetic keystroke data of a specified sentence for a specific user, enabling a comparative analysis with real-time collected keystrokes. The paper outlines the details of the implementation of cGAN and the decision-making network used for comparing synthesized and real data. The results of the conducted tests show that the trained cGAN model is able to replicate the user keystroke distribution accurately. Additionally, the findings highlight the ability of the trained models to capture distinct patterns inherent to individuals, enabling the decision-making network to differentiate between users effectively.

STRESZCZENIE

Niniejsza praca ma na celu opracowanie podejścia polegającego na wykorzystaniu głębokiego uczenia generatywnego dla ciągłego uwierzytelniania opartego na dynamice naciśnięć klawiszy. Praca wykorzystuje Conditional Generative Adversarial Network (cGAN) do generowania syntetycznych danych naciśnięć klawiszy wskazanego zdania dla konkretnego użytkownika, umożliwiając analizę porównawczą z naciśnięciami zebranymi w czasie rzeczywistym. Przedstawiono są szczegóły implementacji cGAN i sieci decyzyjnej wykorzystywanej do porównywania danych syntetycznych i rzeczywistych. Wyniki przeprowadzonych testów pokazują, że wytrenowany model cGAN jest w stanie dokładnie odtworzyć rozkład naciśnięć klawiszy użytkownika. Dodatkowo, podkreślają oni zdolność wytrenowanych modeli do wychwytywania odrębnych wzorców charakterystycznych do poszczególnych osób, umożliwiając sieci decyzyjnej skuteczne rozróżniania użytkowników.

TABLE OF CONTENTS

1. INTRODUCTION

In the modern world, authentication plays a crucial role in our daily lives. From accessing bank accounts, making online purchases, conducting online business, and accessing social media platforms, we rely heavily on widely used authentication protocols and methods to keep our personal information and data confidential, integral, and secure [1]. However, despite the continuous advancements and development of broadly utilised approaches, they are not entirely safe.

The main problem we confront is the fact that the methods used nowadays can be easily bypassed. For example, passwords can be cracked using brute-force approaches, snooped using network capture techniques, or simply guessed by an attacker [2]. Authentication systems that rely on physiological biometrics are also susceptible to hacking: fingerprints can be lifted from surfaces to bypass the security system [3] and face-related verifications could be tricked by using high-quality photos, video recordings, or deep fakes [4]. Although techniques like retina recognition are considered reliable, they would require advanced and expensive hardware [5], which is not worthwhile in devices used daily, such as laptops or mobile phones.

Authentication systems that are built using two-factor authentication (2FA) are considered more secure, as they utilise two methods simultaneously (for example, password and SMS or token and some biometric factor). Although the difficulty of breaking through such a system depends on a choice of factors, this approach is still not completely safe. Common methods of hacking into 2FA systems include session hijacking and social engineering [6].

One way to overcome this problem is to use continuous authentication (CA). It ensures that the person who has logged in is the same person who is accessing the account at this very moment. It is a way to verify that the individual who logged in has not left the device unattended, handed it over to someone else to access, or, as was previously stated, someone illegitimate has not bypassed login policies and gained unauthorised access.

One of the benefits of CA is that it can identify and mark suspicious, irregular, or unconventional behaviour and activities as fraudulent or identify them as security breaches. Services such as cryptocurrency exchanges have been using similar methods for some time. However, the way these are implemented might be inconvenient and not user-friendly, as they are represented as pop-ups with login prompts or requests for face recognition authentication.

To rectify this problem, behavioural biometrics can be utilized as a way of determining the legitimacy of the user. By analysing an individual's behavioural patterns, the system would be able to create a unique profile that reflects the user's typical behaviour in a virtual environment, for instance, the dynamics of the mouse or keystroking. Unlike any other verification factor that can be stolen or hijacked, behavioural biometrics would be difficult and costly to mimic or reproduce by a hacker, as it would require having additional sensors on the victim's side to accurately learn and detect them.

The aim of this paper is to propose a new approach to CA by using a combination of behavioural biometrics, specifically keystroking, and generative deep learning. Keystroking will ensure a seamless authentication process at the underlying level, that will improve the general feeling from using a system, whereas deep learning will help to avoid straightforward comparison and to detect deviations more flexibly in the provided biometrics. The proposed approach will incorporate Generative Adversarial Network (GAN) to enhance the accuracy and efficiency of the authentication process. This method will reduce the need for users to undergo authentication processes repeatedly, allow them to use the services without any interruptions and implement a robust CA mechanism that will balance security and usability.

However, it should be noted that the usage of GAN might require more powerful hardware, which could potentially limit the scalability of the approach. In addition, the inference time might be long, which may not be appropriate for real-time continuous authentication

applications. All these aspects will be described and addressed in detail in the subsequent sections of this paper.

The paper is organized as follows. In the first chapter, we describe the purpose and the scope of the work. Following, in Section 2, we will present the theory behind the keystroking biometric and Generative Adversarial Networks, as well as discuss the related work. In Sections 3 and 4, we will describe the dataset, that will be used for training the models, analyze it, showing various statistics and visualizations, and fill the research gaps. The following section will go through the details of data pre-processing, the implementation of the models used in the proposed system and the training process. Finally, in subsequent sections, we will discuss the results, make conclusions of this paper and mention limitations and further work.

2. THEORETICAL FRAMEWORK

This section aims to provide the necessary foundations of keystroking and Generative Adversarial Networks for understanding the proposed approach. It will cover the key concepts and techniques used and establish the basis for the following sections of the paper.

2.1 KEYSTROKING

Biometrics is the measurement and statistical analysis of people's unique physical and behavioural characteristic [7]. Examples of physical biometrics include fingerprints, facial recognition, and iris/retina scans. On the other side, behavioural biometrics employs patterns of behaviour to identify people, such as keystroke dynamics, mouse movements, and voice recognition. Behavioural biometrics can change over time [8], making them harder to forge than physical biometrics, which are more static and constant (unless they were not impacted by cosmetics, disguises, or surgery) [9] [10].

Keystroke biometrics, also referred to as keystroke dynamics, is a behavioural biometric modality that examines a person's typing technique. This biometric method is based on the distinctive rhythms and patterns of a person's typing behaviour, including the timing and length of keystrokes and the order in which keystrokes are made. Additionally, depending on the capabilities of the equipment used, the amount of weight placed on the keys might also be considered.

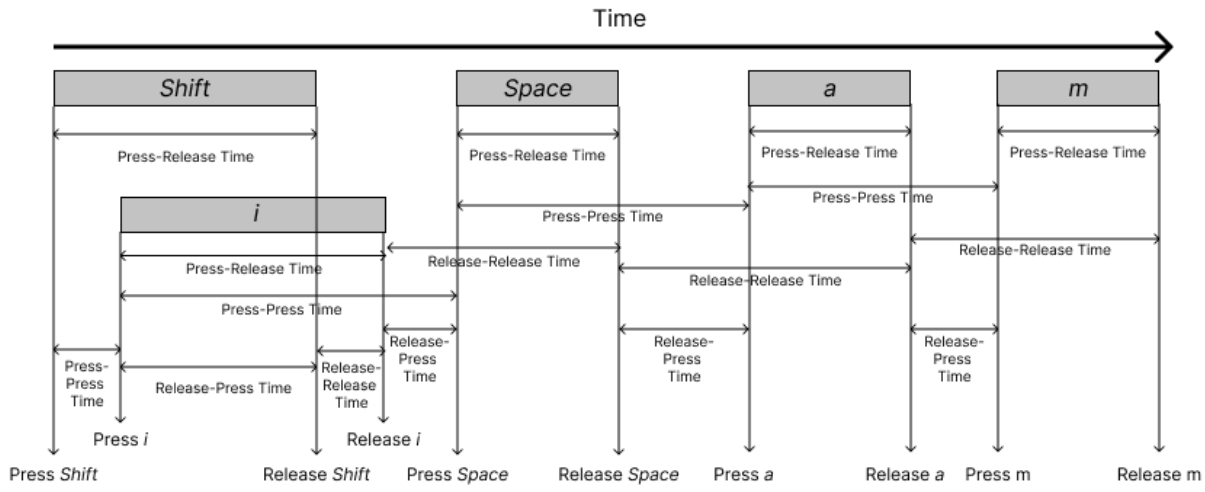

Figure 2.1 Keystroking time measured while typing "I am"

The key element in keystroking dynamics is bigram (or digram) that refers to the pair of two consecutive keystrokes made by user. Consequently, n-grams are sequences of n subsequent keys (for example, trigrams (3-grams), quadrigrams (4-grams) and so on). They are used for calculating several metrics in determining a virtual behaviour of the individual [11].

Those are:

- Interval Time (referred often as Release-Press Time or Inter-Key Interval Time) – time between releasing a key and pressing the second one.
- Dwell Time (referred often as Press-Release Time or Hold Time) – time a particular key was pressed.
- Flight Time (referred often as Press-Press Time) – time between pressing one key and pressing the second one.

- Up to Up (referred often as Release-Release Time) – time between releasing one key and releasing the second one.

Some keys are usually pressed when the previous key has not been released yet. Figure 2.1 depicts keystroking metrics that can be collected while typing the phrase “I am”. It can be observed that to get “I”, “Shift” button is clicked before the “i” key and is not released before “i” will be pressed. The term *Rollover* that describes such a behaviour was introduced in [12]. Technically speaking, we can say that rollover occurred between two keys if Release-Press time between those buttons is negative.

It is important to note that the most frequent and common errors might also be characteristic for a person. When a user makes a typing error, such as a typo or a correction, it can affect their typing behaviour and introduce additional variability into their keystroking pattern. By considering the errors in the analysis, the accuracy and robustness of the CA system can be further improved.

2.2 GENERATIVE ADVERSARIAL NETWORKS

The commonly accepted principle of an artificial neural network (ANN) is how the coefficients, or so-called neurons, are connected into a network in such a way that this model solves a specific task, such as classification. Networks consist of layers of neurons, with each neuron from the previous layer being connected to all neurons of the next layer.

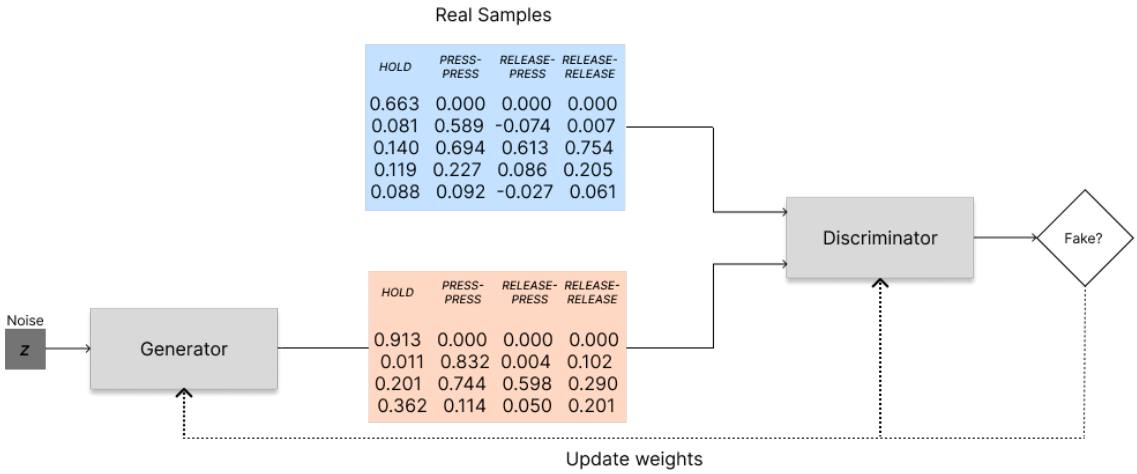

Figure 2.2 Generative Adversarial Network. Generator is responsible for creating synthetic keystrokes from a random noise, discriminator – marking those keystrokes as fakes

Generative adversarial networks (GANs) are a type of deep learning algorithm and a separate architecture of ANNs that are used for generative modelling and were firstly introduced in [13]. They consist of two distinct neural networks, called a generator *G* and discriminator *D*, that are trained in tandem in a competitive manner (Figure 2.2). The generator is responsible for generative new synthetic samples that were allegedly taken from the training data distribution, while the discriminator network would be responsible for distinguishing between the generated samples and the real training ones.

During the training process, the generator network learns how to “fool” the discriminator and tries to create more and more convincing fake samples. Conversely, the discriminator becomes better at distinguishing between real and fake instances.

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[log\, D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - log\, D(G(z))] \quad (2.1)$$

The first term of Equation 2.1 is entropy that the data that are passed through the discriminator comes from a real distribution, so it tries to maximize a factor $\log D(x)$. The second term is entropy that a sample was generated from a random noise z – discriminator tries to maximize the log of the inverse probability for fake images. Generator, on other hand, tries to minimize the function V so that the discriminator would fail in identifying generated samples as fake ones in most cases.

In this work, we will be using a type of GAN called conditional GANs (cGANs), that were introduced in [14]. Conditional GANs generate data based not only on random noise, but also on specific input conditions, including labels, images, or other forms of structured data that guide the generation process (Figure 2.3).

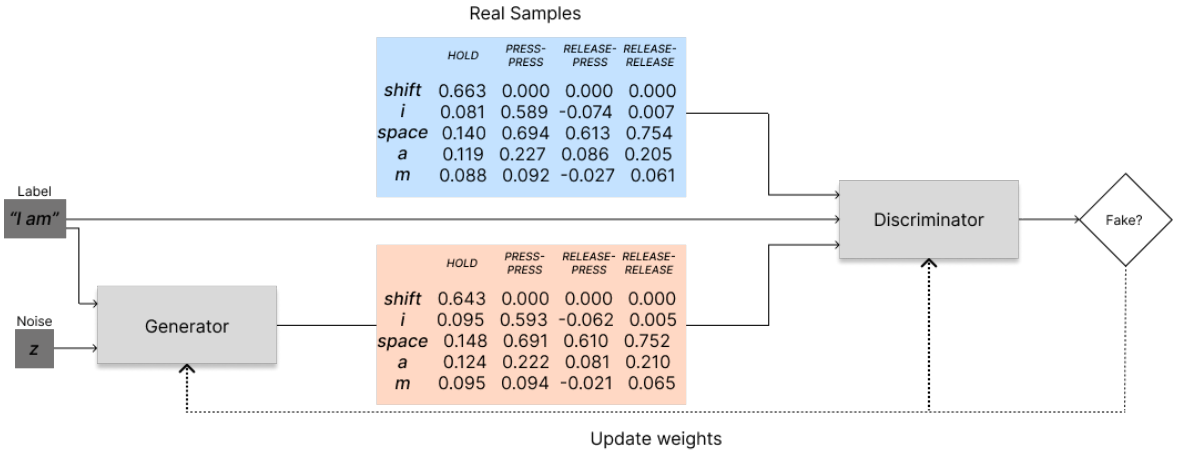

Figure 2.3 Conditional Generative Adversarial Network. Generator is responsible for creating synthetic keystrokes from a random noise and a specific input sentence, discriminator – marking those keystrokes as fakes

2.3 RELATED WORK

In this section, we will review the related work that has already been done using keystroke dynamics and generative deep learning.

