VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master's thesis

Machine Learning Techniques for Biometric Authentication Based on Keystroke Dynamics

Mašininio mokymosi metodai biometriniam autentifikavimui, naudojant klavišų paspaudimo dinamikos duomenis

Evelina Monastyrska

Supervisor : Assoc. Prof. Dr. Viktor Medvedev

Reviewer : Prof. Dr. Olga Kurasova

Vilnius
2025

Acknowledgements

I extend my sincere gratitude to my supervisor, Assoc. Prof. Dr. Viktor Medvedev, for their guidance, unwavering support, and insightful feedback throughout the course of this master's thesis. Their expertise and dedication have been invaluable in shaping the direction and outcomes of this research.

I also wish to express my appreciation to all the professors and lecturers at the Faculty of Mathematics and Informatics at Vilnius University who contributed to my academic journey. Their dedication, knowledge, and mentorship have provided me with the foundational skills and inspiration necessary to excel in my studies and research.

Summary

Biometric authentication has emerged as a crucial technology for enhancing digital security, with keystroke dynamics providing a cost-effective and accessible behavioral biometric solution. This thesis evaluates and compares various machine learning approaches for keystroke dynamics-based user authentication, and proposes solutions to improve the accuracy of existing authentication techniques. Using datasets such as CMU and KeyRecs, various classifiers, including decision trees, random forests, k-nearest neighbors, support vector machines, gradient boosting, XGBoost, and convolutional neural networks, are implemented.

In addition, user classification based on time series images is introduced. The transformation of time series data into gramian angular summation field and gramian angular difference field images further enhances the analysis.

The results demonstrated significant variations in classifier performance between datasets. For the CMU dataset, the voting ensemble model achieves the highest accuracy of 96.48%, whereas for the KeyRecs dataset, the convolutional neural network achieves up to 90.45% accuracy with an equal error rate as low as 0.1006. The obtained results indicate that it is meaningful to analyze keystroke dynamics on both numeric features and time series images.

Keywords: Keystroke dynamics, biometric authentication, machine learning, convolutional neural networks, binary classification, equal error rate

Santrauka

Biometrinis autentifikavimas yra svarbi technologija, siekiant sustiprinti skaitmeninį saugumą, o klavišų paspaudimų dinamika yra ekonomiškas ir prieinamas elgsenos biometrikos autentifikavimo būdas. Šiame darbe vertinami ir lyginami įvairūs mašininio mokymosi metodai, skirti biometriniam autentifikavimui naudojant klavišų paspaudimo dinamikos duomenis bei siūlomi sprendimai, kaip pagerinti esamų autentifikavimo metodų tikslumą. Naudojant tokias duomenų bazes kaip CMU ir KeyRecs, buvo įgyvendint įvairūs klasifikatoriai, įskaitant sprendimų medžius, atsitiktinius miškus, k artimiausius kaimynus, atraminių vektorių klasifikatorių, gradientinį auginimą, ekstremalų gradientinį auginimą ir konvoliucinius neuroninius tinklus.

Be to, buvo analizuojama vartotojų klasifikacija pagal laiko eilučių vaizdus. Laiko eilučių duomenų transformavimas į paveikslėlius dar labiau sustiprino analizės galimybes. Rezultatai parodė reikšmingus skirtumus tarp klasifikatorių ir tarp duomenų bazių. Naudojant CMU duomenų bazę, balsavimo ansamblio modelis pasiekė didžiausią 96,48% tikslumą, o KeyRecs duomenų bazėje konvoliucinis neuroninis tinklas pasiekė 90,45% tikslumą, su lygių klaidų verte siekiančia 0,1006. Gauti rezultatai rodo, kad klavišų paspaudimų dinamiką verta analizuoti tiek naudojant skaitinius kintamuosius, tiek pagal laiko eilučių vaizdus.

Raktiniai žodžiai: Klavišų paspaudimų dinamika, biometrinis autentifikavimas, mašininis mokymasis, konvoliuciniai neuroniniai tinklai, dviejų kintamųjų klasifikacija, lygių klaidų vertė

List of Figures

List of Tables

Contents

List of abbreviations

CMU	Carnegie Mellon University
CNN	Convolutional Neural Networks
DT	Decision Tree
EER	Equal Error Rate
FAR	False Acceptance Rate
FRR	False Rejection Rate
GADF	Gramian Angular Difference Field
GASF	Gramian Angular Summation Field
GB	Gradient Boosting
KNN	K-Nearest Neighbors
KPCA	Kernel Principal Component Analysis
MDS	Multidimensional Scaling
RF	Random Forests
SVM	Support Vector Machine
T-SNE	T-distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection
XGB	Extreme Gradient Boosting

Introduction

As technology advances and digital systems become integral in daily life, the need to protect sensitive information has increased. Therefore, user authentication, -i.e, the process of verifying a claimed identity [57], has emerged. Ensuring that only authorized users have access to systems and data is essential to prevent unauthorized access and potential security breaches.

Historically, many people have used easily guessable passwords, such as *password*, *123456* or *qwerty*. Common passwords are particularly vulnerable to brute-force attacks, as they are among the first combinations tested when hash algorithms are employed to systematically attempt all possible inputs in search of a match to a specific hash value.

For such reasons, the industry had to improve hash functions to include extra randomization components against brute-force hash algorithms. For example, salting ensures that even if two users have the same password, their hashes differ because of the unique salt. Salting prevents attackers from using precomputed hash tables to identify passwords.

As digital systems have increasingly relied on passwords for security, both researchers and cybercriminals have developed new methods to exploit them. Consequently, the industry is always seeking to incorporate new ways to safeguard the authentication process. For example, to defend against automated authentication attacks, in the late 1990s, researchers developed strategies to differentiate between humans and computers. These techniques are known as the Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). A CAPTCHA cannot be used to authenticate a user, but it can be used to protect against some automated authentication assaults [44].

Following security guidelines and recommended practices is important when new passwords are created. Unfortunately, when people and companies do not adhere to these recommendations, it can result in password database leaks, demonstrating that passwords alone cannot protect online identities. Since the 2010s, as smartphones became more widely available, alternatives to passwords—such as biometric technologies, two-factor authentication (2FA), and multi-factor authentication (MFA)—have been introduced and become increasingly accessible to the general population, despite often being costly and challenging to implement. Thus, biometric technologies have gained popularity, especially when used together with traditional authentication methods, providing an extra level of security [57].

Biometric authentication can be divided into physiological biometrics and behavioral biometrics. Physiological biometrics refer to a person's physical attributes, whereas behavioral biometrics refers to a person's behavioral attributes. Due to the variability of the human body, mind, and changes over time, behavioral biometrics tend to exhibit greater variability and lower authentication accuracy than physiological biometrics. However, collecting behavioral biometric data does not require specialized equipment, making the process more cost-effective and less complex [58].

In recent years, more focus has been placed on analyzing behavioral authentication, specifically keystroke dynamics, which allows an individual to be recognized in a way that he or she types on a computer keyboard [10]. This paper's main **goal** is to evaluate and compare various machine learning

approaches for keystroke dynamics-based user authentication, and propose solutions to improve the accuracy of existing authentication techniques.

Considering the methods chosen by other authors, the following **objectives** were formulated to achieve the goal:

1. Conduct a comprehensive literature review on keystroke dynamics for biometric user authentication.
2. Identify relevant keystroke dynamics datasets for further analysis.
3. Conduct data preprocessing and exploratory data analysis.
4. Perform analysis of identified machine learning models for classifying genuine and impostors and propose possible solutions to improve the quality of authentication.
5. Evaluate and compare the performance of the models.
6. Draw conclusions and suggest future research directions.

The following research description consists of several sections: ”Literature Review” which provides a detailed overview of previous studies by other authors on user authentication and keystroke dynamics. This section outlines the key methods and strategies commonly used in this field. The ”Methodology” section presents comprehensive theoretical information on the methods, analysis techniques, and other research details employed in the study. The ”Experiments and Results” elaborates on the data sources used, initial data analysis and preprocessing sequence, the steps taken to classify users, including algorithms, models, and their parameters. Additionally, the results obtained from the research are presented. Finally, the ”Conclusion” section summarizes the key findings derived from the experiments and analysis, offering suggestions and recommendations for future work in this area.

1 Literature review

Technological advancements have driven the need to analyze biometric-based user authentication methods, such as keystroke dynamics. Two main types of keystroke dynamics consist of fixed and free text.

1.1 Fixed-text keystroke dynamics

In 2009, an article [57] presented a comprehensive survey of existing keystroke dynamics methods, metrics, and various approaches. The focus was primarily on two main approaches for identity verification: statistical techniques and neural network techniques, or a combination of the two.

Statistical methods for keystroke dynamic analysis involve comparing a reference (the user's typical typing behavior collected during enrollment) and test sets (the typing data collected during authentication) of a user's typing characteristics. The distance between these two sets is then calculated, and a threshold is established. If the distance falls below the threshold, the individual is identified as the legitimate user; if it exceeds the threshold, the individual is recognized as an intruder.

The second approach, which uses neural networks, involves building a predictive model from historical data and then using this model to predict or classify new observations. One of the key advantages of neural networks over statistical methods is their ability to capture nonlinear relationships between typing patterns.

Research from 2012, [12] explored various approaches for analyzing keystroke dynamics data, including data acquisition methods and the performance of different techniques used by researchers on standard computer keyboards. Beyond the previously mentioned statistical and neural network methods, the authors discussed pattern recognition and learning-based algorithms. These include both simple machine learning techniques, such as nearest neighbor algorithms and clustering, and more complex approaches, such as data mining, Bayes classifiers, Fisher's linear discriminant (FLD), SVM, and graph theory. One of the significant advantages of using probabilistic learning algorithms, as highlighted by the authors, is their ability to provide a confidence value associated with the decisions made. These algorithms can also mitigate the problem of error propagation by disregarding outputs with low confidence values.

Another approach discussed in the article involves the use of search heuristics and the combination of algorithms. For example, search heuristics such as genetic algorithms, which are part of evolutionary algorithms, are employed to find optimal solutions. Ant colony optimization (ACO) is an example of a method that incorporates genetic algorithms.