Çeker et al. [15] demonstrated that the Support Vector Machines (SVM) model can be used as a classification engine for active authentication mechanisms. Free text data was collected in the same institution among 39 participants in two sessions in a period of 2 months. By analysing Press-Press Time (flight time) and Press-Release Time (dwell time) of the 12 most common digraphs, they were able to differentiate 34 uses with EER of 2.94%.

Zijian Fan [16] investigated the potential use of generative models in recreating user behaviour that can *bypass* continuous authentication. Fixed text data was used from the CMU keystroke dataset [17] and the Touchalytics dataset [18] without extra test data collected in real time. To test the hypothesis, four different types of CA were trained to distinguish between legitimate users and attackers. The finding showed that when trained with sufficient data, GAN is capable of producing samples that can bypass well-trained classifiers with average FAR of 80% amongst 4 classifiers.

Acien et al. [19] describes the way to authenticate users utilising keystroking patterns. The fixed text dataset from Aalto University [12] was used with no additional test data collected in real time. The authors suggest creating an RNN-architecture model trained in the Siamese Framework. As input, the model takes two aligned sequences and outputs the probability of

those sequences being typed by the same person. The resulting architecture was capable of classifying 1K users with an average EER of 4.8%.

The paper by Eizaguirre-Peral et al. [20] aimed to create a cGAN to mimic the keystroke of the legitimate user to conduct a presentation attack. The paper also used Aalto University dataset for training and testing purposes with no extra test data collected in real-time. The network was trained on a single-user dataset and conditioned by a word embedding, which allowed it to generate the way a certain word would've been written by a particular user. After that, the Siamese network proposed in [19] was used to define whether two keystrokes belong to the same user or different ones. The resulting model showed 95% accuracy of correctly mimicking a selected user.

As can be seen, the research studies mentioned above have primarily focused on utilising GAN to bypass CA systems. However, in our work we aim to propose an approach that leverages cGAN as the primary security system itself. It is also worth mentioning that in [20] only training data was used to obtain the final performance of the model. In this paper, we will also test cGAN on free-text real-time data.

3. DATASET

In this work, data analysis in Section 4 and all experiments in Section 5 will be conducted using the Aalto University Dataset [12], which amounts to more than 16GB of keystroke data collected from 168000 volunteers in the online study. The task was to type 15 English sentences that were memorised by the participants one by one. All in all, the dataset contains roughly 136 million keystrokes data, each of which contains a timestamp when a particular key was pressed and released, which allows to extract features mentioned in Section 2.1.

4. DATA ANALYSIS

The data analysis section of this paper aims to examine the dataset mentioned in Section 3. Our work has aim of verifying the study that was published with the data [12] while also filling the gaps in the research. We will show visualizations with the statistical data to gain insights into the collected behavioural patterns within the defined groups of participants. The graphs presented further will be augmented with kernel density estimation (KDE) to provide a smoother and more continuous representation of the underlying probability density function.

The collected data allows us to classify the participants into two distinct groups based on their prior typing training: trained and untrained. Furthermore, following the related work, the participants are categorized into slow and fast typists according to a predetermined threshold. This threshold was chosen to be equal to the 80th quantile of all recorded Words Per Minute (WPM) values and amounts to 68 WPM.

Before the analysis, data was initially pre-processed. For the sake of consistency, all keystrokes that were collected on non-qwerty keyboards were removed from the dataset. Additionally, the data gathered from touch and small physical keyboards was also deleted.

Various statistical measures will be provided for the comparison of the distributions. Specifically, the mean and standard deviation will be used to describe the central tendency and spread of the distribution, respectively. Additionally, we will use Fisher's kurtosis, that measures the degree of concentration of observations in the tails of the distribution. Finally, the Fisher-Pearson coefficient of skewness will be presented to indicate whether the distribution is symmetric or skewed and in which direction.

4.1 METRICS OF TYPING EFFECTIVENESS

4.1.1 WORDS PER MINUTE

The Words per Minute (WPM) metric was calculated by taking the number of words that were typed during the timeframe and dividing it by the actual time spent. The mean value of WPM is estimated to be 51, with a standard deviation of 20.15. The distribution's kurtosis is -0.11 indicates a flatter and more dispersed distribution compared to a normal one. The skewness coefficient of the distribution, which is 0.5, shows that it is positively skewed. The recorded WPM numbers cover a wide range, with the quickest typewriter reaching a speed of 152 WPM and the slowest typist only reaching 3.9 WPM. These results suggest that the typing speed among individuals can vary greatly, with some people typing at a much faster rate than others.

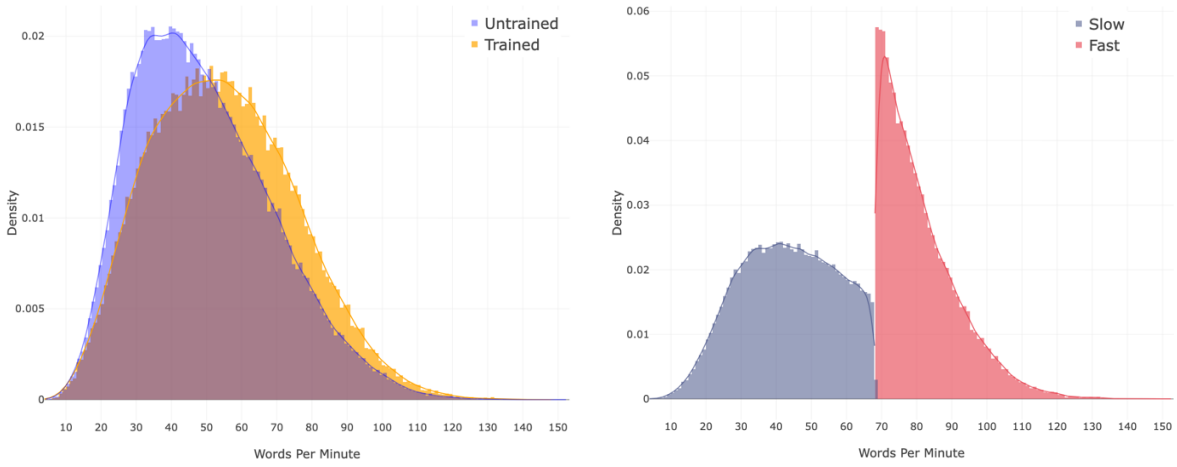

Figure 4.1 Histograms of Words per Minute for Untrained / Trained (left) and Slow / Fast (right) participants

Figure 4.1 depicts the distributions for the groups of participants. The left graph shows the impact of taking a typing course on typing speed. The difference in mean values of the untrained and trained group is relatively small, with the value for the former being 54.8 and the value for the latter being 49.3, having the similar standard deviations with a slight difference of 1 in favour of the trained group. The kurtosis coefficients indicate that the distribution for the trained group is flatter, compared to the one of the untrained group, with respective values of -0.26 and 0.02. Furthermore, it is evident that the distribution for the untaught group is more right-skewed than that of the training group. It suggests that taking a typing course might have a positive impact on improving WPM.

The distributions for fast and slow typists are shown in the right-hand graph. To begin with, the mean of the quick individuals is expectedly larger at 81.23 than that of the slow participants, who only have a mean of 43.28. The standard deviation of the former is lower at 10.93, compared to the latter, with a value of 13.65. It suggests that fast typists' distribution is more packed around the mean. The kurtosis coefficient for quick writers is high and positive at 1.675, showing a peaked central region and a heavy tail. For the slow participants, the kurtosis value is equal to -0.88, which means that the distribution is flatter and has less concentration of observations in the tails. Finally, the Fisher-Pearson coefficient of skewness is positive for the fast typists at 1.23, indicating right-skewness. For the slow typists, the coefficient is negative at -0.09, showing slight left-skewness.

4.1.2 INTER-KEY INTERVAL

The average value of Inter-Key Interval (IKI) for all participants is around 236 milliseconds, with a standard deviation of 109.21 milliseconds. The distribution has a kurtosis value of 7.4 and a skewness coefficient of 1.995. It indicates that it has a relatively peaked central region and is strongly skewed to the right, suggesting that the majority of contributors tend to type with an IKI close to the mean value.

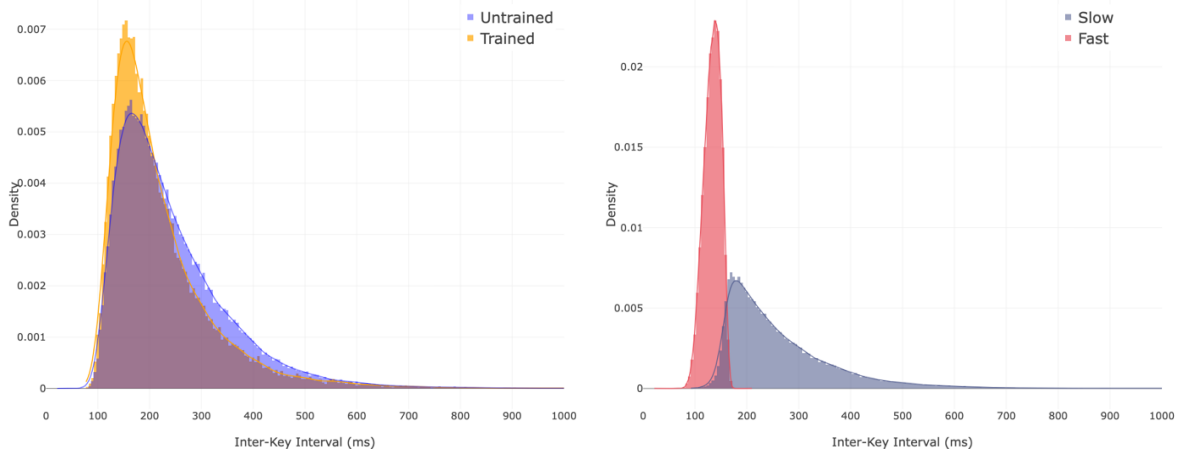

Figure 4.2 Histograms of Inter-Key Intervals for Untrained / Trained (left) and Slow / Fast (right) participants

Figure 4.2 shows the distributions for the same groups of participants. The mean IKI for trained individuals is 220.41 milliseconds and is lower than for untrained contributors, which amounts to 243.27 milliseconds. In addition, the standard deviation for experienced individuals is also lower (104.16 vs 110.66), indicating that the data for this group are more clustered around the mean. Fisher's kurtosis for trained individuals (9.94) is significantly higher than for untrained ones (6.70). It indicates that the distribution for trained subjects has heavier tails and

higher peaks than for untrained subjects. The skewness coefficient for trained individuals at 2.39 is also higher compared to the other group, which has a value of 1.86. It is possible to conclude that trained individuals have quicker typing motor skills, reflected in a more compressed distribution of the values around the mean value.

As for fast and slow typists, the average IKI for the former is 133.40 milliseconds, whereas the latter has roughly twice that – 262.54 milliseconds. The standard deviation for fast people is also lower (16.14 vs 107.31), indicating a more compressed distribution of values for this group. Fisher's coefficient of kurtosis for quick people (-0.34) is closer to zero than for slow people (8.31), indicating that the distribution of inter-key intervals for fast people is flatter than for slow people. The skewness coefficient is closer to zero for quick typists (-0.34) and higher for the other group (2.13).

4.1.3 KEYPRESS DURATION

Data points exceeding 10 seconds were excluded from this metric. The mean of keypress durations is 119.78 milliseconds with a standard deviation of 31.3 milliseconds. The kurtosis and skewness coefficients amount to 31.88 and 3.21, indicating that the distribution is peaked around the mean and has a significant skew towards the right tail.

Keypress durations for typists' groups mentioned above are demonstrated in Figure 4.3. For clear visualizations, the x-axis on both graphs was limited to the interval of [0, 400]. Nevertheless, it is worth mentioning that there are outliers with 900 milliseconds in the dataset.

The average keystroke time for trained (121.17 ms) and untrained users (119.13 ms) is about the same at this scale, which can also be observed from the graph. However, the standard deviation for untaught users is higher (31.83 ms vs 30.29 ms), which may indicate a greater diversity in the way this group presses keys. Fisher's kurtosis coefficients are high in both cases (32.46 and 31.71), indicating the presence of outliers in the distribution of keypress times in both groups. The coefficient of skewness is high in both categories (3.15 and 3.24), suggesting a right-skewed distribution. In conclusion, it can be said that the average keystroke duration is similar for trained and untrained users. However, unskilled users may have a more varied keystroke pattern, reflected in a higher standard deviation.

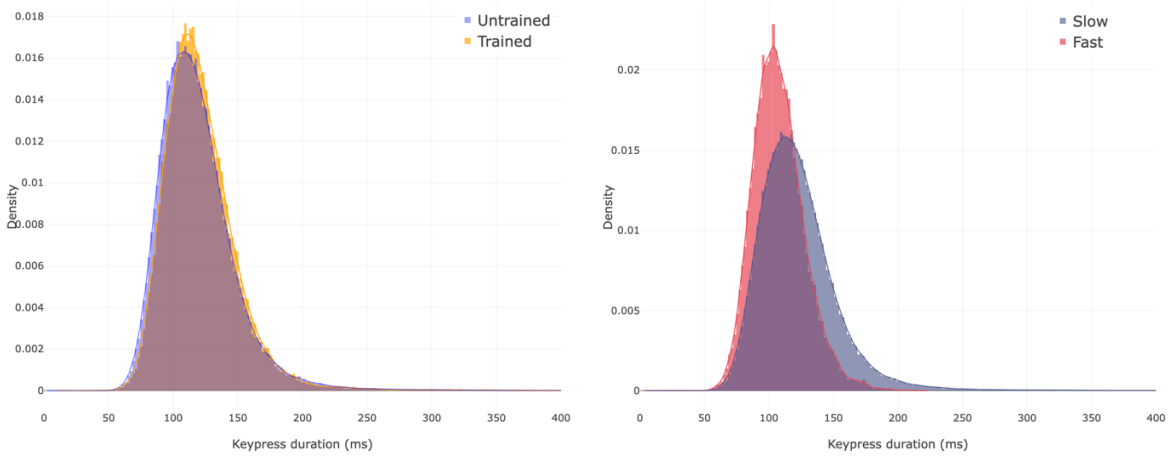

Figure 4.3 Histograms of Keypress Durations for Untrained / Trained (left) and Slow / Fast (right) participants

The difference in the average keypress time between fast and slow typists is approximately 15 ms (107.68 ms and 122.84 ms, respectively). The standard deviation for slow users (32.99 ms) is higher than for quick users (19.82 ms). The kurtosis coefficient for fast typists is low at

1.04 compared to the one for slow typists at 31.37, confirming the presence of the outliers mentioned earlier. The skewness coefficients are 0.64 and 3.26 for the former and the latter, respectively, indicating that the distribution for the quick typists is almost symmetrical, while for the slow typists, it is right-skewed. In conclusion, the keystroke pattern is more homogeneous for fast users and more varied for slow ones.

4.1.4 KEYSTROKES PER CHARACTER & ERROR CORRECTIONS PER CHARACTER

These metrics have been put together in one part because they are interrelated. The skewness for all groups is very similar; therefore – only the mean and standard deviation will be used for the comparison. A lower value for both Keystrokes per character (KSPC) and Error Corrections per Character (ECPC) would indicate faster and more accurate typing, thus giving an idea of the effectiveness of the typist's keystroke technique. Conversely, higher values may suggest a slower and more error-prone typing style.

KSPC metric has a minimum value of 1.01 and a maximum of 4.96. The mean value is 1.17, with a standard deviation of 0.09. It indicates that, on average, it takes just one keystroke to generate a character, which is an expected result. ECPC's mean value is 0.06, with a standard deviation of 0.04. All data points lie within a range from 0.0 to 1.9. It shows that there exist participants who either do not *make* mistakes or do not *correct* them at all.

Figure 4.4 illustrates distributions of KSPC and ECPC that also pertain to the same groups as the previous metrics' comparisons. Untrained participants have a moderately higher KSPC mean value of 1.176, compared to trained ones, with a mean of 1.161. The standard deviation was also slightly higher – 0.0947 for the former and 0.0905 for the latter. As per ECPC, the mean value of the untaught contributors is again moderately higher (0.064) compared to the trained group (0.058). The standard deviation shows the same tendency, with 0.044 for unskilled and 0.042 for skilled.

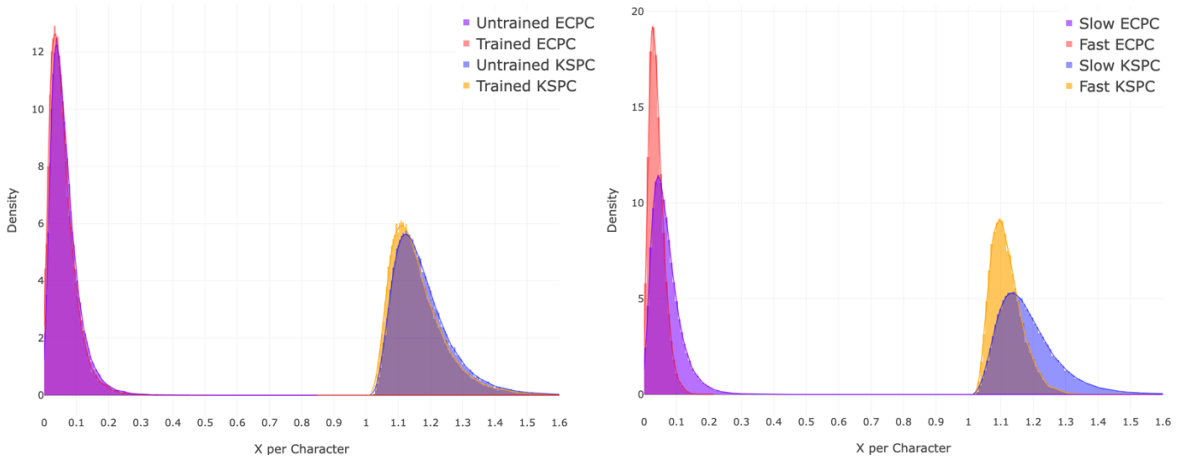

Figure 4.4 Histograms of Keypress per Character (KSPC) and Error Correction per Character (ECPC) for Untrained / Trained (left) and Slow / Fast (right) participants

The mean KSPC value for fast typists is 1.118, which is lower than for slow typists at 1.185. Additionally, the standard deviation of KSPC for quick users is lower than that of slow ones, with values of 0.049 and 0.097, respectively. Furthermore, slow participants have a higher mean value of ECPC of 0.0688 compared to the fast ones with 0.039. The standard deviations are 0.046 and 0.023 for fast and slow groups.

In conclusion, we can highlight that the difference in KSPC between trained and untrained participants is insignificant. However, ECPC data shows that, on average, the former group makes fewer error corrections, which could suggest that training has a positive impact on reducing the number of mistakes. Additionally, as expected, fast typists tend to make fewer keystrokes per character than slow ones while also making a smaller number of error corrections.

4.1.5 UNCORRECTED ERROR RATE

The minimum value of uncorrected error rate is 0%, while the maximum reaches 24.96%, indicating that some participants left almost a quarter of the provided text incorrect. On average, the contributors had an error rate of 1.15%. The standard deviation was high at 1.41, showing that the metric varied significantly across the participants. The kurtosis value of 24.21 indicates a very sharp and high peak with long tails, and the skewness coefficient of 3.63 shows a strong right skew.

Table 4.1 Statistics of typing metrics for different groups of typists

Metric	Group	Mean	Standard Deviation	Kurtosis coefficient	Skewness coefficient
Words per Minute	All	51.0082 ms	20.1549 ms	–0.1114	0.5085
	Trained	54.8551 ms	20.6566 ms	–0.2696	0.3263
	Untrained	49.3040 ms	19.6900 ms	0.0280	0.5913
	Fast	81.2331 ms	10.9300 ms	1.6758	1.2345
	Slow	43.2834 ms	13.6510 ms	–0.8829	–0.0906
Inter-Key Interval	All	236.249 ms	109.212 ms	7.4022	1.9951
	Trained	220.407 ms	104.157 ms	9.9436	2.3914
	Untrained	243.276 ms	110.656 ms	6.7018	1.8635
	Fast	133.401 ms	16.1429 ms	–0.3363	–0.3437
	Slow	262.535 ms	107.307 ms	8.3107	2.1332
Keypress duration	All	119.782 ms	31.3018 ms	31.882	3.2193
	Trained	121.172 ms	30.2869 ms	32.457	3.1521
	Untrained	119.130 ms	31.8259 ms	31.705	3.2409
	Fast	107.681 ms	19.8156 ms	1.0381	0.6421
	Slow	122.843 ms	32.9981 ms	31.370	3.2619
Keystrokes per Character	All	1.1717	0.09370	27.012	2.3334
	Trained	1.1613	0.09050	69.359	3.1668
	Untrained	1.1763	0.09471	11.865	2.0237
	Fast	1.1183	0.04980	46.489	2.2121
	Slow	1.1854	0.09733	27.131	2.2670
Error Corrections per Character	All	0.0627	0.04438	26.707	2.3490
	Trained	0.0583	0.04297	75.856	3.3281
	Untrained	0.0647	0.04485	8.7920	1.9768
	Fast	0.0391	0.02340	1.4982	1.0112
	Slow	0.0688	0.04641	26.773	2.2863
Uncorrected error rate	All	1.1495	1.40759	24.210	3.6260
	Trained	1.0027	1.2834	34.045	4.1717
	Untrained	1.2145	1.4544	21.354	3.4433
	Fast	0.7795	1.0155	47.226	4.5289
	Slow	1.2440	1.4765	21.760	3.4702

The uncorrected error rate, on average, is higher for untrained participants, compared to the trained ones, with a mean of 1.21% and 1%, respectively. The standard deviation is also higher for the untaught group (1.45 vs 1.28). The Fisher's kurtosis and skewness coefficients for both trained and untrained show that the distributions are strongly skewed to the right, with a high number of outliers. These results might indicate that trained participants tend to fix their typing errors. It can be assumed, that during the typing courses, contributors were taught to correct the mistakes. Thus, it takes them less time and seems worthwhile to fix the errors with no relative time loss, as they have gotten used to it.

The uncorrected error rate for fast typists has a lower mean of 0.779% compared to the slow participants, with a mean value of 1.244%. The same tendency is preserved for the standard deviations – 1.0154 vs 1.4765, respectively. In terms of Fisher's kurtosis measure, the coefficient is significantly higher for fast typists at 47.22 than for the slow ones at 21.76, showing that the distribution of uncorrected error rate for quick contributors has a sharper peak. The skewness coefficients are high for both groups – 4.5289 and 3.4702 for fast and slow typists, respectively, indicating that both distributions are skewed towards higher error rates. Overall, these statistics suggest that, on average, quick contributors fix more errors than slow ones, as they might be aware that it does not impact their general typing pace in terms of time spent.

4.2 DETERMINANTS OF TYPING PERFORMANCE

In this section, we discuss factors that may affect typing performance. Specifically, we will assess the potential influence of a self-reported number of fingers used and rollover rate on the typing performance indicators WPM and KSPC. Here, we assume that KSPC reflects the errors made during data collection.

In addition, we introduce a group of high performers. The participant is defined as one if the recorded WPM is greater than the 70% quantile (61 WPM) and KSPC is smaller than the 30% quantile (1.10). In the final part of this subsection, we will compare the Inter-Key Interval, Rollover Rate, and Keypress Duration of high performers to other contributors.

4.2.1 NUMBER OF FINGERS

During the study, participants were instructed to report the number of fingers they usually use for typing. This data can provide insights into the potential association between the number of utilized fingers and WPM.

Table 4.2 Statistics of WPM for different self-reported number of fingers

Number of Fingers	Mean (ms)	Standard Deviation (ms)
1-2	40.459	18.003
3-4	41.334	15.884
5-6	45.983	17.257
7-8	50.288	18.122
9-10	57.740	20.357

As can be seen from Figure 4.5 and Table 4.2, there is, on average, a positive correlation between number of used fingers and typing speed.

It should be mentioned that the distribution for 9-10 reported fingers covers a wide range of values from 5.7 to 152 WPM, with the highest standard deviation value at 20.38. Therefore, it is impractical to define how fast the participant is only looking at the number of fingers used.

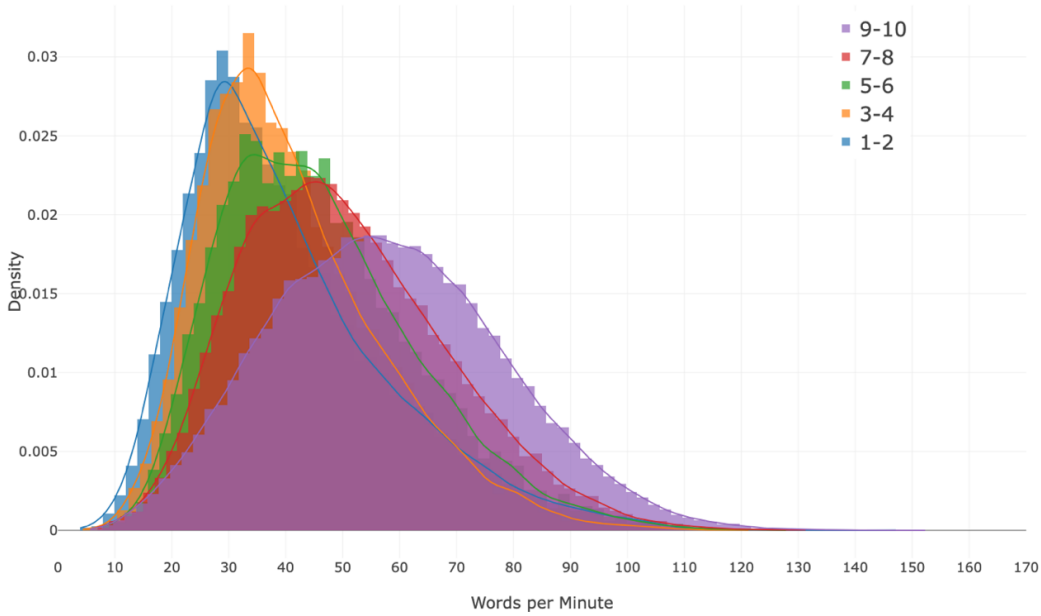

Figure 4.5 Histograms of Words per Minute for different self-reported number of fingers

The second factor that we considered as a typing performance indicator is Keystroke Per Character. In Figure 4.6, the KSPC distributions for different reported numbers of fingers used for typing are presented. The graph suggests that a “9-10” group demonstrates the smallest mean value for KSPC, meaning that its representatives’ writing is the most accurate compared to other groups.

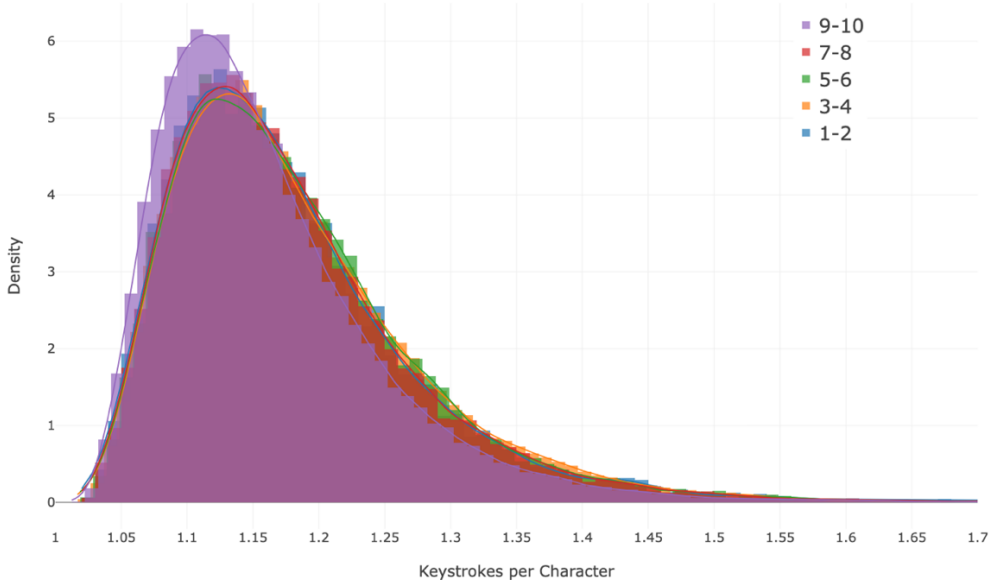

Figure 4.6 Histograms of Keystrokes per Character for different self-reported number of fingers

From Table 4.3, it is possible to draw a conclusion about all the groups presented. First, the mean values show a downward trend from the group '1-2' to '9-10', illustrating that the greater the number of fingers, the fewer mistakes the participant makes.

Therefore, the more fingers are used, the faster a user types and the more the group distribution becomes closer to a normal one. This means that the scale of the KSPC values becomes more homogeneous, while the typing patterns are more similar to the other representatives of the group.

Table 4.3 Statistics of Keystrokes per Character for different self-reported number of fingers

Number of Fingers	Mean	Standard Deviation
1-2	1.179	0.104
3-4	1.183	0.097
5-6	1.181	0.095
7-8	1.179	0.096
9-10	1.161	0.086

4.2.2 ROLLOVER RATE

As was described in Section 2, the rollover term refers to the behaviour when the consecutive key is pressed before releasing the previous one. The graphs presented in this part are scatterplots with an ordinary least squares (OLS) regression trend line, that depicts an overall tendency and interconnection between independent and dependent variable.

In the left part Figure 4.7 the relationship between the rollover rate and the WPM metric is illustrated. The findings demonstrate a positive correlation between these two variables, suggesting that as the rollover rate increases, the typing speed tends to be faster. The Pearson correlation coefficient of 0.731 shows a strong positive linear relationship between these two variables. These results show that the rollover rate is a significant factor that impacts typing performance.