Several key factors affecting the performance of the mentioned models were identified in the article. Shorter and simpler passwords are more vulnerable to impersonation, and keystroke patterns can vary significantly between structured and unstructured text. Passwords with special characters are more distinct than those with regular text, and non-English words are generally identified more accurately than English words are. Longer passwords with mixed alphanumeric characters also improve identification accuracy. The cost of enrollment and authentication can vary, but advanced

resampling techniques can drastically reduce computational costs. Research has also discussed the impact of emotional state on typing speed, with negative emotions reducing speed significantly and positive emotions increasing it. Health, the typing environment, and device type also affect accuracy.

The author [20] proposed a method that focuses exclusively on latency and hold time for keystroke dynamics analysis for fixed text. The extracted features are arranged in both sequential and batch modes. The sequential mode extracts features after each keystroke, whereas the batch mode processes them after the entire string is typed, which results in better performance for longer input strings. The authors utilized classification techniques, specifically Gaussian mixture models (GMMs) and NNs, to analyze these timing features. The GMM is trained using only positive (genuine subjects) data, while NN uses both positive and negative (impostors) data for training. The GMM classifier achieves up to 90% accuracy, while the NN classifier achieves up to 99% accuracy. The results show that batch mode is more effective than sequential mode and that using both the username and password as input strings yields better authentication performance than using only one.

The study [11] evaluated the performance of fixed - text short inputs in a recently developed keystroke biometric classification system. A key component of this system is the Pace University classification procedure. The Pace classifier employs a vector-difference authentication model that converts a multiclass problem into a two-class problem. The system was tested on password data from 51 subjects in the CMU dataset. The experiments analyzed two scenarios using different feature sets: one based on the 31 original CMU features and another new set of 75 features designed for this study. The EER in both cases was 8.7%.

In [18], authors explored a wide range of machine learning and deep learning techniques. This study uses the CMU dataset often used in studies involving fixed-text keystroke dynamics analysis. The authors implement a range of models, including traditional machine learning methods such as KNN, RF, SVM, XGB and advanced deep learning techniques like long short-term memory (LSTM), CNN, and multilayer perceptron (MLP), which focus on optimizing the models. The XGB model with data augmentation achieves the highest accuracy of 96.39%, although the exact EER values are not explicitly stated. Without augmentation, XGB achieved an accuracy of 95.42%. MLP - 95.96% accuracy, CNN - 92.57%, and SVM with an accuracy of 88.02%.

In the article [35], the authors present a novel approach to user authentication by focusing on partial passwords. This study aims to increase security by developing a system capable of recognizing users on the basis of their typing patterns, even when only part of the password is entered. To achieve this goal, a unique dataset was created by simulating a bank login scenario with 39 participants, capturing both full and partial password entries. The authors employed a Siamese neural network (SNN) architecture to compare users' typing dynamics and identify them based on the similarity of their keystroke patterns. The SNN-based approach was evaluated against traditional methods, including the scaled Manhattan distance and Mahalanobis distance. The results showed that the SNN outperformed these classical techniques, particularly in scenarios with limited enrollment samples. For full password authentication, the SNN classifier achieved an EER of 0.27, while for partial password authentication, it achieved an EER of 0.3. Additionally, after 12 previous logins, the system reached an accuracy of 89% for full passwords and 73% for partial passwords.

The analysis in [46] utilized the KNN algorithm with Manhattan distance, combined with T-SNE for dimensionality reduction, to visualize distinct data groups. The keystroke dynamics authentication (KDA) algorithm was employed to classify password entries as belonging to a genuine user or an impostor by correlating the entry with both defense and attack datasets and comparing these correlations to a predefined threshold. The study revealed that typing speed was positively correlated with authentication accuracy. Additionally, the password length enhanced the performance of the KNN method but had a negative effect on the distance threshold method. However, increased password complexity adversely affects both approaches.

The research paper [17] presents a novel approach to user authentication by leveraging deep learning techniques, specifically transforming keystroke dynamics (numerical data) into images. This transformation enables CNNs to extract features effectively and recognize patterns. The methodology employs an SNN architecture to compare features of newly entered passwords with those of previously stored passwords, facilitating accurate user authentication. The study achieved competitive results, with an EER as low as 4.545% and accuracy rates reaching up to 98.9%. These findings underscore the potential of visual representations of keystroke data in strengthening user authentication systems, especially in the context of evolving cyber threats.

Alternative approaches for analyzing keystroke dynamic data using images have also been explored. Article [6] introduced the concept of keystroke barcodes, which involve converting habitual biometric data into compact, storable barcode representations. One-class SVMs were employed as the primary classifiers for training and testing these barcodes. This method achieved promising results, with a low EER of 1.83%, demonstrating its effectiveness for keystroke dynamics-based authentication.

1.2 Free-text keystroke dynamics

[3] presented a free-text keystroke biometric system called TypeNet, which is based on a recurrent neural network (RNN) architecture trained with different learning strategies and evaluated on four public databases. The model achieves state-of-the-art performance with an EER of 2.2% on desktop keyboards and 9.2% on mobile keyboards. TypeNet is scalable, maintaining low error rates even when tested on up to 100,000 users.

Keystroke dynamics can be utilized not only for user authentication but also for detecting depressive tendencies. In [24], a binary classification task was proposed to identify these tendencies through keystroke dynamics collected during routine smartphone interactions. The participants were categorized into two groups: individuals with depressive tendencies and healthy controls. By analyzing typing patterns such as hold time and flight time, along with other engineered features, the study employed machine learning techniques to extract insights from these digital biomarkers. The objective of this study was to develop a tool for the early detection of depression.

Multiple classifiers, including GB, RF and Neural Network (NN), were trained and evaluated via metrics such as the AUC, sensitivity, specificity, and accuracy. Among these classifiers, the GB classifier with mutual information feature selection achieved the best performance, with an AUC of

0.98 and an accuracy of 95.83%. These results underscore the potential of keystroke dynamics as a real-time diagnostic tool for mental health monitoring.

Keystroke dynamics can also be utilized to determine whether a user is above or below the age of 18. In [16], the authors proposed leveraging typing patterns, such as key hold durations and latency between keystrokes, to analyze these features for continuous authentication and identification. Using a dataset collected from 116 participants—70 adults and 46 children—during online chat sessions, this study explores the possibility of keystroke-based age prediction for applications such as access control in age-restricted environments.

A trust model was implemented to dynamically evaluate classification confidence as more keystrokes were recorded, allowing for real-time user age assessment. The study achieved promising results, with approximately 80% accuracy in authentication after analyzing 180 keystrokes and a 75% true positive rate for identification within just 20 keystrokes. However, the research identified several challenges. A significant device usage discrepancy was noted, as children predominantly used mobile devices while adults utilized physical keyboards, introducing potential bias into the results. Furthermore, the dataset's limited representation of children, all aged approximately 14, restricted the generalization of the findings. The authors recommend that future research address these limitations by incorporating a broader range of age groups and developing methods to mitigate the influence of device variability.

[4] research focused on the detection of free-text keystroke dynamics. The authors proposed an approach that involves approximating missing values by integrating a key mapping technique with neural network analysis. This method not only examines the timing and sequence of each character pressed by the user but also considers pairs of consecutive characters or digraphs. By analyzing the timing between two consecutive keystrokes, the approach gains additional insights into the user's typing pattern, enhancing the detection and authentication process. Unlike fixed-text methods, which require users to type predefined text, the free-text method analyzes typing in a natural setting without requiring specific input. This makes it ideal for continuous, unobtrusive monitoring. The system introduced in the article achieves strong performance, with a FAR of 1.52% and an FRR of 4.82%, with an EER of 2.46% in a heterogeneous environment (53 users). In a more controlled setting (17 users), an FAR of 0% and an FRR of 5.01% were achieved, with an EER of 2.13%.

The study by [40] employed a comprehensive approach to analyze typing patterns and develop an effective user authentication system using keystroke dynamics. Data were collected from 42 participants, who captured a diverse range of typing behaviors through both structured tasks, such as predetermined phrases, and unstructured tasks, including free-form text, to ensure that natural typing patterns were recorded. The data collection occurred in a natural environment, with participants using their own devices, providing authenticity to the dataset.

The authors focused on extracting key features from the typing data, including keystroke latency (the interval between pressing consecutive keys) and keystroke duration (the length of time each key is pressed). These features were instrumental in creating unique typing profiles for each user, capturing their habitual typing rhythms. Statistical analysis played a crucial role in the methodology, with tests like the t-test employed to identify significant differences in keystroke latencies across

sessions for the same user. Probabilistic models were also used to assess the likelihood of a given typing pattern belonging to a specific user, assuming a normal distribution of features in the pattern vector.

To increase user recognition rates, various classification techniques have been explored. The Euclidean distance classifier calculated the distance between an unknown profile and reference profiles in the database, identifying the closest match. Additionally, the weighted probabilistic classifier incorporated weighted scores based on the frequency distributions of individual features, emphasizing the most reliable features for classification. This classifier achieved a correct identification rate of approximately 90%, highlighting its effectiveness compared with other methods.

The reviewed literature on keystroke dynamics offers significant insights into its application for user authentication and behavioral biometrics. Various studies have explored different algorithms, feature extraction techniques, and data collection methods, highlighting the effectiveness of keystroke dynamics in distinguishing between individual users based on their typing patterns.

2 Methodology

This study employs a systematic methodology to explore the application of traditional machine learning classifiers and CNNs for keystroke dynamics-based authentication. The following section outlines the theoretical foundations, data preprocessing strategies, model architectures, and evaluation metrics adopted in this research. By integrating numerical and image-based approaches, this methodology aims to uncover insights that contribute to the advancement of secure and reliable authentication systems.

2.1 Types of user authentication

User authentication methods can be categorized into three types: knowledge-based, object-based or token-based, biometric-based (see Fig. 1).

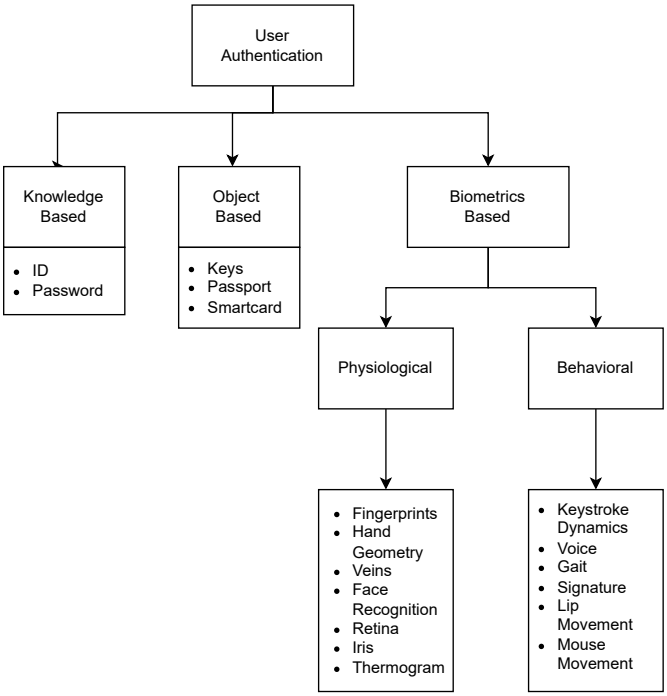


Figure 1: *Classification of user authentication*

2.1.1 Knowledge-based user authentication

The most common type of knowledge-based authentication is personal identification number (PIN) codes or passwords. The knowledge aspect is represented by a secret (known only to the user) [44], which can be an alphanumeric password, PIN or a graphical secret [30].

The main drawback of this type of authentication is its reliance on memorability, as it requires users to remember specific sequences. This often leads to the use of simple passwords, writing passwords down, or reusing the same passwords for different services. Despite these issues, knowledge-based authentication remains widely used because it does not require high development and ad-

ministrative costs associated with tokens or biometric systems. To authenticate a user, only an input device is needed. Additionally, PIN codes are globally accepted due to their extensive adoption in ATMs and the widespread use of mobile phones.

2.1.2 Object- or token-based user authentication

Object-based authentication relies on something one has and is characterized by possession. Traditional keys to doors can be assigned to this category, along with examples such as smart cards, smartphones, and wearable smart devices.

Token-based authentication offers several advantages over knowledge-based methods. While passwords are static, often reused across multiple applications, and remain unchanged unless manually updated, tokens typically generate dynamic passwords valid for a single operation or a limited time, creating an additional layer of security and reducing the risk of unauthorized access.

However, token-based authentication has notable usability limitations. Tokens must be carried, making them prone to being forgotten, lost, or stolen. When smart devices are used, they can be broken or run out of battery, leaving users unable to authenticate them. These devices and their application software are also vulnerable to malware infection, phishing attacks, and reverse engineering. Another example of a token is a smart card, which employs near field communication (NFC) and radio frequency identification (RFID) techniques. While convenient, smart cards are susceptible to various security threats, including impersonation attacks, stolen card attacks, offline password guessing attacks, and server masquerading attacks. Moreover, smart cards face practical limitations, as some computers and devices may not support smart card software.

However, it is very common to combine token-based and knowledge-based authentication methods. For instance, the combination of a bank card with PIN code. Owing to the use of two types of authentication factors, namely, what the user knows (a password) and what the user possesses (e.g., a mobile device or a card), this type of authentication is commonly referred to as two-factor authentication (2FA) [7].

2.1.3 Biometric-based user authentication

Biometric technologies are defined as automated methods of verifying or recognizing the identity of a living person based on physiological or behavioral characteristics.

Physiological characteristics measure the physical parameters of a certain part of the body, such as [7], [60]:

- fingerprints - examining the ridges and valleys of the finger;
- hand geometry - examining the shape and size of a person's hand, length, width, and thickness of the fingers;
- vein checking - looking for distinctive vein patterns under the skin, usually in the hand or finger;
- iris scanning - examining the unique patterns in the colored ring surrounding a person's pupil;

- retinal scanning - analyzing the distinctive pattern of blood vessels in the retina;
- facial recognition - measuring the distance between the person's eyes, the breadth of the nose, the distance of the cheekbones, and other unique features of the subject;
- facial thermogram - measuring the heat patterns emitted by a person's face, which are unique to each individual.

Behavioral characteristics explain how a person uses the body:

- voiceprint - analyzing the unique characteristics of a person's voice;
- gait recognition - analyzing person's walking patterns;
- signature recognition - examining a person's handwritten signature;
- mouse dynamics - identifying how a person uses a computer mouse, focusing on their unique interaction patterns;
- keystroke dynamics - analyzing a person's unique typing patterns.

Physical biometric authentication is referred to as standard biometrics, while non-physical biometric authentication is referred to as cognitive biometrics.

Biometric authentication has several advantages, such as better quality and security, and using a unique feature of the human body for authentication creates an obstacle for an attacker that cannot be predicted. Moreover, it eliminates many of the difficulties of the identification methods that are associated with what a user knows or what a user possesses.

Nevertheless, biometric authentication also has disadvantages. For instance, it is costly—a biometric authentication system typically requires a substantial investment to set up. These costs could be due to both software and hardware requirements and may increase due to ongoing maintenance expenses. A significant drawback of biometric authentication is that once a biometric signature is compromised, it cannot be "reset" or changed, as it is inherently tied to the user's body.

Cognitive biometrics, such as keystroke dynamics, present a more affordable alternative. They do not require additional hardware or specialized user interfaces. This method is also convenient, as it allows seamless authentication without requiring extra steps from the user. Additionally, keystroke dynamics are inherently reliable, and are tied to a user's unique typing behavior, which cannot be lost or deleted.

However, keystroke dynamics have limitations. They generally suffer from low accuracy, as typing patterns can change due to factors such as injury, fatigue, or distraction. Furthermore, a user's typing behavior may evolve over time due to increased proficiency, adaptation to different input devices, or familiarity with typing a particular password.

2.1.4 Keystroke dynamics

Keystroke dynamics is a behavioral biometric trait that aims to recognize individuals based on their unique typing habits. Features such as the velocity of pressing and releasing keys, hand posture during typing, and pressure exerted on keys are often considered when distinguishing between users.

Keystroke authentication operates by creating a template for each user based on their typing patterns during an enrollment period. Once a user is enrolled, their test samples are compared with their stored template, and a matching score is calculated. This matching score is determined using the timing features of keystrokes, allowing the system to identify or verify the user based on their typing behavior.

Researchers use three kinds of keystroke data for authentication [56, 61]:

1. Free-text or dynamic text - permits the user to type freely without any restrictions.

2. Fixed-text, or static text, remains consistent throughout the authentication process, including both template creation and testing.

3. Semi fixed-text, shares some characteristics with free and fixed-text. An example of semi fixed-text - linux commands.

Keystroke dynamics systems can operate in two modes: identification and verification. Identification involves analyzing a person's biometric pattern based on their typing features to determine their identity. The system identifies the user by comparing their keystroke dynamics data to previously collected templates. During the training stage, a biometric template is created for each user. The system matches the test pattern to all available templates, calculating a score or distance that indicates similarity. The user is identified as the individual with the most similar template. To detect impostor patterns (those belonging to unknown individuals), a similarity threshold is applied. If the threshold is not met, the pattern is rejected. This mode allows users to be recognized without providing additional information, relying solely on their typing patterns.

Verification involves confirming a person's claimed identity by comparing their keystroke pattern to their specific template. The system checks whether the test pattern matches the stored template for the claimed identity, ensuring that the user is who they claim to be.

There are several different ways of collecting data and measuring typing patterns for keystroke analysis [9, 57]:

- Static at login - authenticates a typing pattern based on a known keyword, phrase, or predetermined text. The captured typing pattern is compared against a previously recorded pattern stored during system enrollment. This approach checks not only what is being typed, such as a username or password but also, how it is being typed, by measuring features such as the timing between key presses.

- Periodic dynamic - authenticates a user based on typing patterns recorded during a logged session. The captured data in the logged session is then compared to an archived typing pattern to determine the deviations. Periodic authentication occurs either at regular intervals or in

response to suspicious events or triggers. Unlike the static approach, the periodic dynamic does not depend on the entry of a specific text but instead monitors any typing activity.

- Continuous dynamic - authenticates by capturing and analyzing typing patterns throughout the entire duration of a logged session. This approach can detect impostors earlier in the session than can periodic checks, providing continuous monitoring of user behavior.
- Keyword-specific - monitoring extends continuous or periodic monitoring by focusing on metrics related to specific keywords. Additional analysis is applied to sensitive commands or keywords, enabling static analysis for a higher confidence judgment in critical scenarios.
- Application-specific authentication enhances continuous or periodic monitoring by tailoring keystroke pattern analysis to specific applications, enabling context-aware authentication.
- Digraph latency – is a metric that measures the delay between key-up events and subsequent key-down events during typing, such as the sequence of pressing the letters 'T-H.' As illustrated in Fig. 2, latency measurements include the following: the time interval between the press of one key and the press of the next key (DD or down-down), the interval between the release of one key and the press of the next key (UD or up-down) also known as the flight time, and the interval between the release of one key and the release of the next key (UU or up-up). The dwell/hold time is the interval between the press and the release of a single key (DU or down-up).
- Trigraph latency extends the digraph latency metric to consider the timing for three successive keystrokes, such as typing the sequence "T-H-E".
- Keyword latency - evaluates the overall latency for an entire word or analyzes unique combinations of digraphs and trigraphs within a specific word context, providing detailed insights into word-specific typing dynamics.

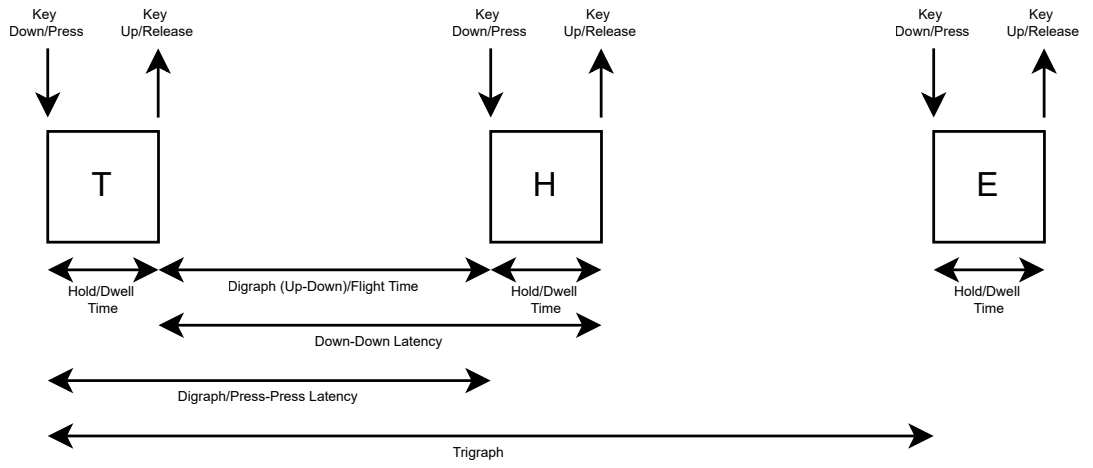


Figure 2: *Features of keystroke dynamics*

2.2 Similarity and difference metrics

This subsection provides a brief overview of the similarities and differences in measurements between the classes analyzed in this research.