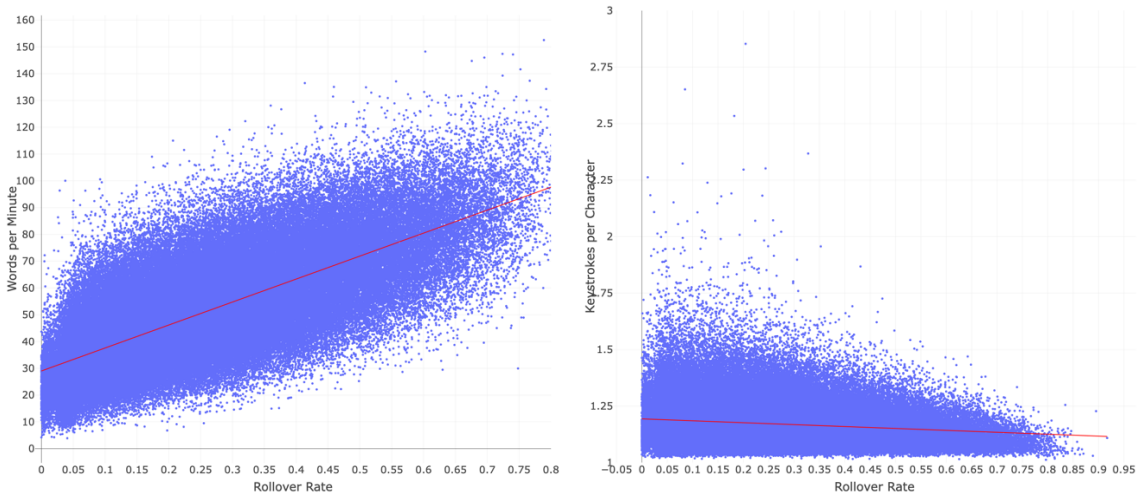

Figure 4.7 Rollover Rate vs Words per Minute (left) and Rollover Rate vs Keystrokes per Character (right)

The dependence of KSPC on the rollover rate is shown in the right part of Figure 4.7. On the basis of the evidence presented in the graph, it is possible to infer a slight impact. The correlation coefficient of -0.15 implies that the more rollover behaviour occurs, the smaller the value of KSPC is, consequently demonstrating better typing accuracy.

4.2.3 HIGH PERFORMERS – INTER-KEY INTERVAL COMPARISON

As was mentioned in the beginning of this subsection, we will further conduct the comparison of some metrics between high performers and other participants.

Figure 4.8 illustrates the IKI distributions for high-performing and regular users. First, the difference in the mean values of these groups is roughly 110 milliseconds (136 and 246ms, respectively). It confirms that, on average, the IKI of a high performer is much smaller than that of the other users, leading to much faster typing.

Nevertheless, it is also seen that there is no strict boundary that would allow for a precise determination. It is reflected in the standard deviations for these distributions, with values of only 18.32 and 109.53 for high performers and other contributors, respectively. The skewness and kurtosis coefficients demonstrate a significant difference in the shape of the distributions. For high performers, corresponding values are at -0.24 and -0.56, while for the other participants – at 1.98 and 7.45, respectively. It shows that the latter distribution has heavier tails and a broader range of values covered, with a notable right skew.

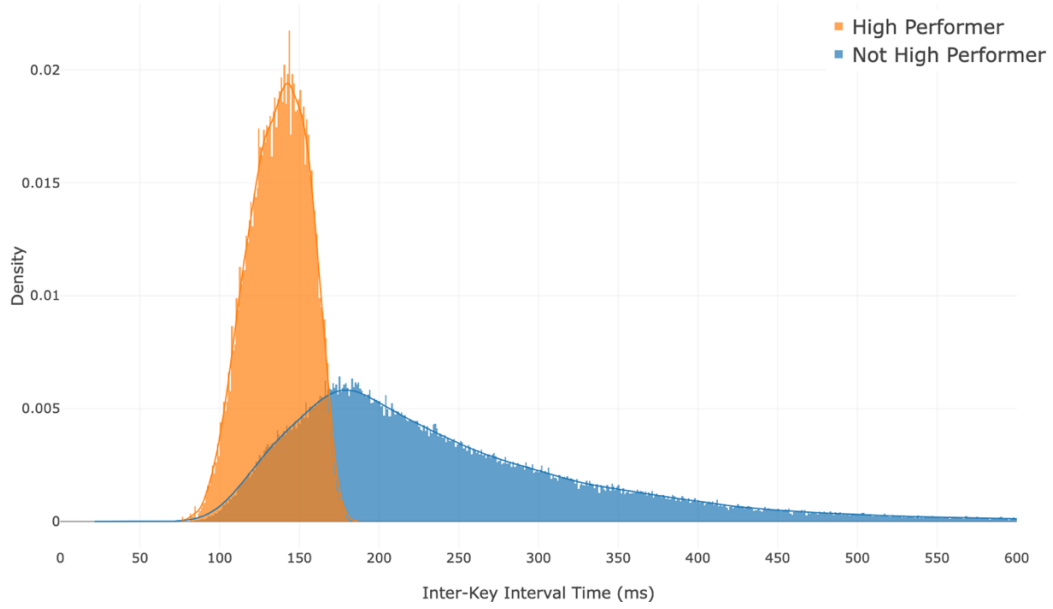

Figure 4.8 Histograms of Inter-Key Intervals for high performers and other participants

In conclusion, the graph suggests the inability to decide if the person has a high typing effectiveness by looking only at the Inter-Key Interval value. Although it can be used to define speed (Section 4.1), from Figure 4.8, it can be seen that many participants with IKI values in the range of high performers are not considered as such due to the number of mistakes reflected in KSPC.

4.2.4 HIGH PERFORMERS – ROLLOVER RATE COMPARISON

Figure 4.9 demonstrates distributions of rollover rate for high performers and other participants.

The mean values for these two groups are 0.433 and 0.238, correspondingly. Although the difference is roughly double, it is seen that almost the whole possible range from 0 to 1 is covered by representatives from both groups. The standard deviations do not differ much, with

0.163 for regular contributors and 0.154 for high performers. As per the shapes of the distributions, skewness coefficients have values of −0.018 and 0.718, with kurtosis values at −0.55 and −0.14 for high performers and the other participants.

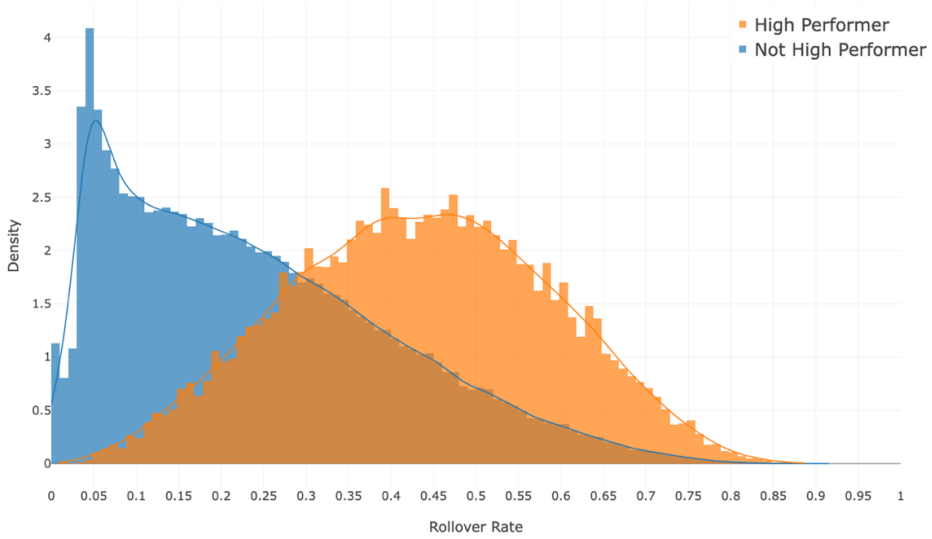

Figure 4.9 Histograms of Rollover Rate for high performers and other participants

In summary, results indicate that the rollover rate cannot be treated as a determinant of typing performance, as the behaviour is quite common for both categories of contributors, even though, on average, it is more intrinsic to the high performers.

4.2.5 HIGH PERFORMERS – KEYPRESS DURATION COMPARISON

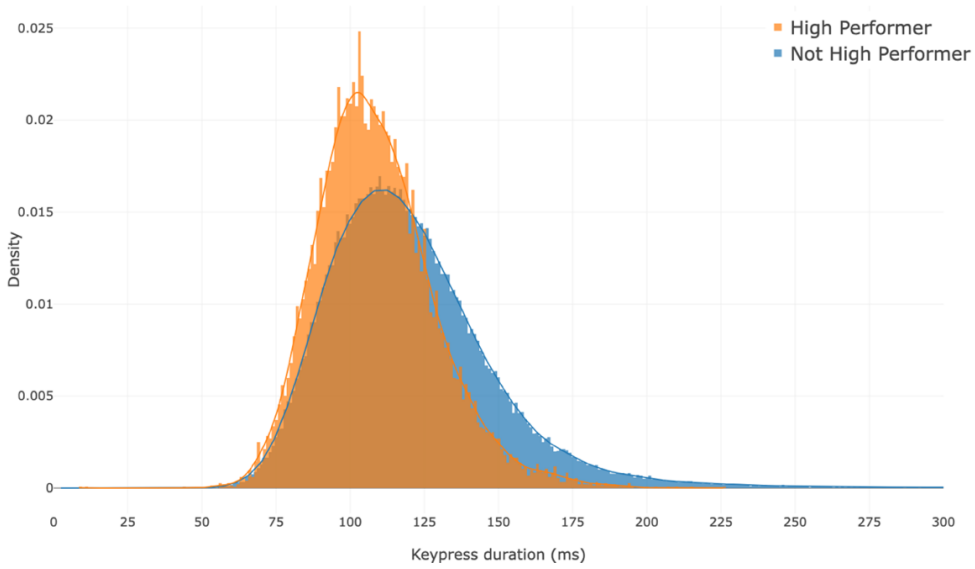

Figure 4.10 Histograms of Keypress duration for high performers and other participants

Figure 4.10 shows the distributions of keypress durations for high performers and other participants. Comparing the statistics for these two groups, the average durations for high

performers and the other contributors are 108.9 and 120.8 ms, with the standard deviations being 19.8 and 32.8, respectively.

It suggests that the "high performers" group had a shorter average keypress duration and less variation compared to the other participants. Both distributions are right-skewed, with the skewness coefficients at 0.65 for high performers and 2.92 for another group. Additionally, kurtosis coefficients show a significant difference in the tails. The corresponding values are 1.06 for high performers and 23.57 for other participants.

In summary, while there is a moderate difference between the groups, the metric of keystroke duration alone cannot serve as a proper indicator of a participant's typing performance, since the range of values covered by both distributions, on average, is the same.

Table 4.4 Statistics of typing metrics for high performers and other participants

Metric	Group	Mean	Standard Deviation	Kurtosis coefficient	Skewness coefficient
Inter-Key Interval	High Performers	136.8079 ms	18.3727 ms	–0.5620	–0.2418
	Others	246.2483 ms	109.537 ms	7.4553	1.9846
Rollover Rate	High Performers	0.4335	0.1541	–0.5521	–0.0188
	Others	0.2385	0.1632	–0.1419	–0.5521
Keypress duration	High Performers	108.9000 ms	19.8698 ms	1.0628	0.6547
	Others	120.8252 ms	31.8374 ms	23.5761	2.9205

4.3 HAND PERFORMANCE COMPARISON

This part will discuss the differences in the typing behaviour for different hands by analysing Inter-Key Intervals for bigrams. Initially, the data do not include information about the fingers and hands with which a particular key was typed. Following the dataset paper mentioned earlier, we used the How We Type dataset [21] in order to add these details. Eventually, each bigram had information about the hand with which it was typed – left, right, or both. It should be noted that the “Space” key was excluded from the analysis, since a pressing hand strongly depends on previous keystrokes.

4.3.1 GENERAL COMPARISON

Figure 4.11 illustrates the distributions of Inter-Key Intervals for left, right and altering hands. From the graph, we can infer that the bigrams typed with hand alteration were the fastest. The mean value for altering hands is 76.80 ms, followed by the left hand with 81.59 ms and the right hand with 112.28 ms. The standard deviation values are 105.26, 101.92 and 132.18 ms, correspondingly. Same tendency is preserved for letter repetitions bigrams – left hand is faster that the right one with the mean values at 88.5 and 99 ms, respectively.

The vertical red line on a graph represents the rollover boundary at 0. As can be seen, rollover is inherent in all groups. However, the distribution of the right hand has the heaviest left tail. It is because the right hand is mostly used for the “Shift” button to start sentences and proper names.

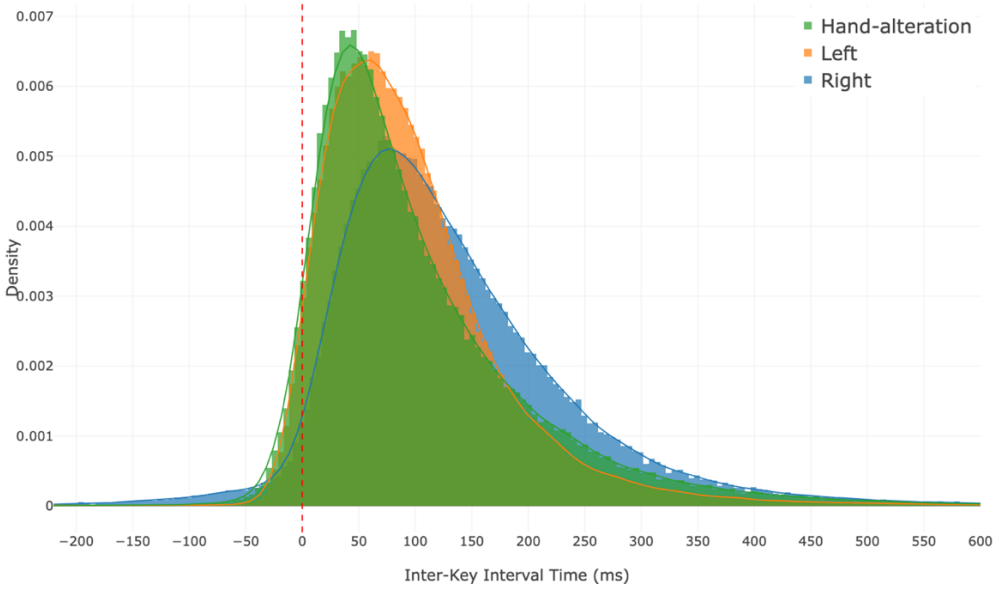

Figure 4.11 Histograms of Inter-Key Interval Time for different hands used for typing

The results obtained in our study contrast with the findings presented in the dataset paper, which reported that the left hand was marginally slower than the right hand. On the other hand, our conclusions are in line with [22]. Additionally, to support the statement, [21] shows that the left hand generally uses more fingers and moves less compared to the right one. Furthermore, it is essential to mention that the QWERTY keyboard arrangement is made so that most of the most used keys in English are placed under the left hand [23] (Table 4.5).