2.2.1 Cosine similarity

Cosine similarity is one of the most commonly used similarity measures. It measures the similarity between two vectors in an inner product space by calculating the cosine of the angle between them [26]. The cosine similarity measure for two data points (see Fig. 3) is given by:

$$\text{Cosine similarity}(|A,B|) = \cos(\theta) = \frac{A \cdot B}{||A||\,||B||},$$

where $A \cdot B$ is the dot product of the A and B vectors, with $A \cdot B = \sum_{i=1}^{n} A_i B_i$ and $||A|| = \sqrt{A \cdot A}$ [33].

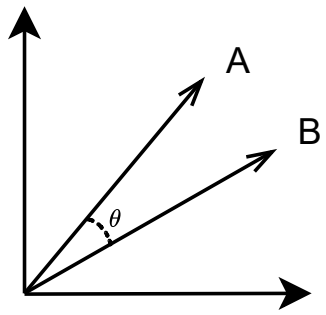


Figure 3: *Cosine similarity*

A perfect correlation will have a score of 1 (0° angle) and no correlation will have a score of 0 (90° angle) [42].

2.2.2 Euclidean distance

The Euclidean distance is the square root of the sum of the squared differences between the corresponding elements of two vectors. The distance between vectors X and Y is defined as follows:

$$d(x,y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}.$$

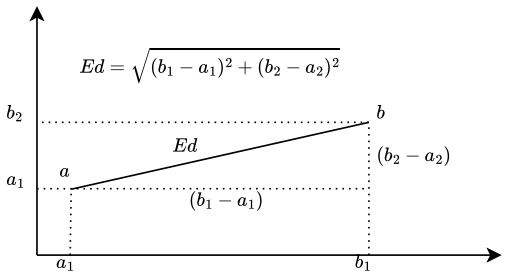


Figure 4: *Euclidean distance*

The Euclidean distance is only appropriate for data measured on the same scale. It is often used to compare profiles of respondents across variables [42]. The generalized concept of the Euclidean distance [45] is shown in Fig. 4. The image and formula indicate that when the distance equals 0, the two vectors are identical.

2.3 Dimensionality reduction methods

Dimensionality reduction and data visualization techniques are important in machine learning, particularly when working with complex datasets. These methods are especially valuable in explanatory analysis, as they provide insights into similarity relationships within multidimensional data [34].

2.3.1 Kernel principal component analysis

PCA reduces the dimensionality of the data by identifying orthogonal linear combinations of the original features that have maximum variance.

Mathematically, PCA finds a way to project the data onto a new subspace. If the data are represented as x_i, the goal is to project them onto a new space $y_i = Ax_i$, where A is a matrix containing the principal components. These principal components are directions in the data where the variance is largest.

To find these principal components, the data is calculated using the covariance matrix S_x. The directions u_k of maximum variance are obtained by solving the equation $S_x u_k = \lambda_k u_k$, where u_k are the eigenvectors of S_x and where λ_k are the eigenvalues. The eigenvectors corresponding to the largest eigenvalues represent the directions with the highest variance, and these are selected as the principal components.

However, the linear PCA approach may not fully capture the complexity of nonlinear structures presented in the data, which may lead to the usage of kernel PCA. By using a kernel function to map data into higher dimensional space, kernel PCA can uncover complex structures within the data that traditional PCA might miss.

A kernel function denoted $\kappa(x_i, x_j)$ calculates the similarity between two data points x_i and x_j in the high-dimensional space. For example the radial basis function kernel [53]:

$$\kappa(x_i, x_j) = exp\Big(- \frac{d(x_i, x_j)^2}{2l^2} \Big),$$

where l is the length scale of the kernel and $d(.,.)$ is the Euclidean distance. The kernel matrix K, which stores the similarities between all pairs of data points, is then computed. PCA is then performed on this kernel matrix instead of the original data matrix. The principal components are derived by solving the eigenvalue equation: $Ka_k = \lambda_k N a_k$ where a_k are the eigenvectors of the kernel matrix and N is the number of data points. Once these eigenvectors are obtained, the new features in the kernel space are calculated as:

$$y_k(x) = \sum_{i=1}^{N} a_{ki} \kappa(x_i, x_j).$$

2.3.2 Uniform manifold approximation and projection

The UMAP technique operates in two main phases: graph construction and low-dimensional embedding optimization [63].

In the graph construction phase, UMAP begins by representing the high-dimensional data as a

graph. Each data point is treated as a node, and edges between nodes indicate the similarity between points. To achieve this, UMAP first identifies the k nearest neighbors for each data point using a chosen distance metric, such as the Euclidean distance. To adjust for how spread out the data are, UMAP calculates two values for each point: one to measure how close its nearest neighbor is, the second value adjusts the distances, ensuring that they are meaningful within the local density of the dataset so that points in sparser regions are not unfairly treated as dissimilar and that points in denser regions are not overly connected. Using these distances, UMAP assigns weights to the connections between points, with closer points receiving higher weights. This step creates a graph where the points are connected on the basis of their similarity.

In the optimization phase, UMAP takes this graph and tries to place the points in a new, low-dimensional space. It uses two opposing forces, attraction and repulsion, to decide where each point will be placed. Points that are close in the original data are pulled closer together in the new space, while points that are far apart are pushed away. This helps preserve the overall structure of the data. These forces are calculated and adjusted repeatedly until the points settle into a position that best represents the original data.

By combining an efficient graph-based representation with a force-directed layout optimization, UMAP achieves a computationally efficient and versatile method for dimensionality reduction. This method is particularly useful for visualizing complex datasets and uncovering meaningful patterns in high-dimensional data.

2.3.3 Metric multidimensional scaling

MDS is a powerful method for analyzing and visualizing dissimilarity data by placing objects in a low-dimensional space [65]. Unlike classical scaling, which assumes direct proportionality between dissimilarities and distances, metric MDS allows for a more flexible relationship. It models the distances between points in the low-dimensional space as a function of the dissimilarities, $f(\delta_{ij})$, which can be linear or nonlinear depending on the application.

The main goal of metric MDS is to minimize the discrepancy between the given dissimilarities and the distances in the reduced space. This is achieved by defining an error function, commonly referred to as ”stress”, which quantifies the difference between these values:

$$\text{Stress} = \sum_{i,j} w_{ij}(d_{ij} - f(\delta_{ij}))^2,$$

where d_{ij} represents the distance between points i and j in the low-dimensional space, and w_{ij} represents weights that can emphasize or de-emphasize certain pairs of points. By minimizing this stress function, metric MDS ensures that the low-dimensional representation preserves the relationships in the original data as accurately as possible.

The optimization process in metric MDS involves iteratively adjusting the coordinates of the points to reduce the stress function. This iterative approach makes metric MDS more computationally intensive than classical scaling but also more versatile. It is particularly valuable when the rela-

tionship between the dissimilarities and distances is complex or when additional flexibility is required to model the data [65].

2.3.4 T-distributed stochastic neighbor embedding

T-SNE is a technique for reducing high-dimensional data into two or three dimensions, primarily for visualization. It aims to preserve the local structure of the data, meaning that points that are close together in the original space remain close in the lower-dimensional space. The method works by defining probabilities in both high-dimensional and low-dimensional spaces and minimizing the difference between them.

In high-dimensional space, t-SNE computes the similarity between points using a Gaussian distribution. For each data point x_i, the probability $p_{j|i}$ of x_j being a neighbor is calculated based on the distance between the two points and a parameter σ_i, which controls the spread of the Gaussian around x_i. The value of σ_i is adjusted to ensure that the distribution has a fixed perplexity, a user-defined parameter that determines the number of effective neighbors. The perplexity is calculated as $2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$, ensuring that the Gaussian adapts to the local density of the data. The probabilities are then symmetrized as $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2N}$, where N is the total number of points.

In the low-dimensional space, t-SNE represents each point x_i as a lower-dimensional point y_i. To compute similarities in this space, t-SNE replaces the Gaussian distribution with a Student's t-distribution with one degree of freedom, which is calculated as $q_{ij} = \frac{(1+\|y_i-y_j\|^2)^{-1}}{\sum_{k\neq l}(1+\|y_k-y_l\|^2)^{-1}}$. This distribution has heavier tails than a Gaussian distribution, which helps alleviate the "crowding problem", where many points are compressed into a small area in the low-dimensional space. This allows better separation of clusters.

To align the high-dimensional and low-dimensional relationships, t-SNE minimizes the difference between the two distributions using the Kullback-Leibler (KL) divergence. The loss function, $\mathcal{L} = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$, ensures that points close in the high-dimensional space remain close in the low-dimensional space, while points far apart are also separated. Optimization is achieved through gradient descent, where the gradient of the loss function determines how the positions of points in the low-dimensional space are updated. If the similarity between two points is greater in the high-dimensional space than in the low-dimensional space, they are pulled closer together. Conversely, if they are overly close in the low-dimensional space, they are pushed apart.

T-SNE typically initializes the low-dimensional points y_i randomly or uses a technique such as PCA. Gradient descent is then performed with momentum to stabilize and accelerate convergence. Hyperparameters such as the learning rate and momentum are dynamically adjusted during the optimization process.

Overall, t-SNE excels at preserving the local structure of data and is widely used for creating visually distinct clusters, making it an effective tool for understanding complex datasets. However, it has limitations in preserving global relationships, meaning that the relative distances between far-apart clusters may not accurately reflect the original data. Despite this, its ability to produce meaningful and interpretable visualizations has made it a favorite choice for exploratory data analysis [63].

2.4 Machine learning models

2.4.1 K-nearest neighbors

The KNN algorithm is an instance-based learning method designed for solving both classification and regression tasks. The algorithm analyzes data objects and identifies the k-nearest neighbors of an unknown feature vector to identify the class [8]. By selecting an appropriate distance function, KNN algorithm assigns a class to the analyzed data point using majority voting. The number of neighbors k is determined by the researcher.

If there are multiple neighbors, the algorithm calculates how many of the nearest neighbors belong to class 0 and how many belong to class 1. Based on this information, the class to which the majority of the nearest neighbors belong is assigned to the analyzed data point [39, 41].

Hyperparameters [51] used in the research can be found in Table 1.

Table 1: *KNN hyperparameters*

Parameter	Model	Default Value	Possible Values	Description
n neighbors	KNN	5	Integer	Number of neighbors to use
Weights	KNN	*uniform*	Uniform - all points in each neighborhood are weighted equally. Distance - closer neighbors of a query point will have greater influence than neighbors which are further away.	Used in prediction
p	KNN	*2*	Float	When *p* = 1, this is equivalent to using manhattan distance (*l1*), and Euclidean distance (*l2*) for *p* = 2. For arbitrary *p*, Minkowski distance (*lp*) is used.

2.4.2 Support vector machine

In the case of a linear classifier, the support hyperplane is constructed to separate two distinct classes: positive and negative. The hyperplane is described by the following formula:

$$\omega^T X + b = 0,$$

where ω is the weight vector of the inputs, X is the input vector, and b is the bias.

During classification, input vector values satisfying $\omega^T X + b > 0$ are assigned to the positive class, while

values satisfying $\omega^T X + b < 0$ are assigned to the negative class. This distinction determines the separation of positive and negative classes by the hyperplane.

The classification rule that determines which class a new input vector X belongs to is defined using the sign function:

$$f(X) = sgn(\omega^T X + b),$$

where sgn(x) is the sign function, which returns:

$$\text{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

At a point X_i, if $\omega^T X_i + b = 1$, then the value:

$$\frac{\omega^T X_i + b}{||\omega||} = \frac{1}{||\omega||}$$

represents the distance from the hyperplane $\omega^T(X - X_i) = 0$ to the origin. The value:

$$\frac{2}{||\omega||} = \frac{2}{\sqrt{\omega^T \omega}}$$

is the distance between the support vectors $\omega^T X + b = \pm 1$ [13, 39]. The hyperparameters [54] of SVM are described in Table 2.

Table 2: *SVM hyperparameters*

Parameter	Model	Default Value	Possible Values	Description
C	SVM	1	Positive float	Regularization parameter. The strenght of the regularization is inversely proportional to C.
Kernel	SVM	*rbf*	Linear $K(a,b) = a^T b$; poly K(a,b) = $(\gamma a^T b + r)^d$; radial basis function, *rbf*) $K(a,b) = exp(-\gamma \mid\mid a - b \mid\mid^2)$.	Specifies the kernel type to be used in the algorithm.

2.4.3 Decision tree

DT is a hierarchical structure consisting of nodes, branches, and leaves. Each node represents an event or condition aimed at classifying or predicting an outcome. A node has branches, which represent classification

rules. This means that each branch reflects a possible choice or decision, splitting into further nodes. Finally, the tree ends with leaves, where the final decision or prediction is made (see Fig. 5) [37, 39].

The classification and regression tree (CART) algorithm is one of many algorithms used to construct DT models. This algorithm initially splits the dataset into two subsets using a feature k and a threshold t_k. The values of k and t_k are chosen to yield the purest subset, thereby minimizing the loss (cost) function:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right},$$

where $G_{left/right}$ measures the impurity of the left/right subset and where $m_{left/right}$ represents the number of events in the left/right subset.

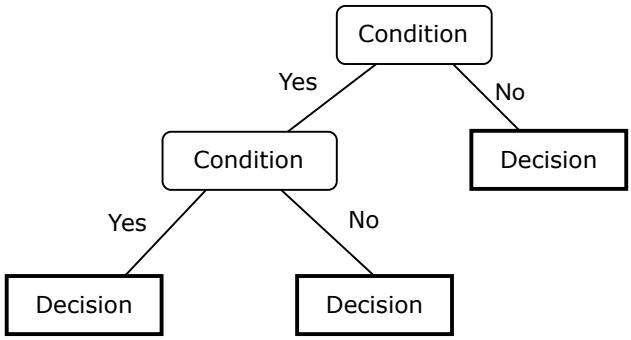


Figure 5: *Decision tree diagram*

Once the algorithm successfully splits the dataset into two subsets, it recursively applies the same logic to each of these subsets, continuing the process until the tree reaches a fixed depth [25, 39].

To improve model performance, hyperparameters are often selected and specified. The hyperparameters [49] applied to DT models in this research are described in Table 3.

Table 3: *DT, RF, GB and XGB hyperparameters*

Parameter	Model	Default value	Possible values	Description
n estimators	XGB, RF, GB	100	Integer	Number of trees in the model
Max depth	XGB, RF, DT	6	Interger	Maximum depth of a tree.
Lambda	XGB	1	[0; ∞]	L2 regularization term on weights.
Min Sample Split	DT	2	Integer or float	The minimum number of samples required to split an internal node.
Learning Rate	GB	0.1	[0; ∞]	Shrinks the contribution of each tree.

2.4.4 Random forest

The RF algorithm belongs to the group of supervised learning algorithms. It utilizes the bootstrap aggregating (bagging) method (see Fig. 6), which creates an ensemble of DTs using random subsets of the data. The results from all DTs are then combined, and the final outcome is determined based on majority voting. As the number of trees increases, the prediction accuracy improves up to a certain point [8, 25, 39].

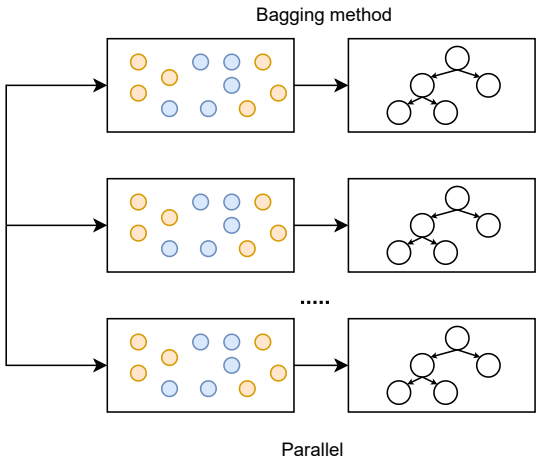


Figure 6: *Principle of the bagging method*

Similar to DT models, adjusting default parameter values is often used to achieve more accurate results. Several hyperparameters [52] of the RF model are described in Table 3.

2.4.5 Gradient boosting

GB is a supervised machine learning algorithm, similar to the RF, that is based on an ensemble of DT, but it operates using the boosting method (see Fig. 7) by sequentially adding predictors to an ensemble, each one correcting its predecessor. The algorithm is based on the idea that the next best model is constructed by learning from the residual errors made by the previous predictor, thereby reducing the prediction error [25, 39].

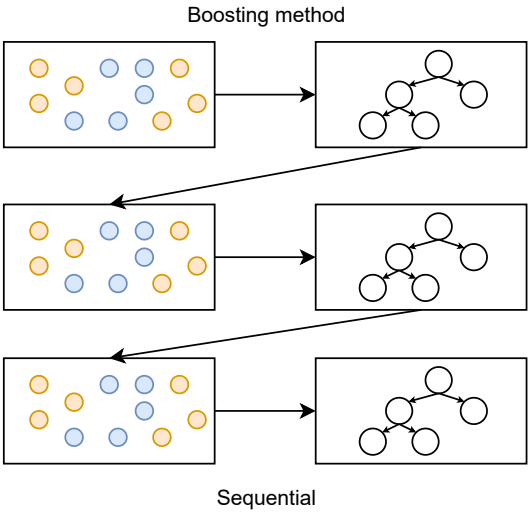


Figure 7: *Principle of the boosting method*

Hyperparameters [50] used in the tuning process described in Table 3.

2.4.6 XGBoost

XGB is a GB method, known for its fast learning speed and high prediction accuracy compared with those of tree-based algorithms.

XGB is an extension of gradient boosting trees. It employs the gradient descent method to update the model with new predictions, aiming to minimize the loss of the added predictions. Additionally, XGB incorporates regularization into its optimization function to reduce overfitting [15, 39, 62]. Several hyperparameters [55] of the model are presented in Table 3.

2.5 Ensemble models

2.5.1 Voting ensembles

Ensemble methods are learning algorithms that combine a group of different classifiers and work as a single classifier. There are two types of voting ensemble methods: hard and soft. The difference between them is that hard voting method is a more straightforward approach based on a majority rule, whereas soft voting involves collecting the predicted probabilities for each class label and predicting the class label with the largest probability: $y^* = argmax_i \sum_{j=1}^{i} \omega_j P_{ij}$ (see the left graph in Fig. 8).

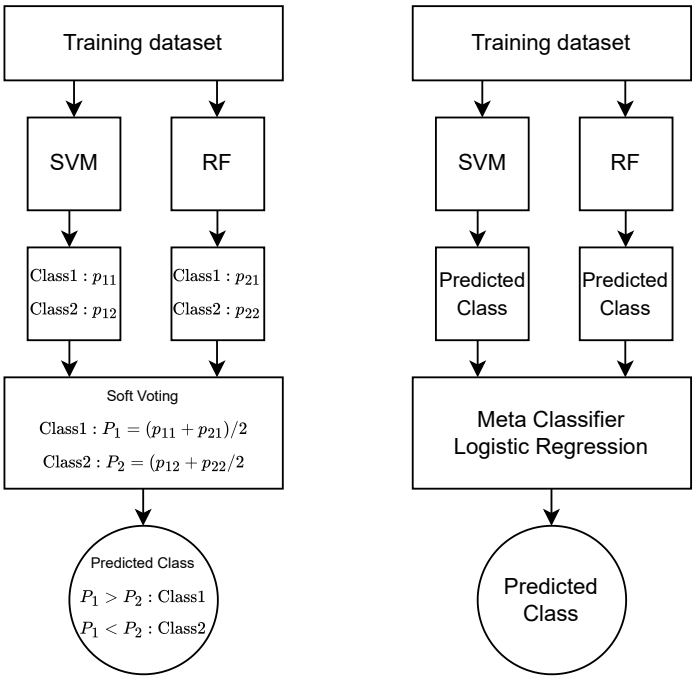


Figure 8: *Ensemble models*

Soft voting needs to store and use all the distribution values, making it more computationally costly for storage. However, in soft voting various methods can be used to calculate the prediction, such as calculating maximum or average probability values [38].

2.5.2 Stacking

The stacking method, also known as stacked generalization, is a model ensembling technique used to combine information from multiple predictive models to generate a new model (meta-model). The architecture of a stacking model involves two or more base models, referred to as a level-0 model, and a meta-model that combines the predictions of the base models, referred to as a level-1 model. In level 0 models (base models), models fit on the training data and their predictions are compiled. However, in the level 1 model (meta-model), the model learns how to combine the base models' predictions best. The outputs from the base models used as inputs to the meta-model may be probability values or class labels in the case of classification [38] (see the right graph in Fig. 8).