Table 4.5 Top 10 most common letters in English [23]

Letter	Count	Percent	Hand
E	445.2 B	12.49%	Left
T	330.5 B	9.28%	Left
A	286.5 B	8.04%	Left
O	272.3 B	7.64%	Right
I	269.7 B	7.57%	Right
N	257.8 B	7.23%	Right
S	232.1 B	6.51%	Left
R	223.8 B	6.28%	Left
H	180.1 B	5.05%	Right
L	145.0 B	4.07%	Right

Moreover, 3 out of 10 of the most common bigrams in English are also typed using only the left hand [23] (Table 4.6).

In conclusion, the results can depend on the way the data was collected. The initial dataset also contains the keystrokes collected from mobile devices, not only computers and laptop keyboards. Additionally, different layouts were present, such as QWERTZ, AZERTY, and DVORAK.

Table 4.6 Top 10 most common bigrams in English [23]

Bigram	Count	Percent	Hand
TH	100.3 B	3.56%	Alteration
HE	86.7 B	3.07%	Alteration
IN	68.6 B	2.43%	Right
ER	57.8 B	2.05%	Left
AN	56.0 B	1.99%	Alteration
RE	52.3 B	1.85%	Left
ON	49.6 B	1.76%	Right
AT	41.9 B	1.49%	Left
EN	41.0 B	1.45%	Alteration
ND	38.1 B	1.35%	Alteration

4.3.2 TRAINED & UNTRAINED PARTICIPANTS

Table 4.7 shows the statistics for different hands used for typing for trained and untrained participants. We can observe that hand alteration bigrams were the fastest among the trained contributors, with the mean values of 87.96 ms for altering, 88.15 ms for the left hand, and 116.92 for the right one. As per untrained typists, interestingly, we can see that the left hand was typing with the lowest IKI value, which could be due to the same reason, as was described in Section 4.3.1. It is possible to assume that the training focuses on improving fine motor skills of each hand separately rather than on enhancing a pair work of both hands. Generally, we can observe that the training process had a beneficial impact on each bigram hand – all show a positive decrease of ~ 23 ms in IKI values.

Table 4.7 Inter-Key Interval statistics for different hands used for typing for untrained and trained participants

Bigram Hand	Group	Mean (ms)	Standard Deviation (ms)
Right	Trained	116.9201	108.1039
	Untrained	138.9460	113.6319
Left	Trained	88.1504	83.4337
	Untrained	102.5164	88.6157
Alteration	Trained	87.9615	101.2513
	Untrained	112.9250	109.0404
Right (repetition)	Trained	115.3858	88.3479
	Untrained	117.0642	85.2348
Left (repetition)	Trained	101.9205	70.6645
	Untrained	107.1066	93.5487

Furthermore, data for key repetition for different hands are also presented in Table 4.7. First, repeating symbols for the right hand was faster than for other bigrams for the same hand. Surprisingly, the left hand showed the opposite trend – repetitions took more time than regular

bigrams. Following the previous findings, we can see that the left hand was the fastest for repeating keystrokes and that the training had a slight positive effect on the typing speed.

4.3.3 FAST & SLOW PARTICIPANTS

As can be seen in Table 4.8, the average difference between fast and slow participants for all hands (with no repetitions) is ~ 90 ms. For quick typists, hand alteration bigrams were the swiftest, with the mean at 18.57 ms, followed by the left hand (29.69 ms) and the right hand (39.18 ms). The trend is slightly different for the slow participants – the left hand was the quickest, with the average at 115.58 ms, followed by hand alteration (127.41 ms) and the right hand (155.95 ms).

As per the repetitions, the average difference between mentioned groups of participants is ~ 30 ms for both hands. Interestingly, it can be observed that for the fast typists, the IKI values of the repeating symbols are greater than those of the other bigrams. The situation is different for the slow contributors: both hands needed less time to press the same key twice than two different ones. Additionally, the IKI values for repetitions for the slow participants were smaller than the one for the bigrams typed with altering hands.

Table 4.8 Inter-Key Interval statistics for different hands used for typing for fast and slow participants

Bigram Hand	Group	Mean (ms)	Standard Deviation (ms)
Right	Fast	39.1879	35.9871
	Slow	155.9518	112.9739
Left	Fast	29.6968	28.3337
	Slow	115.5899	88.6804
Alteration	Fast	18.5739	22.8364
	Slow	127.4165	109.1735
Right (repetition)	Fast	90.6835	54.6569
	Slow	123.1592	91.3925
Left (repetition)	Fast	82.2113	29.8928
	Slow	111.7246	85.5452

4.4 CONCLUSION

As was mentioned at the beginning of Section 4: it had the aim of verifying the study that was published with the data and filling the gaps in the research.

Starting with the metrics of the typing effectiveness part, we repeated the same actions as the authors of the initial work, except further introduced our own vision of the pre-processing. We showed the visualizations not presented in the dataset study and managed to reproduce the same results. Additionally, we compared different groups of participants in more detail using metrics such as Fisher's kurtosis and Fisher-Pearson coefficient of skewness.

In the "determinants of typing performance" section, we enriched the analysis made in the dataset study with graphs. Additionally, we introduced the "high performers" group for further analysis with the other research participants.

Finally, in hand performance comparison, we conducted a similar analysis, which was presented in the initial paper. Although most of the results were in line with the study, we found one inconsistency and described it with the findings from other research.

5 PROPOSED APPROACH

5.1 GENERAL IDEA

As mentioned previously, the main purpose of this work is to propose an approach to a seamless continuous authentication system. The scheme of a suggested system is depicted in Figure 5.1.

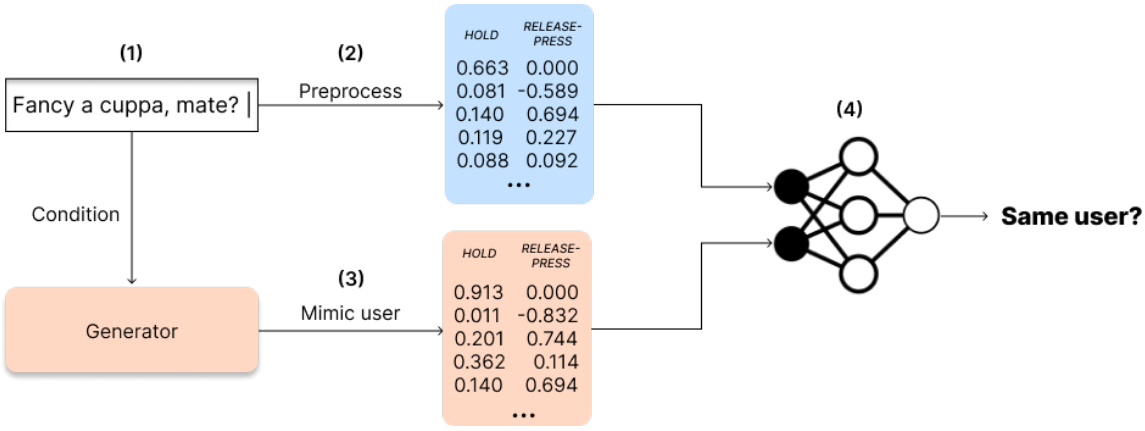

Figure 5.1 An outline of the proposed approach. The user's keystroke data collected in real-time is being passed to the cGAN's generator in order to mimic a fixed user's behaviour. Then, the authentication network is used to compare the generated sample with the real-time data to define whether the user is legitimate or not

The system will go through the following steps:

1. Collect a user's input in real time from text field. In this step, the system will record user keystrokes with each key press and release timestamps (in milliseconds).
2. Prepare (pre-process) the data to be in a proper format. Here, the system will calculate Hold Time (the difference between release and press timestamp for each key) and Release-Press Time (or Inter-Key Interval – the difference between the release timestamp of the previous key and the press timestamp of the next one).
3. Use the user input as a condition for the Generator network to mimic the user's behaviour for a particular sentence. Pressed keys represented as integers between 0 and 255 will be used to generate how the legitimate user would type this particular sequence of keys.
4. Compare the real-time user dynamics with the generated keystroke data using the authentication network. The final model will compare the synthesised samples with the data collected from the input field to define the legitimacy of the user.

Although the method described in [20] for training cGAN on a single user dataset could be utilised for this study, it is important to note that training such a network "from scratch" for each new user registration would require significant computational resources. Hence, our approach suggests a two-step training methodology involving pre-training and fine-tuning.

Initially, a large-scale model will be trained using extensive keystroke data collected from multiple users. The pre-training phase aims to enable the model to learn the desired output and understand the relationships between different keys. Subsequently, each newly registered user will be provided with a personalised instance of the pre-trained model, which will be further fine-tuned using only the keystrokes specific to that individual. This approach aims to reduce the time and data requirements to develop a reliable model.

It is worth mentioning that since we will use a cGAN output as a ground truth to compare the real-time dynamics, we assume that the model needs to be trained to correctly represent the

user behaviour. For that reason, the stage that includes typing the proposed sentences will be added during the sign-up procedure for each new user.

During the choice of sentences, it is essential to include as many combinations of symbols as possible. Here, the more data will be collected from each user, the more robust the personalised models will be. On the other hand, it is also important to keep the number of sentences as small as possible to ensure the convenience of using such a system. As mentioned above, the Aalto University dataset contains 15 randomly selected sentences for each participant. We acknowledge that it can be a topic for separate works and discussions that might also involve creating questionnaires to study the opinions of the potential users, but for the purposes of this work, we propose to use a set of 12 same sentences for each new user. This set contains 6 selected pangrams [24], to ensure that the data will be collected for each letter key, and 6 randomly selected sentences from IMDb review dataset [25] to include more realistic statements. The chosen sentences are as follows:

- "*Few black taxis drive up major roads on quiet hazy nights."* – pangram.
- "*Levi Lentz packed my bag with six quarts of juice."* – pangram.
- "*A quick brown fox jumps over a lazy dog.*" – pangram.
- "*Bobby Klum awarded Jayme sixth place for her very high quiz*." – pangram.
- "*Back in June we delivered oxygen equipment of the same size.*" – pangram.
- "*J. Fox made five quick plays to win the big prize.*" – pangram.
- "*I loved this movie the first time I saw it and on each subsequent viewing I always notice at least one new detail*." – part of the review on IMDb.
- "*Wasn't the shop located like right on the beach or something?*" – part of the review on IMDb.
- "*Am I the only one who sees this?*" – part of the review on IMDb.
- "*It was worth all the money I gave for it!*" – part of the review on IMDb.
- "*I can't believe there are people out there that did not like this movie!*" – part of the review on IMDb.
- "*This would have to be by far the greatest series I have ever seen.*" – part of the review on IMDb.

5.2 DATA PREPARATION

Recalling Section 3, the used dataset contains the timestamps of pressing and releasing different keys collected from typing 15 distinct English sentences by 168000 participants. To train the network, we extracted two features that will define the user's typing behaviour: a duration of keypress or hold time (difference between the release and press timestamp of the same key) and a flight time (difference between the timestamp of pressing the succeeding key and the release of a previous one).

Initially, the data were presented in milliseconds. As can be seen from Section 4, the values differ moderately from person to person, so before training the GAN it is decisive to normalise them. For that reason, we used MinMax normalization to scale the numbers between the interval [0; 1].

In a related paper [20], the authors used single words (with a length of 15 or additional padding if needed) to train the network. Although this approach is acceptable with enough data, we assume that during a registration process, a user would need to provide the system with 12 sentences to give the network the data to learn the patterns. Thus, in our work, we will use a windowed dataset with a window size of 16 and a shift of 1 at a key level. It will allow us to enrich the dataset for each user and use a more comprehensive context, including not just letters, but also other keys, like "Shift", "Space", "Backspace", etc.

5.3 MODELS IMPLEMENTATION

This section will describe the details of each model used for the proposed approach. It will show the architecture of Conditional GAN and the Authentication Network, as well as the implementation and training details of these networks. As a baseline for the comparison, we will also create a simple binary classifier trained for a single user.

5.3.1 CONDITIONAL GAN

This part will provide an overview of the cGAN architecture along with the results obtained from training the network on the dataset of 1000 users.

As mentioned previously, the first step is pre-training the network on the data obtained from many users, which should enable the model to learn patterns in the keystroking data. Then, the pre-trained model will be fine-tuned using the keystroke data from a specified user.

5.3.1.1 PRE-TRAINING

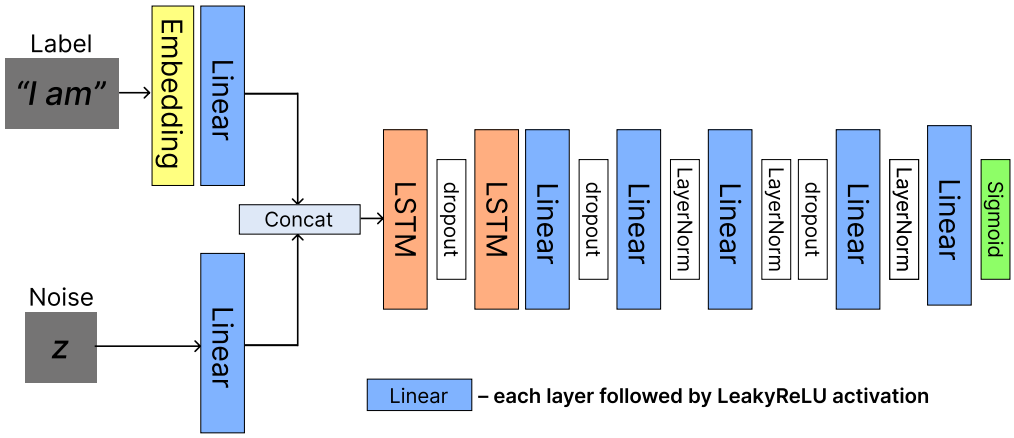

Figure 5.2 Generator architecture. The model takes keycodes and noise as input. The main module consists of two LSTM layers that allow capturing a longer "context" of keystrokes, followed by 5 linear layers, with LeakyReLU activations (with a slope of 0.05), LayerNormalizations [26] and Dropouts (0.2) [27]. The final activation function is sigmoid

The generator architecture is illustrated in Figure 5.2. Each linear layer in the network is followed by a LeakyReLU activation with a slope of 0.05. To process the condition sequence, an embedding layer with a vocabulary size of 100 and an embedding dimension of 144 is used. The resulting sequence is then transformed by a linear layer to match the shape of random noise, which consists of 500 random samples drawn from a Gaussian distribution, before being concatenated. Subsequently, the concatenated data is fed into two long-short-term memory (LSTM) layers, followed by linear layers with layer normalizations [26] and dropouts [27]. The dropout rate is set to 0.2. Finally, the activation function used is sigmoid, as the data is normalized within the range of [0, 1].