2.6 Data transformation

2.6.1 Quantile transformation

Quantile transformation is a non-linear feature scaling technique that maps data to a uniform or normal distribution using the cumulative distribution function (CDF). This transformation mitigates the effects of outliers and enhances model robustness by spreading tightly clustered data and bringing extreme values closer to the rest of the distribution. It is particularly useful in preprocessing for tasks such as classification, as it ensures better feature separability.

Mathematically, the transformation is expressed as:

$$x_{\text{transformed}} = F^{-1}(\phi(x)),$$

where $\phi(x)$ is the empirical CDF of the original feature x, computed as $\phi(x) = \frac{\text{rank}(x)}{N}$, with N being the total number of observations. F^{-1} is the quantile function (inverse CDF) of the target distribution, such as a uniform or Gaussian distribution.

This transformation standardizes features to match the desired distribution, spreading out tightly clustered values and compressing outliers [2, 48].

2.6.2 Gaussian noise

Gaussian noise is a statistical noise commonly modeled with a normal distribution. It is described by the probability density function (PDF) as:

$$P(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}},$$

where z represents the intensity value, μ is the mean (average) value of z and σ^2 is the variance, and σ is the standard deviation of the noise [1].

Gaussian noise can be applied in digital image processing to enhance dataset generalization by introducing variability and simulating real-world conditions.

2.7 Imaging time series data

For some numeric data, the ordering of features can be reversed in a two-dimensional space to explicitly represent the relationships among these features. This approach allows tabular data to be transformed into images, enabling CNNs to extract meaningful patterns from the visual representation [17].

GASF is a method used to convert a time series into an image that captures temporal dependencies in the data. The key idea is to represent the time series in a polar coordinate system, where the value of each time step is transformed into an angle, and the time itself is treated as the radius.

The transformations begin by rescaling the time series $X = \{x_1, x_2, \ldots, x_n\}$ into a normalized range, such as $[-1, 1]$ using the following formula:

$$\tilde{x}_{-1}^i = \frac{(x_i - \max(X) + (x_i - \min(X))}{\max(X) - \min(X)}$$

or $[0, 1]$ using the following formula:

$$\tilde{x}_0^i = \frac{x_i - \min(X)}{\max(X) - \min(X)}.$$

Next, the rescaled time series is converted into a polar coordinate system by encoding the value as the angular cosine and the time stamp as the radius. The angular representation is given by:

$$\phi_i = \arccos(\tilde{x}_i), \quad -1 \leq \tilde{x}_i \leq 1$$

and the radius as $r = \frac{t_i}{N}$.

Using these angular values, the GASF matrix is constructed by calculating the cosine of the summation of angles between each pair of time points:

$$\text{GASF}[i, j] = \cos(\phi_i + \phi_j).$$

In contrast, the GADF matrix captures the sine of the difference of angles between pairs of time points, highlighting dynamic changes:

$$\text{GADF}[i, j] = \sin(\phi_i - \phi_j).$$

Both GASF and GADF preserve the temporal order of the original time series, with time increasing along the diagonal from the top-left to the bottom-right of the resulting matrix. Additionally, GASF with rescaled data in the range $[0, 1]$ provides a bijective mapping, allowing reconstruction of the original time series from the diagonal elements:

$$\tilde{x}_i = \cos\left(\frac{\phi_i}{2}\right), \quad \phi_i \in [0, \pi].$$

These methods effectively transform temporal data into structured images, enabling the application of CNNs and other computer vision techniques for time series analysis [64].

2.8 Convolutional neural network

CNNs are deep learning algorithms designed to process input images and to identify key features in the images. They excel in tasks such as image processing and speech recognition due to their ability to effectively leverage local information through weight sharing.

CNNs are composed primarily of several layers. The input layer accepts the raw image data [56]. The convolutional layer is composed of multiple convolutional kernels whose parameters are optimized through backpropagation. Each kernel is designed to extract specific features, such as horizontal or vertical edges, based on its learned parameters. While individual convolutional layers capture basic low-level features, deeper networks with additional convolutional layers can progressively extract higher-level, more complex features. After the convolutional layer, the pooling layer reduces the spatial dimensions of the feature maps. This is achieved by dividing the feature maps into regions and applying operations such as maximum pooling or average pooling, which generate a more compact representation. This step not only decreases computational complexity but also helps prevent overfitting. The fully connected (dense) layer integrates all the local features extracted by previous layers into global features by flattening them into a one-dimensional vector. It then passes these features through an activation function to calculate the final scores for each class, enabling the model to produce the final classification result.

Overfitting is a common challenge in deep learning, often resulting in models that perform well on training data but poorly on unseen data. Techniques to address overfitting include early stopping, L1 and L2 regularization, and dropout. In this work, dropout is utilized as a preventive measure. During training, dropout randomly disables a proportion of neurons based on a predefined probability, compelling the network to learn more robust and generalized features.

A neural network without nonlinearity behaves as a linear system, which severely limits its ability to solve complex problems. To overcome this limitation, activation functions are used to introduce nonlinearity into the network. The rectified linear unit (ReLU) is one of the most widely used activation functions. It not only introduces nonlinearity but also contributes to mitigating overfitting by enabling sparse activations, thereby improving the efficiency and generalizability of the model [36].

2.9 Evaluation metrics

The effectiveness of a biometric authentication method, specifically its ability to distinguish between legitimate and illegitimate users, is evaluated using various metrics. The following sections explain the key evaluation criteria [22].

2.9.1 False acceptance rate and false rejection rate

FAR represents the probability of an impostor successfully posing as a genuine user and gaining access to a secured system. In statistical terms, this corresponds to a type II error. In contrast, the FRR measures the percentage of valid users who are mistakenly rejected as impostors by the keystroke dynamics-based authentication system, which is referred to as a type I error in statistics [22].

Ideally, both the FAR and FRR should be 0%, as this would ensure a perfect balance between security and convenience. From a security perspective, minimizing type II errors (FAR) is critical to prevent unauthorized access. However, minimizing type I errors (FRR) is equally important to avoid frustrating genuine subjects with incorrect rejections.

A system with 0% FAR is considered to provide maximum security, while a system with 0% FRR offers optimal convenience. In practice, there is often a trade-off between these two metrics. Highly secure systems may inconvenience users by rejecting genuine attempts, whereas overly convenient systems may compromise security. Notably, FAR is also referred to as the false positive rate (FPR), and FRR is equivalent to 1 - true positive rate (TPR) [56].

$$FAR = \frac{\text{Number of false acceptances}}{\text{Total number of impostor attempts}};$$

$$FAR = \frac{\text{Number of false rejections}}{\text{Total number of genuine attempts}}.$$

2.9.2 Receiver operating characteristic (ROC) curve

The performance of a biometric verification system is typically represented by its ROC curve. The ROC curve illustrates the tradeoff between the FAR and the FRR.

As shown in Fig. 9, the ideal classifier is represented by a point at the top-left corner of the ROC plot, with coordinates (0.0, 1.0), indicating that there are no false accepts or false rejects. While achieving this level of performance is impractical for large-scale systems, a classifier closer to the (0.0, 1.0) coordinate is highly desirable. The diagonal line from the top-right corner to the bottom-left corner represents the performance of a random classifier, equivalent to random guessing, which is undesirable. A classifier whose ROC curve falls below this diagonal is often assumed to have incorrect labeling and may be flipped for correction [56].

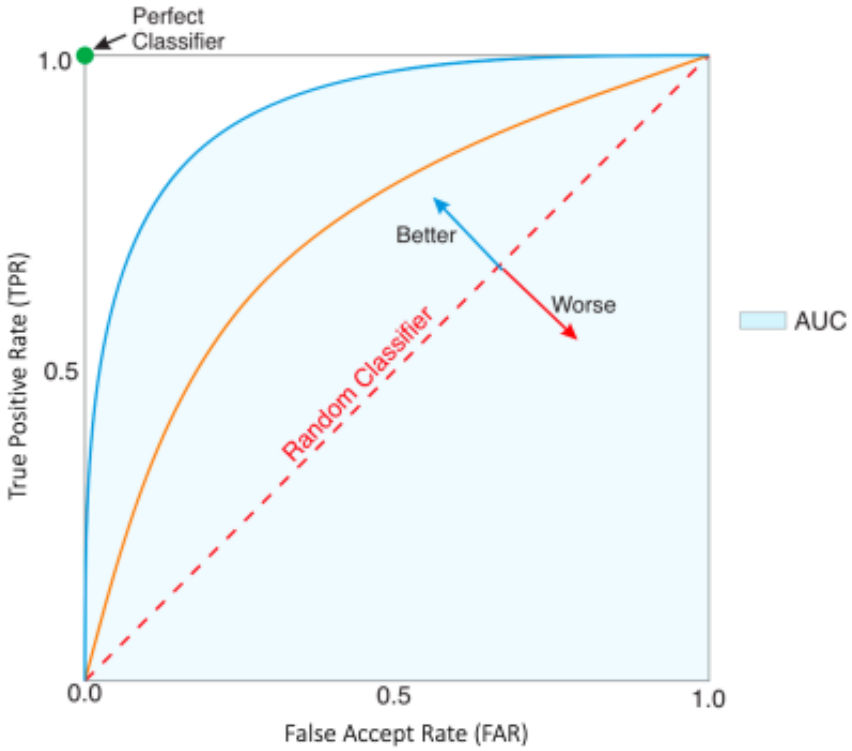


Figure 9: *ROC and AUC for different classifiers [56]*

2.9.3 Equal error rate

EER or the cross-over error rate (CER), is the point where the rates of false acceptance and false rejection are equal. This value indicates the overall accuracy of a biometric system, with lower EER values corresponding to higher accuracy.

Threshold values (on the x-axis) play a critical role in balancing security and convenience. A threshold closer to 0 prioritizes convenience by minimizing FRR but increases FAR, while a threshold closer to 1 enhances security by lowering FAR but raises FRR. The EER represents the threshold that balances these two factors [56].

To compute the EER, genuine users are labeled *0* and impostors are labeled *1*. The prediction scores for both classes are combined, and the ROC curve is used to calculate the FAR and TPR across different threshold levels.

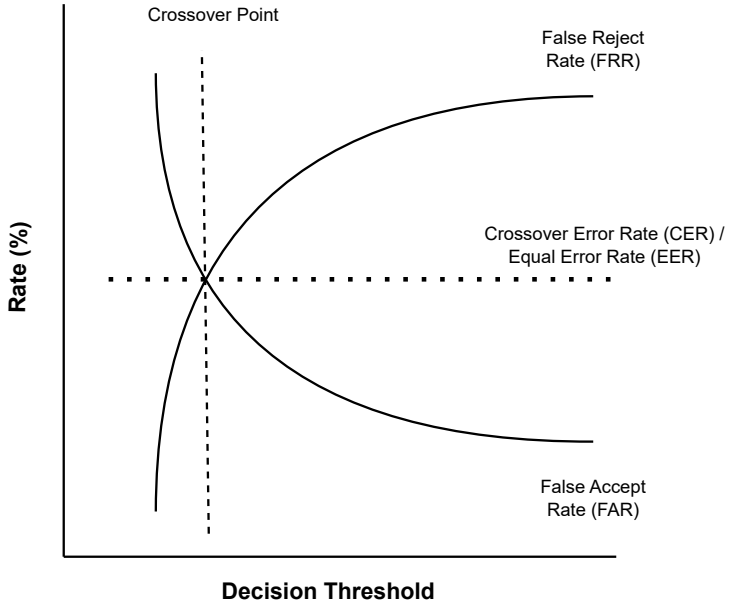


Figure 10: *Relationships between FAR, FRR and ERR*

The EER is identified at the point where FAR and FRR are closest to each other. Since these rates may not intersect exactly, linear interpolation can be used between the two closest points where FRR transitions from being greater than FAR to being smaller. This method provides an accurate estimate of the threshold at which the two rates converge. The resulting EER value offers a single metric that reflects the system's performance by balancing the likelihood of incorrectly accepting impostors and rejecting genuine users [32].

Fig. 10 shows the relationships between FAR, FRR, and EER. A lower EER indicates better performance [5].

2.9.4 Accuracy

Classification metrics are measures used to evaluate the performance of classification algorithms or models [28, 39].

A confusion matrix is an $N \times N$ table used to assess the performance of a classification model. It compares the actual values with the predicted values (see Table 4).

Table 4: *Confusion matrix*

	Positive	Negative
Positive	TP	FN
Negative	FP	TN

TP (True Positive): The model correctly predicts the positive class. TN (True Negative): The model correctly predicts the negative class. FP (False Positive): The model predicts the positive class incorrectly—the true value is negative. FN (False Negative): The model predicts the negative class incorrectly—the true value is positive.

Accuracy represents the ratio of correct predictions to the total number of input values. It is calculated using the following formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}.$$

2.10 Methodology section conclusions

This section describes the theories related to data processing, machine learning methods, and convolutional neural networks used in biometric authentication based on keystroke dynamics. The topics discussed include types of user authentication, similarity and distance metrics, dimension reduction methods, machine learning models, ensemble models, data transformations, time series data transformation into images, CNNs and evaluation metrics. Data preparation is a crucial step in building effective machine learning models for keystroke dynamics classification. Based on these methods, the study classifies the genuine and imposter subjects.

3 Experiments and results

The text processing, data analysis, and machine learning model codes were written in the *Python* programming language (see Appendix D: Code), using the *Pycharm* application.

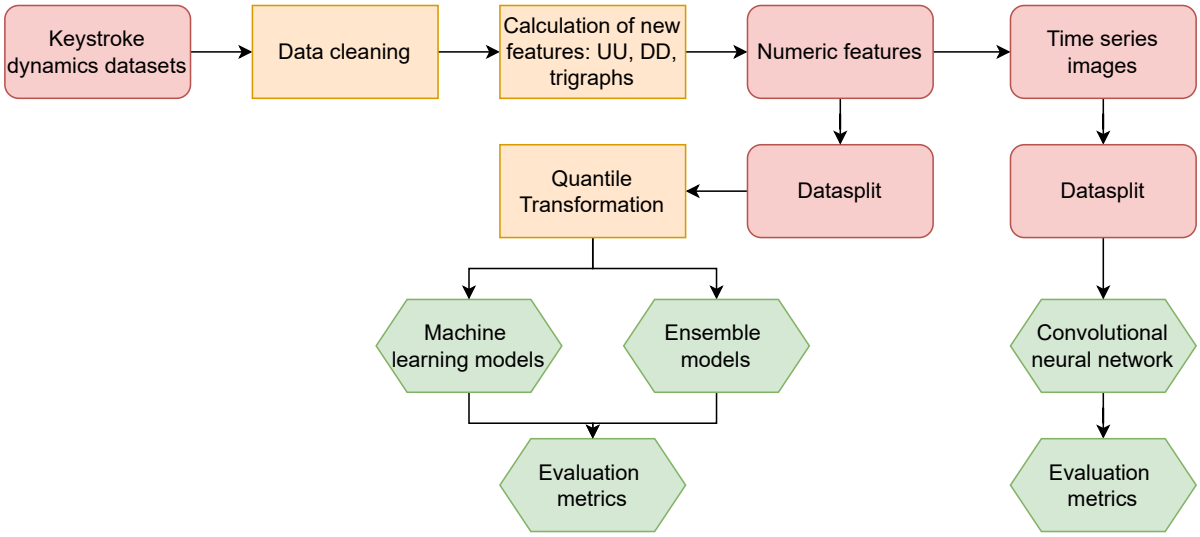


Figure 11: *Data analysis diagram*

The data processing sequence is illustrated in Fig. 11. Red rectangles represent raw data or intermediate datasets, orange rectangles represent variable transformations or added new variables, and green hexagons represent the methods used for data classification and model evaluation.

The CMU dataset used for machine learning models and the convolutional neural network were split separately for each subject, following the logic outlined in Table 5 [59].

Table 5: *Data split for the CMU dataset*

Dataset	Subject X (Genuine)	Other Subjects (Impostor)
Training	Sessions 1-4	Session 5 (First 5 repetitions)
Testing	Sessions 5-8	Session 1 (First 5 repetitions)

The dataset split for both the CMU and KeyRecs datasets is designed to evaluate the model's ability to distinguish between genuine and impostor samples while avoiding data leakage between the training and testing phases. For the CMU dataset, the training data for a selected subject (genuine user) consists of all repetitions from sessions 1 to 4, labeled as "genuine" and the first 5 repetitions from session 5 of all other subjects, labeled as "impostor". The test data include all repetitions from sessions 5 to 8 of the genuine user and the first 5 repetitions from session 1 of all other subjects as impostors.

Table 6: *Data split for the KeyRecs dataset*

Dataset	Subject X (Genuine)	Other Subjects (Imposter)
Training	Session 1	Session 2 (First repetition)
Testing	Session 2	Session 1 (First repetition)

In the KeyRecs dataset, the split follows a similar structure but with fewer sessions and repetitions (see Table 6). The training data comprises all repetitions from session 1 of the genuine user and the first repetition from session 2 of all other subjects as impostors. In contrast, the testing data include all repetitions from session 2 for the genuine user and the first repetition from session 1 for all other subjects as impostors. Each subject is treated as a genuine user in their dataset, with all other subjects acting as impostors, enabling independent analysis for every user.

3.1 Carnegie Mellon University Dataset

The CMU dataset [31] was collected by Computer Science Department professors and is described in the article [32]. As explained by the researchers, the data is arranged as a table with 34 columns. Each row of data corresponds to the timing information for a single repetition of the password by a single subject.

The password was chosen to pass a password-strength checker, which determines whether the password is strong or weak. This means that the password had to contain letters, numbers, and punctuation. Additionally, a length of 10 was chosen based on the reviewed articles on this topic, which revealed that 10-character passwords are typical and that longer passwords usually contain names and English phrases. The following password was generated: *.tie5Roanl*

The researchers recruited 51 participants for the data collection. In total there were 8 sessions during which participants were asked to type the password 50 times if the participants input the password incorrectly, and the system asked to retype the sequence. All sessions were conducted with at least one day between them. All the data was collected with the same keyboard and system, which captured the times of keyboard key presses and released with ± 200 microseconds precision.

After each input of the password, the subjects had to press the *ENTER* key, which became the 11 characters of the password. Based on these characters and information collected from the system about participants typing patterns the following features were presented in the data frame. The first column, *subject*, is a unique identifier for each subject (e.g., s002 or s057). The second column, *sessionIndex*, is the session in which the password was typed (ranging from 1 to 8). The third column, *rep*, represents the repetition of the password within the session (ranging from 1 to 50). The remaining 31 columns in the dataset contain timing information for the password. The column names encode the type of timing information being represented. Columns with names in the form H.key represent hold time, which is the time interval between pressing and releasing the same key. Columns named in the form DD.key1.key2 indicate the keydown-keydown time interval, measured from when key1 was pressed to when key2 was pressed. Similarly, columns with names in the form UD.key1.key2 denote the keyup-keydown time interval, which is the duration from releasing key1 to pressing key2. The timing for each feature is stored as a floating-point number in seconds.

Additionally, several features were derived and added to the dataset based on the original data [47]. The UU.key1.key2 feature represents the keyup-keyup time, which is calculated as the sum of the UD.key1.key2 time and the H.key2 time. Trigraphs were computed as the sum of two DD.key1.key2 features, representing the timing of three consecutive key presses. The DU.key1.key2 feature represents the keydown-keyup time, measured as the duration from when key1 was pressed to when key2 was released, calculated as the sum of the UD.key1.key2 time and the H.key2 time. In total, the final dataset comprises 60 features.

The descriptive analysis of the CMU dataset, presented in Table C1, shows that the average H keystroke times typically range between 0.07 and 0.1 seconds. This indicates that keys are generally released quickly

and consistently, with a standard deviation of hold times of approximately 0.02 to 0.03 seconds, reflecting low variability across users.

Similarly, the mean values for UD keystroke times range from 0.07 to 0.28 seconds, highlighting quick transitions between key presses. Unlike hold times, the minimum UD values can sometimes be negative, ranging from -0.23 to 0.1 seconds, due to overlaps between key releases and subsequent presses. While the maximum UD values can reach up to 12.5 seconds, the maximum hold time remains significantly lower at 2.035 seconds.