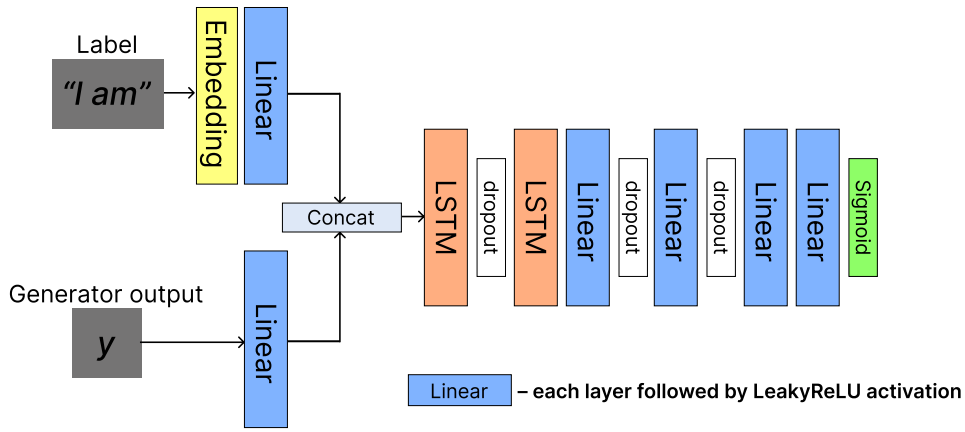

Figure 5.3 Discriminator architecture. The model takes keycodes and the generator output as input. The main module consists of two LSTM layers that allow capturing a longer "context" of keystrokes, followed by 4 linear layers, with LeakyReLU activations (with a slope of 0.05), and Dropouts (0.2) [27].The final activation function is sigmoid

The architecture of the discriminator is depicted in Figure 5.3. The structure of this network is similar to the one of the generator, except that it has a smaller number of linear layers after LSTM. The number of linear layers is reduced, as the discriminator's main objective is to distinguish between fake and real samples, and it does not need to perform those many calculations as the generator. The dropout rates and slopes of LeakyReLU are equal to 0.2 and 0.05, respectively.

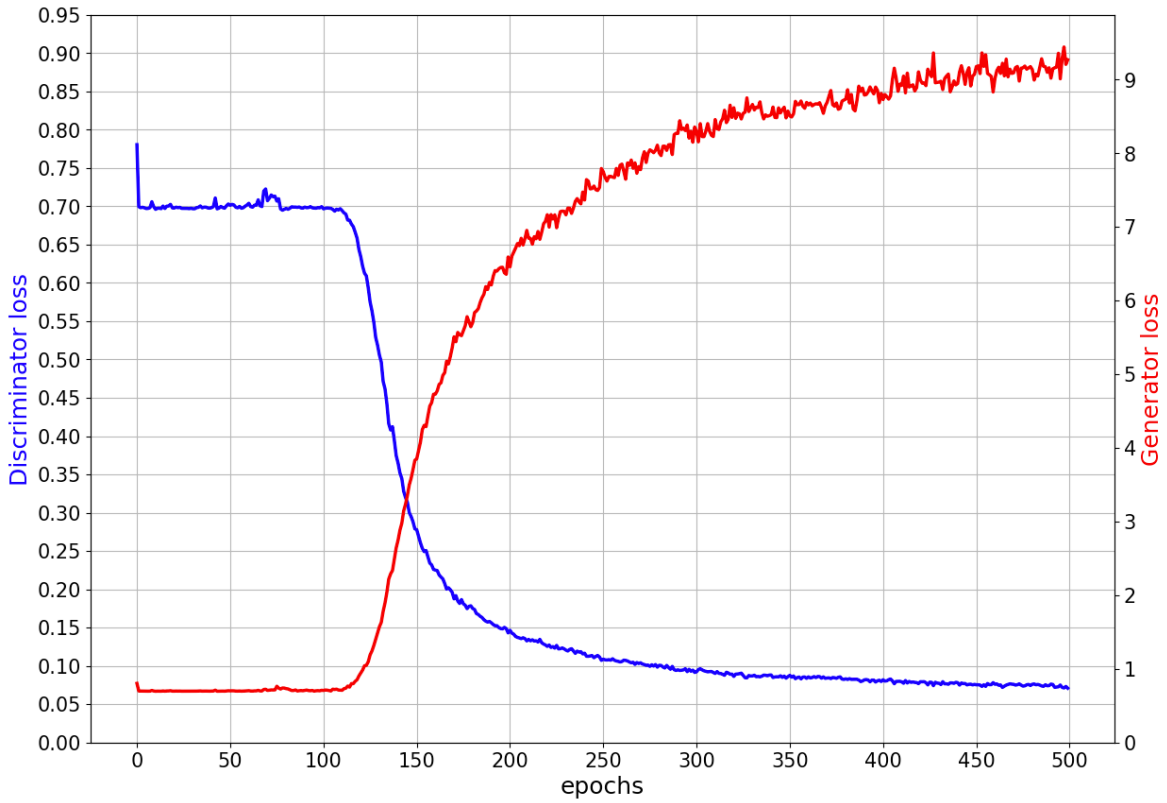

Figure 5.4 Generator and discriminator loss curves after training on 1000 users

For the training process, the loss described in Section 2.2 was utilised along with the reconstruction loss between the generated output and the true keystroke values, implemented as a Mean Squared Error (MSE) function. This approach allows for a more stable training and shows improvements in the results compared to the network trained with regular GAN loss [28]. This application also implies a new hyperparameter λ, that controls the influence of the reconstruction loss, which in our work was set to 5.

The network was trained for 500 epochs and evaluated on a validation dataset every 25 epochs. As an optimizer, we used the Adam algorithm [29], with learning rates $\alpha_G = 3 \cdot 10^{-4}$ for the generator and $\alpha_D = 4 \cdot 10^{-4}$ for the discriminator, and $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We also used a batch size of 128 samples. The models and the training loop were implemented using Python deep learning library PyTorch v2.1.0.

The loss curves of the discussed cGAN are depicted in Figure 5.4. The graph shows that the discriminator loss decreases as the epochs progress, while the generator loss exhibits the opposite pattern. This observation is expected because the training process is primarily influenced by the discriminator. When the discriminator becomes more proficient in distinguishing between real and fake samples, it leads to a higher loss for the generator. Consequently, this situation yields better training progress.

Figure 5.5 illustrates the reconstruction loss curve on the left-hand side and the real / false accuracy evaluation curves on the right. Speaking of the former, we can observe the growth for the first 100 epochs, followed by a slight decrease until the end of the training process, where the loss stabilises at around 0.001.

As mentioned, the model was evaluated every 25 epochs, which included checking how well the model can distinguish between real and generated samples. These accuracies are depicted on the right graph of Figure 5.5. We can see that in the beginning, models have been overtaking each other. From the fourth evaluation onwards, both accuracies started to stabilise and finally reached 100% and 95% for real and fake samples in the final evaluation loop.

The results of the model after the training process were: 0.00083 for the reconstruction loss, 100% real accuracy and 94.6% fake accuracy.

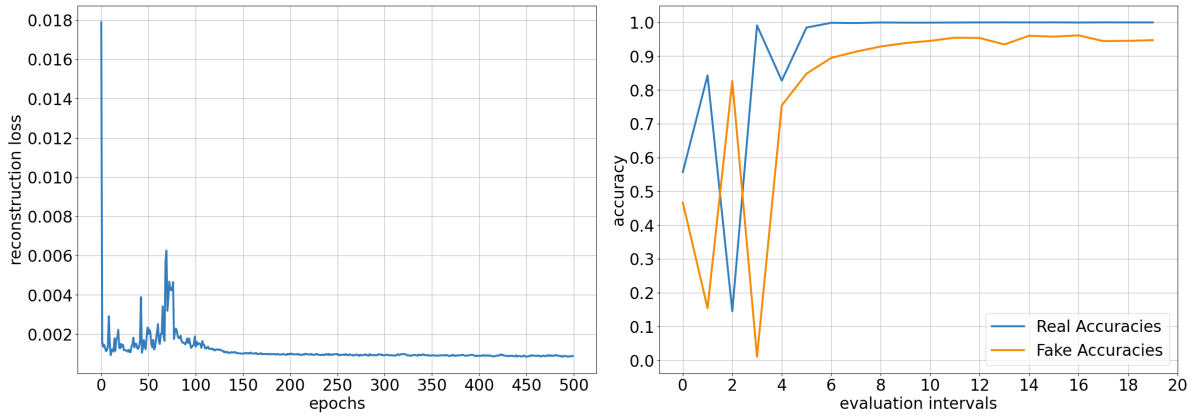

Figure 5.5 Reconstruction loss curve (left) and real/fake accuracy curves (right) after training on 1000 users

5.3.1.2 FINE-TUNING

In this stage, the general model described in the previous section will be fine-tuned separately for different users. As a result, each user will have its very own model capable of producing keystrokes that are similar to those from that user. The training process and the architectures of the generator and the discriminator were identical, as well as the data was pre-processed in the same way.

Since the patterns in keystrokes differ from user to user, it is complicated to come up with a universal solution to ensure a good training process using one set of hyperparameters. For this purpose, we employed the mechanism of finding the most suitable learning rates for each single-user dataset prior to training itself. The interval to find the best learning rate was set to $[7 \cdot 10^{-7}; 7 \cdot 10^{-5}]$. The model was trained for 20 epochs for each one of 50 values evenly spaced over the interval. The best learning rate was chosen based on the largest difference in the reconstruction losses between the 1st and the 20th epoch.

Since the main goal of this stage was to replicate the keystrokes as accurately as possible, we increased the reconstruction loss lambda to 50, so it has more influence on the general loss of the models. The rest of the hyperparameters remained the same. The models were trained for 500 epochs for each user.

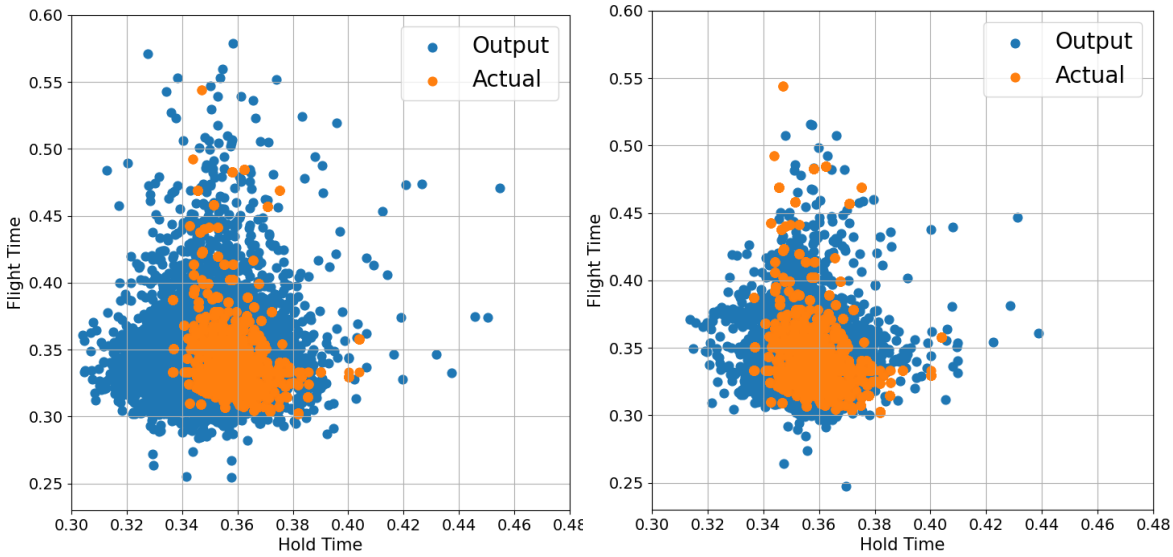

Figure 5.6 Hold Time vs Flight Time before (left) and after (right) the fine-tuning process for a randomly selected user

5.3.2 AUTHENTICATION NETWORK

The authentication network will be used to decide whether the generated keystrokes and the real-time keystrokes belong to the same user or not. We will use the architecture described in [19] (later referred to as TypeNet), which is shown in Figure 5.6. The model consists of two LSTM layers with 128 units each, which allow to capture a long "context" of keystrokes, with Batch Normalizations [30] and Dropouts. Each LSTM layer uses recurrent dropout with a value of 0.2, which is applied after each timestep taken by the cell. In addition, a regular dropout at 0.5 is applied directly between the two layers.

In the present model, both Hold Time and Flight Time will undergo the same pre-processing steps as previously applied in the cGAN. However, a difference from the previous approach lies in the utilization of clicked keys. Rather than serving as mere conditions, they will also be considered features. As previously stated, the keycodes are denoted as integers ranging from 0 to 255. In this context, these keycodes will undergo normalization and thus be transformed to fall within the interval [0, 1]. Next, the normalized keycodes will be concatenated with the initial features. Consequently, a singular input will adhere to the following format: [Normalized keycode, Hold Time, Flight Time].

The authors of [19] proposed to train the network to make it possible to calculate the distance between two provided keystrokes, and then use the threshold τ to make a decision on whether the keystrokes came from one person. The creators compared several ways for the

training, but the best one turned out to be Triplet Loss, which will be used for the purposes of this work. This approach requires three inputs: anchor (A), positive (P), and negative (N) data, which are passed through the same model one by one. Anchor and Positive are the different samples from one class of data, while Negative is sampled from any other class.

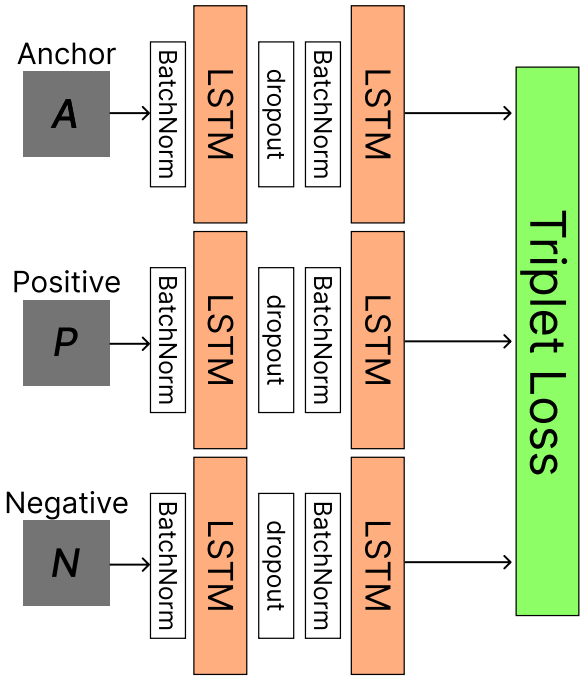

Figure 5.7 Authentication network architecture. All samples (Anchor, Negative and Positive) are passed through the very same model, which consists of two LSTM layers with Batch Normalization [30] and Dropouts

As mentioned, during the training, the system receives three inputs that are passed to the very same architecture sequentially. After that, the L2-squared Euclidean distances (5.1) are calculated between Anchor and Positive and Anchor and Negative, which are later used to determine the triplet loss.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{5.1}$$

The formula for Triplet Loss (5.2) also includes the margin α between positive and negative pairs. Following [19], in this work we set it to 1.5.

$$\mathcal{L}_{TL} = max\{0, d^2({X_A}^i, {X_P}^i) - d^2({X_A}^i, {X_N}^i) + \alpha\} \tag{5.2}$$

During the training process, the model learns to maximise the distance between Anchor and Negative (AN) while keeping the distance between Anchor and Positive (AP) as small as possible. The bigger the difference between AN and AP, the better the network is able to distinguish Negative and Positive samples. During the inference, it takes two samples and outputs a single distance between them. Then, the threshold τ is used to determine whether they belong to the same class.

The network was trained for 100 epochs on the 500-user dataset. We used the Adam optimizer with a learning rate $\alpha = 10^{-4}$, betas $\beta_1 = 0.9, \beta_2 = 0.999$ and the batch size of 128 samples. The training and validation loss curves after 100 epochs presented in Figure 5.7.

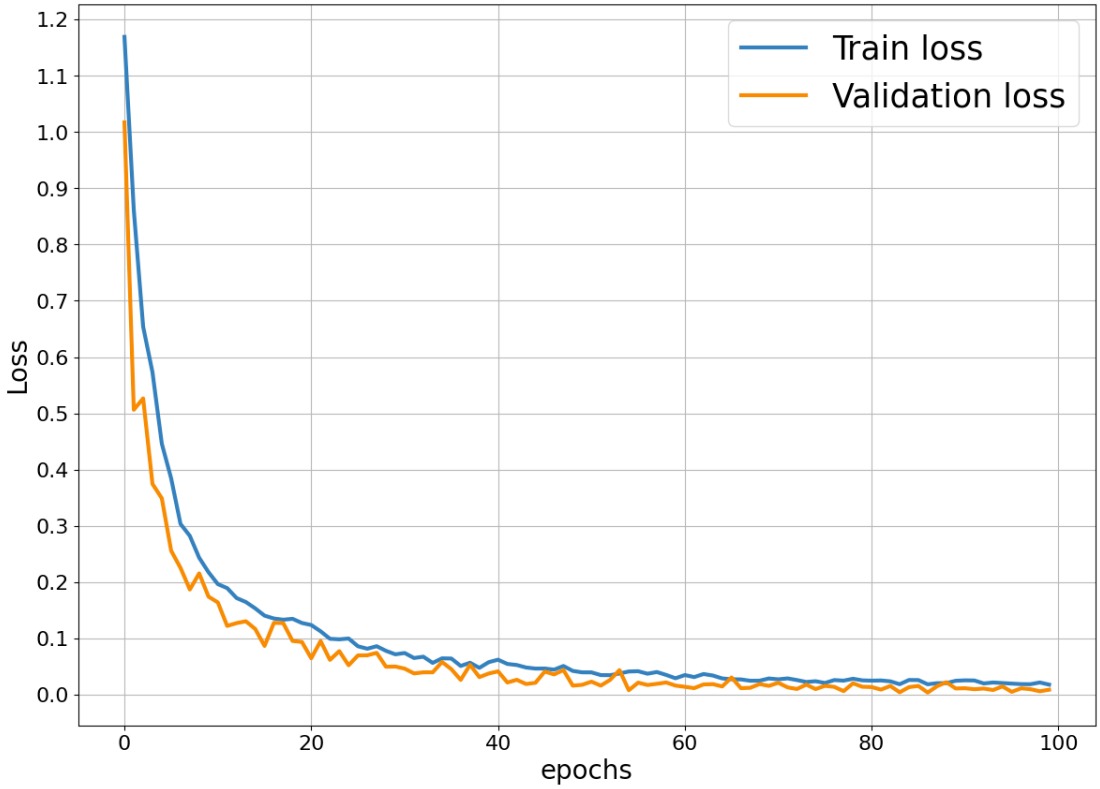

Figure 5.8 Training and validation loss curves of the Authentication Network after 100 epochs

Figure 5.8 illustrates the training and validation differences between the Anchor-Negative and Anchor-Positive (ANP) distances on the left, and the AN and AP distances separately on the right-hand side. As can be observed, both graphs show the desired behaviour. The differences between the trained and validation ANP have been increasing throughout the training process, which was caused by the growth of the AN and the stability of the AP distance.

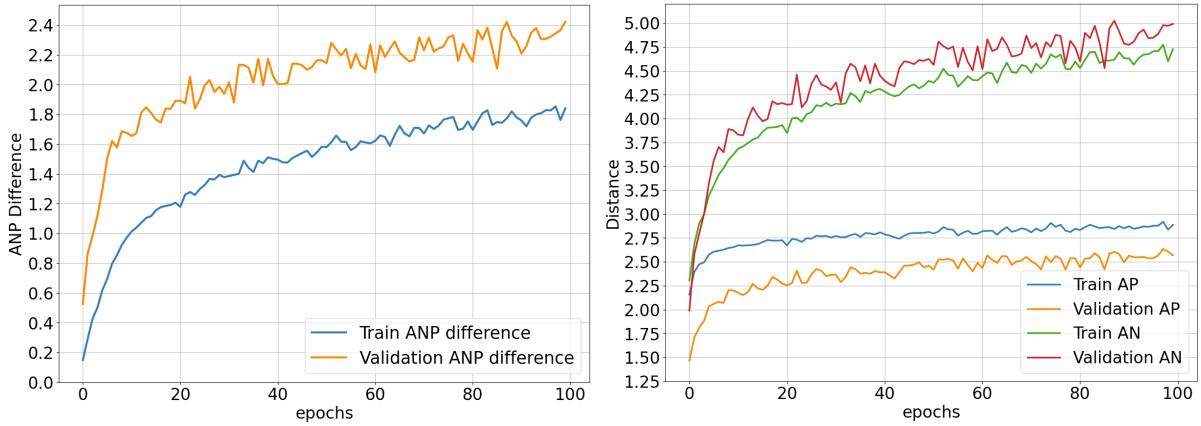

Figure 5.9 Differences in training and validation between Anchor-Negative and Anchor-Positive (ANP) distances (left) and train and validation Anchor-Positive (AP) and Anchor-Negative (AN) distances (right) after 100 epochs

In security systems, the False Acceptance Rate (FAR) refers to the ratio of the illegitimate users that were incorrectly marked as the legitimate ones and granted access to the system, while the False Rejection Rate (FRR) refers to the ratio of the legitimate users that were incorrectly rejected and who have been denied access. Generally, those metrics are used to find the desired sensitivity of the security system by selecting a point called the Equal Error Rate (EER), in which both FAR and FRR are at their lowest. It is worth mentioning that it is a common practice to set a sensitivity as a value that corresponds to the EER, however, it might depend on the needs and the requirements of the system, since some of them might sacrifice the user's convenience in favour of a better protection (FRR will increase, while FAR will decrease), or vice versa.

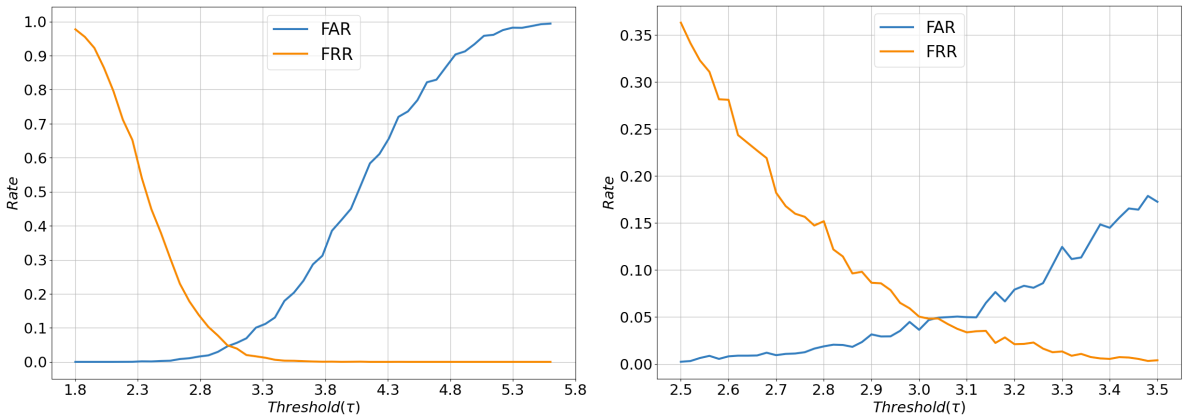

Figure 5.10 Threshold vs False Acceptance Rate (FAR) and False Rejection Rate (FRR) for TypeNet model for threshold values of [1.8; 5.8] (left) and [2.5; 3.5] (right)

Figure 5.9 depicts FAR and FRR for different values of threshold τ for the TypeNet. The threshold values on the graph on the left-hand side were selected based on the values for the AP and AN distance on the test data – 1.8 was the smallest AP distance, while 5.8 was the largest AN distance. As can be seen in the graph, the EER point falls within the interval [2.8; 3.3]. For that reason, the graph on the right side gives a closer look. From there we can see that the EER point corresponds to the threshold value of 3.04, where both FAR and FRR are equal to 5%. It can be also observed that throughout the interval from 3.02 to 3.1 FAR is stable on that value. Because of that, we can look for a lower value of FRR, which would allow us to keep the general error at the lowest level. From the graph we can see that the threshold of 3.1 yields the most optimal trade-off between FAR and FRR, so we will use this value as the threshold for our model.

5.3.3 BASELINE NETWORK

As previously mentioned, in this work we will also develop a user-dependent binary classification model to serve as a baseline for further comparisons, which will take one sample as input and predict whether the keystrokes belong to that specific user or not. The architecture of this network will resemble that of TypeNet, with the addition of two Linear layers and a dropout between these layers with the rate of 0.5 after a final LSTM block. The pre-processing steps applied to the data will remain consistent. Training the model will involve using keystroke data obtained from a specific user, designated as positive instances labelled as "1", alongside data sampled from 10 other randomly selected users, labelled as negative instances denoted as "0". The model will be trained for 150 epochs, with learning rate $\alpha = 7 \cdot 10^{-4}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the Adam optimizer.

Figure 5.11 illustrates accuracy and loss curves for the baseline network trained on the data from a randomly selected user. Final accuracy obtained on the test set was 97.65%.

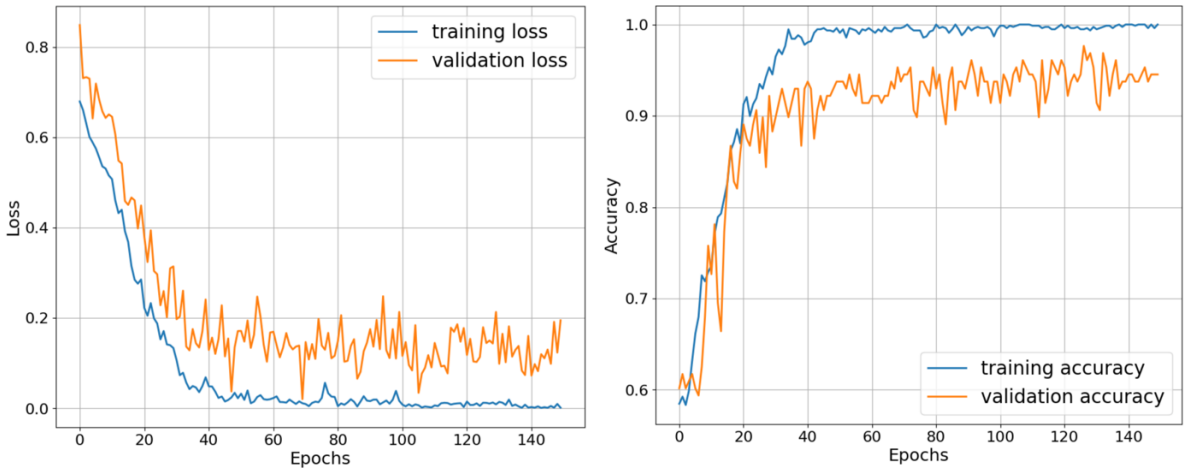

Figure 5.11 Training and validation loss (left) and accuracy (right) curves for a baseline network trained on the data from a randomly selected user

5.4 OTHER EXPERIMENTS

This section will mention other methods we tried to build the cGAN described in Section 5.3.1.

We experimented with utilising the Attention blocks described in [31]. Generally, these blocks help to revisit the provided context and build the attention matrix that for each token in a sequence contains the information about the importance of each other token to the current one. For example, instead of concatenating the latent space and the embeddings at the beginning of the generator and the discriminator, the initial idea was to employ a Cross-Attention block that creates associations and dependencies between two separate contexts. Additionally, we tried to replace the LSTM layers with the Self-Attention blocks that should have allowed the context to build associations with itself (thus, self). Both approaches showed no acceptable results, an unstable training process, and no performance gain whatsoever.

6 RESULTS

This section will describe the conducted tests and present the results obtained. First, we will look at synthetic tests, that were performed on the data used to train the models. Next, the results of the real-time tests will be presented.

In our work, the task can be defined as a binary classification that uses the following values to define the performance of the model: True Positives (TP) – positive instances that were correctly predicted as positives, False Positives (FP) – negative instances that were incorrectly predicted as positives, False Negatives (FN) – positive instances that were incorrectly predicted as negatives, and True Negatives (TN) – negative instances that were correctly predicted as negatives. The evaluation metrics for the subsequent tests are as follows:

- Precision (accuracy)

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

- F1-Score

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \tag{6.2}$$

6.1 SYNTHETIC TESTING

For testing purposes, we individually fine-tuned the cGAN for 20 randomly selected users using the approach described in Section 5.3.1.2 and using their keystroke data. Here, three different tests that will assess the performance of TypeNet and the generators were employed:

1. Real vs. Fake: real keystrokes from the sentences from different users will be compared to the fake ones, generated by the models that **correspond to these users**. Here, the goal is to check whether the authentication model is capable of finding the similarities between real user keystrokes and generated user keystrokes.
2. Real other vs. Fake: real keystrokes from the sentences from different users will be compared to the fake ones, generated by the model for **one chosen user**. Here, the goal is to check whether the authentication model is capable of finding the differences between real keystrokes from other users and generated keystrokes for one selected user.
3. Fake vs. Fake: generated keystrokes created by the model for **one chosen user** will be compared to the generated keystrokes created by the model for **another chosen user**, conditioned on the same keystrokes. Here, the goal is to check whether the authentication model is capable of finding the differences between the generated keystrokes for two different users.

Table 6.1 presents the precision and F1 scores obtained from three different tests. The findings from the "Real vs. Fake test" indicate that the authentication model possesses the capability to detect similarities between two samples, suggesting that the generator model effectively replicates a specific user's keystrokes, which can be further used as a ground truth in a real-time authentication process.

The results of the “Real other vs. Fake” test show lower values for accuracy and F1-score compared to the first test. These imply that the authentication model performs worse in terms of identifying the unauthorized users, despite the generator’s proficiency in replicating the keystrokes of the legitimate user.

Finally, the results for "Fake vs. fake" demonstrate that the authentication model faces some challenges in distinguishing between two generated samples. Although this scenario might be considered implausible, since we would always have one real (collected) and one generated sample for the comparison, it is interesting to note that the authentication model sometimes finds similarities between two mimicked examples.

Table 6.1 Accuracies and F1 Scores for "Real vs. Fake", "Real other vs. Fake", and "Fake vs. Fake" test

Test	Accuracy	F1-Score
Real vs Fake	85.87%	92.40%
Real other vs Fake	76.38%	86.63%
Fake vs Fake	73.39%	84.65%

6.2 REAL-LIFE EXAMPLE

This section will present the results for situations that simulate real-life usage of the proposed system. Here, the goal is to check whether the legitimate user is identified correctly regardless of the typed symbols and the keyboard used for typing.

For this part, the keystrokes of the author (legitimate) and another user (illegitimate) were collected. To make a fair comparison, both participants were chosen based on the same area of activity they perform on a daily basis, software development. Additionally, we used 3 different keyboards for the tests:

- Keyboard 1: Apple MacBook Pro 2021 keyboard with the scissor mechanism and a key travel of 1mm [32].
- Keyboard 2: Apple Magic Keyboard 2017 with the scissor mechanism and a key travel of 1.2mm [33].
- Keyboard 3: Keychron K2 mechanical keyboard with a key travel of 4mm [34].

The generator model was fine-tuned using the keystrokes collected from the legitimate user during typing the sentences described in Section 5.1 (referred to as Same Input) on Keyboard 1. Additionally, we trained the baseline network described in Section 5.3.3 using the user data as positive instances and the samples randomly selected from 10 users from the Aalto University dataset as negative ones.

Furthermore, to represent an arbitrary text that could be typed by the user, we used 15 randomly generated sentences from [35] (referred to as Random Input). Both sets of sentences were typed by the legitimate and illegitimate user on 3 different keyboards. Then, we compared the data collected in real time, both for training and random texts, with the samples generated by the model for the legitimate user. Finally, we made predictions for the real-time keystrokes using the baseline network.

Table 6.2 shows the accuracies and F1 scores of the TypeNet tests. Generally, the system works well in all cases. However, on average, it performs better in illegitimate user detection than in recognising a legitimate one. Additionally, we can notice insignificant differences between the metrics while typing training and random sentences, which confirms that the generator model replicates an entire user's keystroke distribution well. However, clear dissimilarities can be seen in the results for the different keyboards used. It means that although some patterns are preserved, user keystrokes are strongly dependent on the hardware used for typing.

Table 6.2 Accuracies and F1-Scores of the TypeNet model for a legitimate and illegitimate user while typing sentences used for training (Same Input) and randomly selected ones (Random Input) on 3 different keyboards

Keyboard	Legitimate user				Illegitimate user			
	Same input		Random input		Same input		Random input	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
1	85.79%	92.66%	84.11%	90.12%	96.88%	98.10%	96.09%	98.00%
2	78.04%	87.71%	76.85%	86.22%	97.66%	98.81%	98.74%	99.21%
3	80.31%	86.84%	82.81%	89.56%	98.63%	99.31%	99.22%	99.60%

Table 6.3 shows the results of the tests of the baseline network described in Section 5.3.3. As can be seen, the approach proposed in this work is superior to a simple classifier. The baseline network demonstrates satisfactory performance only for training sentences typed on the same keyboard by the legitimate user. For the majority of other cases, it is evident that the model performs marginally better or even worse than just random guessing.

Table 6.3 Accuracies and F1-Scores of the baseline network for a legitimate and illegitimate user while typing sentences used for training (Same Input) and randomly selected ones (Random Input) on 3 different keyboards

Keyboard	Legitimate user				Illegitimate user (Accuracy / F1)			
	Same input		Random input		Same input		Random input	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
1	83.20%	90.83%	55.46%	71.35%	54.88%	70.87%	60.15%	75.12%
2	68.35%	81.20%	54.68%	70.70%	44.33%	61.43%	46.09%	63.10%
3	70.70%	82.83%	51.56%	68.04%	55.85%	71.67%	59.37%	74.50%

In this part, it is also important to mention the time required for a single pass through the system, starting from the initial data collection to the final prediction. All in all, the time required for a 48-character sentence to go through the entire pipeline that utilizes the proposed approach was 0.13 seconds on the Apple M2 Pro chip:

- Calculate and normalize features – 0.001131 seconds.
- Generator forward pass – 0.03 seconds.
- Authentication network forward pass – 0.1 seconds.

Assuming that all calculations will be done on the server side, the result can be considered acceptable, since it will not affect the average response time much (which lies between 200 milliseconds and 1 second [36]).

The time required for the approach that involves the baseline network is expectedly slightly lower and constitutes to 0.127 seconds.

7 LIMITATIONS AND FURTHER WORK

The training and inference process of Generative Adversarial Networks sometimes might be random and unstable. For that reason, some correctness checks might be added to a fine-tuning process in production to ensure the best possible quality of the model prior to using it as a part of the authentication pipeline. Additionally, the GANs can be continuously fine-tuned throughout the entire lifecycle of the network, which can be done by either prompting the users to retype training sentences each fixed interval of time (i.e., every 6 months), or by using the real-time data, if it is predicted as legitimate.

As with any other biometric, keystroke dynamics can vary as a result of external influence, for example physical injury. Worth mentioning, that this biometric might be also developed by taking trainings and performing typing exercises. Furthermore, increased time spent on typing will enhance proficiency in this skill, suggesting that the dynamics may undergo slight changes over time. Additionally, factors such as posture of the person and location of the keyboard in relation to the person's body could affect the precision and rhythm of the typing. As was observed in Section 6.2, the keyboard itself also has an impact on the accuracy of the predictions, so it needs to be addressed before considering such a system as well.

Moreover, the work focused only on the deep learning approach to the task. Although it might be used as the main driver of the system, some other factors can be considered before making a final decision. For instance, the analysis of the type of errors done by the user might be used here as well, along with the other user-specific features. For example, the typo in the middle of the word might be corrected by clicking "Backspace" multiple times or by retracing the mistake using the left and right arrow keys. The typing language should also be considered, since the users might be faster in their mother tongues than in non-native languages.

Finally, as mentioned in Section 5.1, the ideal number of sign-up sentences should be thoroughly examined, by conducting a survey or a questionnaire to collect and analyze the feedback of potential users.

8 CONCLUSION

This work aimed to propose a generative deep learning approach for continuous authentication based on keystroking dynamics. First, we analysed the Aalto University Dataset [12] and addressed research gaps from the original paper.

Leveraging the capabilities of the Generative Adversarial Network to replicate the patterns found in the user data, we created a system that relies on the comparison of the generated keystrokes with those collected in real time. The paper provided a detailed account of the data preparation, architecture of the Conditional Generative Adversarial Network, and the training process of the model.

For the decision-making network, we adopted the model proposed in [19], which utilizes triplet loss to assess the distance between the provided samples. We demonstrated how to determine an appropriate threshold for this distance, enabling the model to accurately determine whether generated and real-time keystroke originate from the distribution of the same user, achieving an EER of 5%.

Additionally, we presented the results for the synthetic tests that were conducted on the training data, and real-life examples that simulated the use cases of the system. The findings showcased the ability of the generator model to replicate the user keystroke distribution, and that the decision-making network performed well in finding similarities and differences in the generated and real-time keystrokes for legitimate and illegitimate users, respectively. We also evaluated the system performance using 3 keyboards with distinguishing characteristics to assess whether the different devices used for typing had an impact on its functionality. The results indicated moderate differences among the hardware used, suggesting the need for further research and additional techniques to mitigate the disparities and create a fully generalized model.

Finally, we discussed the results of the same tests for a simpler approach that involved only a binary classification network trained independently for a specified user. The results presented a low average accuracy at just 58.71%, compared to the one of the proposed approach at 89.59%, suggesting that this simple approach is ungeneralizable and unsuitable for a CA system.

9 BIBLIOGRAPHY

[1] M. Papathanasaki, L. Maglaras and N. Ayres, “Modern Authentication Methods: A Comprehensive Survey,” *IntechOpen,* 2022.

[2] L. Han, “Password Cracking and Countermeasures in Computer Security: A Survey,” *arXiv,* 2014.

[3] S. Shimbun, “Japan researchers warn of fingerprint theft from 'peace' sign,” *phys.org,* 2017.

[4] S. Tariq, S. Jeon and S. S. Woo, “Am I a Real or Fake Celebrity?,” *arXiv,* 2021.

[5] M. Pisani, “Retinal Recognition: the Ultimate Biometric,” Rootstrap, 22 February 2022. [Online]. Available: https://www.rootstrap.com/blog/retinal-recognition-the-ultimate-biometric. [Accessed 15 April 2023].

[6] R. Grimes, “The many ways to hack 2FA,” in *Network Security*, Amsterdam, Elsevier, 2019, pp. 8-13.

[7] A. S. Gillis, P. Loshin and M. Cobb, “TechTarget,” July 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/biometrics. [Accessed 16 April 2023].

[8] F. Cherifi, B. Hemery, R. Giot, M. Pasquet and C. Rosenberger, “Performance Evaluation Of Behavioral Biometric Systems,” *Hal Science,* 2009.

[9] S. Yoon and A. K. Jain, “Longitudinal study of fingerprint recognition,” *Pnas,* 2015.

[10] D. O. Gorodnichy and M. P. Chumakov, “Analysis of the effect of ageing, age, and other factors on iris recognition performance using NEXUS scores dataset,” in *IET Biometrics*, 2018, pp. 29-39.

[11] R. Moskovitch, C. Feher and A. Messerman, “Identity theft, computers and behavioral biometrics,” in *IEEE International Conference on Intelligence and Security Informatics*, Dallas, 2009.

[12] V. Dhakal, A. M. Feit, P. O. Kristensson and A. Oulasvirta, “Observations on Typing from 136 Million Keystrokes,” *ACM Digital Library,* 2018.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative Adversarial Nets,” *arXiv,* 2014.

[14] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” *arXiv,* 2014.

[15] H. Çeker and S. Upadhyaya, “User Authentication with Keystroke Dynamics in Long-Text Data,” University at Buffalo, Buffalo, NY, 2016.

[16] Z. Fan, “Applying Generative Adversarial Networks for the Generation of Adversarial Attacks Against Continuous Authentication,” KTH Royal Institute of Technology, Stockholm, 2020.

[17] K. Killourhy and R. Maxion, “Keystroke Dynamics - Benchmark Data Set,” Carnegie Mellon University, 2009.

[18] R. B. Mario Frank, E. Ma, I. Martinovic and D. Song, “Touchalytics,” Mario Frank, 2013.

[19] A. Acien, A. Morales, R. Vera-Rodriguez, J. Fierrez and J. V. Monaco, “TypeNet: Scaling up Keystroke Biometrics,” *arXiv,* 2020.

[20] I. Eizaguirre-Peral, L. Segurola-Gil and F. Zola, “Conditional Generative Adversarial Network for keystroke presentation attack,” *arXiv,* 2022.

[21] A. M. Feit, D. Weir and A. Oulasvirta, “How We Type: Movement Strategies and Performance in Everyday Typing,” *ACM Digital Library,* 2016.

[22] D. J. LaBonty, “Relationship of finger size and typing speed and errors,” The University of Montana, 1981.

[23] P. Norvig, “English Letter Frequency Counts: Mayzner Revisited,” 2012. [Online]. Available: http://norvig.com/mayzner.html. [Accessed 5 May 2023].

[24] “Cambridge Dictionary,” [Online]. Available: https://dictionary.cambridge.org/dictionary/english/pangram. [Accessed 14 April 2023].

[25] L. N, “Kaggle,” [Online]. Available: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews. [Accessed 14 April 2023].

[26] J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer Normalization,” *arXiv,* 2016.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research 15,* 2014.

[28] Y. Li, N. Xiao and W. Ouyang, “Improved generative adversarial networks with reconstruction loss,” in *Neurocomputing*, Amsterdam, Elsevier, 2019, pp. 363-372.

[29] D. P. Kingma and J. L. Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION,” *arXiv,* 2017.

[30] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv,* 2015.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention Is All You Need,” *arXiv,* 2017.

[32] M. Wuerthele, “14-inch MacBook Pro with M1 review: Where the 'Pro' starts,” 9 November 2021. [Online]. Available: https://appleinsider.com/articles/21/11/09/new-14-inch-macbook-pro-review-where-the-pro-starts#:~:text=The%20new%20MacBook%20Pro%20has,based%2016%2Dinch%20MacBook%20Pro.. [Accessed 16 June 2023].

[33] A. Laforge, G. Vodden and O. Gariepy, “Apple Magic Keyboard 2017 review,” 7 January 2022. [Online]. Available: https://www.rtings.com/keyboard/reviews/apple/magic-keyboard-2017. [Accessed 16 June 2023].

[34] S. Breton, N. D. Giovanni and a. Y. Khong, “Keychron K2 (Version 2) Review,” 18 January 2022. [Online]. Available: https://www.rtings.com/keyboard/reviews/keychron/k2-version-2. [Accessed 16 June 2023].

[35] “Random Sentence Generator,” [Online]. Available: https://www.thewordfinder.com/random-sentence-generator/. [Accessed 16 June 2023].

[36] “What is Response Time and How to Reduce it?,” Sematext, [Online]. Available: https://sematext.com/glossary/response-time/#:~:text=A%20web%20response%20time%20ranging,won%27t%20notice%20the%20delay.. [Accessed 17 June 2023].

LIST OF FIGURES

LIST OF TABLES

APPENDIX 1 – DIGITAL SUPPLEMENTARY MATERIALS

This thesis comes with supplementary materials uploaded as "W11_251512_W11-BDA-MIN_W11-BDAA-000A-OSMW3.zip". The compressed file consists of:

- "W11_251512_2023_praca_magisterska/pretrained/GAN_generator.pt" – weights of the pretrained GAN generator.
- "W11_251512_2023_praca_magisterska/pretrained/GAN_discriminator.pt" – weights of the pretrained GAN discriminator.
- "W11_251512_2023_praca_magisterska/pretrained/TypeNet.pt" – weights of the pretrained TypeNet model.
- "W11_251512_2023_praca_magisterska/source/BaselineNetwork/model.py" – implementation of baseline network in PyTorch.
- "W11_251512_2023_praca_magisterska/source/BaselineNetwork/train.py" – implementation of training loop for baseline network in PyTorch.
- "W11_251512_2023_praca_magisterska/source/GAN/model.py" – implementation of GAN model in PyTorch.
- "W11_251512_2023_praca_magisterska/source/GAN/train.py" – implementation of training loop for GAN in PyTorch.
- "W11_251512_2023_praca_magisterska/source/TypeNet/model.py" – implementation of TypeNet model in PyTorch.
- "W11_251512_2023_praca_magisterska/source/TypeNet/train.py" – implementation of training loop for TypeNet in PyTorch.
- "W11_251512_2023_praca_magisterska/source/utils.py" – additional helper methods.