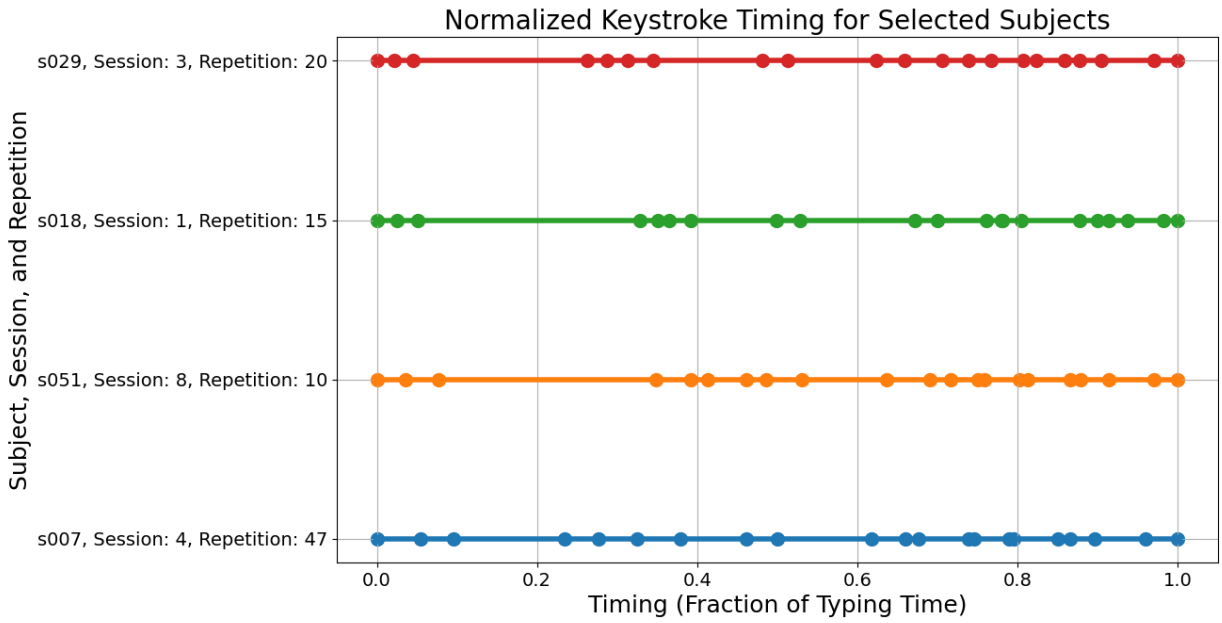


Figure 12: *Subjects normalized timing comparison*

Other keystroke types, such as DD, DU, UU, and tri, which are combinations of H and UD, present higher values and greater variability. This increased variability arises because these metrics capture more complex interactions involving multiple key presses and releases. The timing of these interactions naturally varies depending on the user's typing rhythm and coordination, resulting in broader ranges and more diverse patterns.

Fig. 12 illustrates the differences between the four typing attempts for various users. The x-axis represents cumulative normalized timing values in seconds, while the y-axis is labeled "Subject, Session, and Repetition" displaying unique identifiers for each combination of subject (e.g., s002, s003), session (e.g., Session 1, Session 2), and repetition (e.g., Repetition 1, Repetition 2).

The plot features multiple lines with circular markers at distinct points, representing the normalized timing of keystroke events. Each line corresponds to a specific combination of subject, session, and repetition, with each marker indicating a key press or release event. This visualization highlights the variations in typing patterns across different users and attempts.

After observing the figure, certain patterns in the typing behavior become apparent. For example, the first and second characters of the password are often typed consecutively, followed by a noticeable pause between the press and release of the second character. Similarly, the fourth character tends to have a longer hold time.

Notably, the timing patterns of subjects 29 and 18 share similarities, particularly at the start of typing, where both exhibit a pattern of presses and releases interspersed with occasional longer hold times. In contrast, subjects 51 and 7 demonstrate a steadier typing rhythm across all keys, indicating a more consistent typing style.

These patterns are also evident in Fig. 13, which displays the raw timing data. It is clear from the figure that subject 18's overall timing is approximately twice as long as that of subjects 7 and 51. subjects 7 and 51 maintain a steady typing rhythm throughout, whereas subjects 29 and 18 exhibit longer hold times, adding noticeable pauses to their timing patterns.

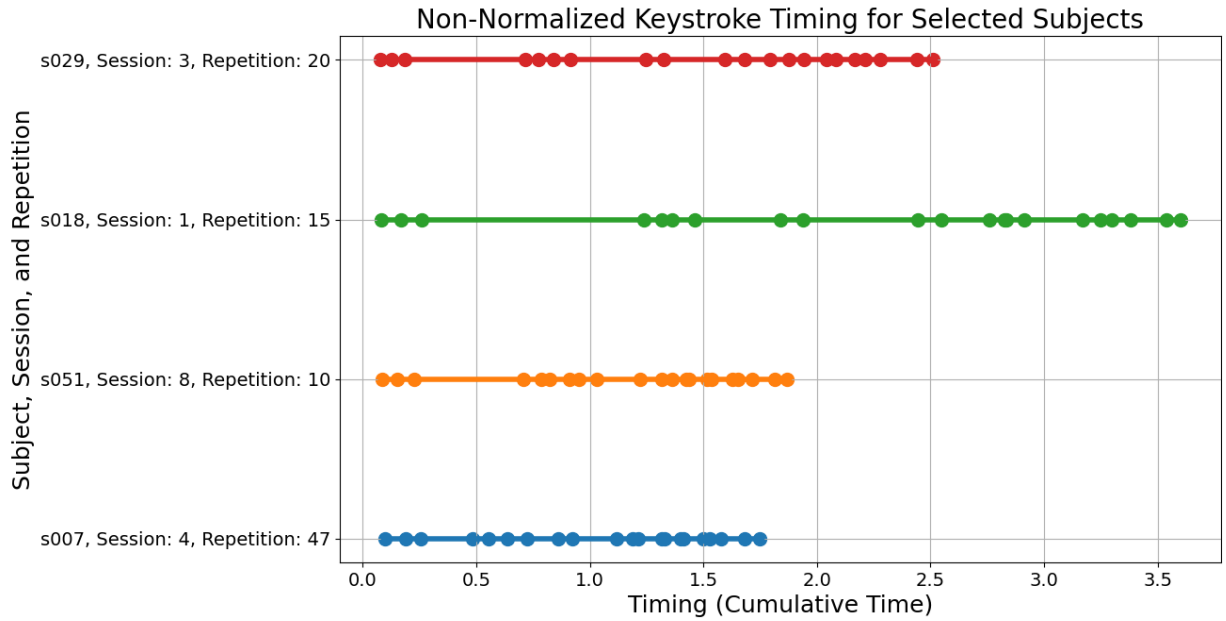


Figure 13: *Subjects timing comparison*

To evaluate the overall similarity between subjects, cosine similarity and Euclidean distance metrics were calculated. Figure 14 illustrates a subset of subject similarities, revealing that all subjects exhibit considerable similarity, with cosine similarity values above 0.85 for all pairs. The lowest cosine similarity, 0.854, is observed between subjects 2 and 52, while the largest Euclidean distance, 3.696, is found between subjects 36 and 55. Furthermore, the highest cosine similarity, 0.997, is noted between subjects 18 and 29, and the smallest Euclidean distance, 0.171, occurs between subjects 7 and 51.

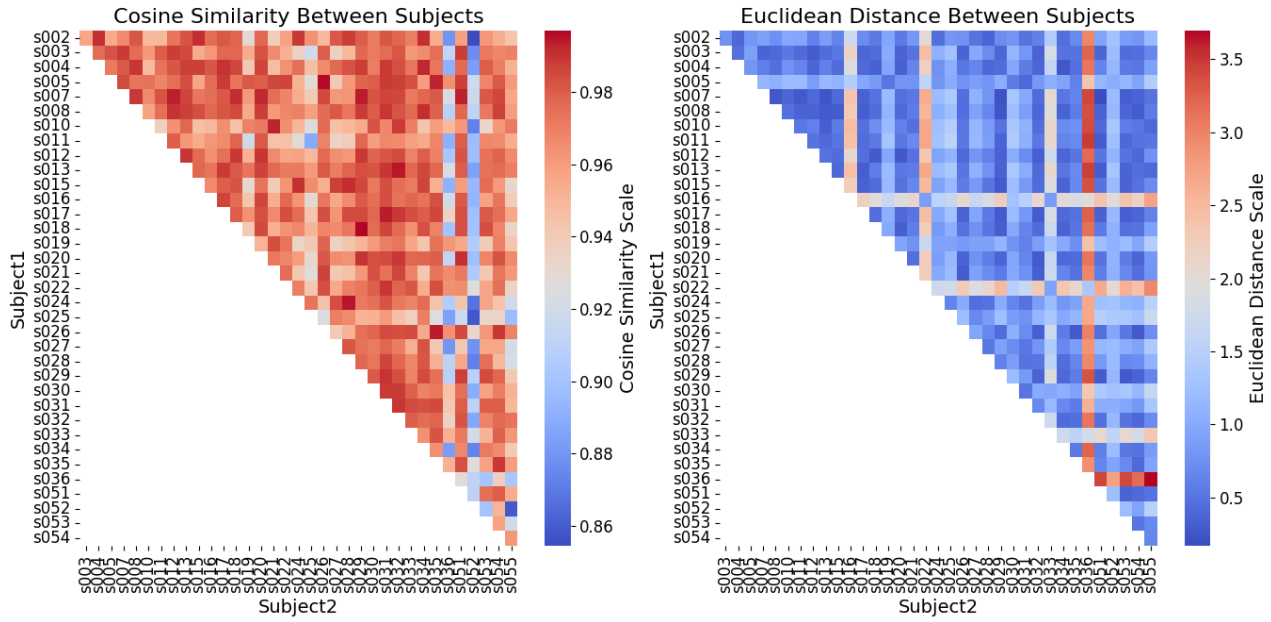


Figure 14: *Cosine similarity and Euclidean distance between subjects*

To further investigate the patterns in H and UD feature values, Fig. 15 was plotted to determine whether these attributes exhibit similarities that are consistent with the findings of the similarity graph. These metrics were chosen because all other data points in the dataset are derived from them. The graphs are based on the subject pairs identified in the previous analysis.

The resulting linear graphs show that the UD attributes for the most similar pairs align closely, indicating consistent transitions between key presses. However, notable differences in the H attribute values are observed, even among pairs with high similarity metrics. Additionally, pairs with lower similarity exhibit more pronounced discrepancies in H feature values, highlighting variability in key hold times across less similar subjects.

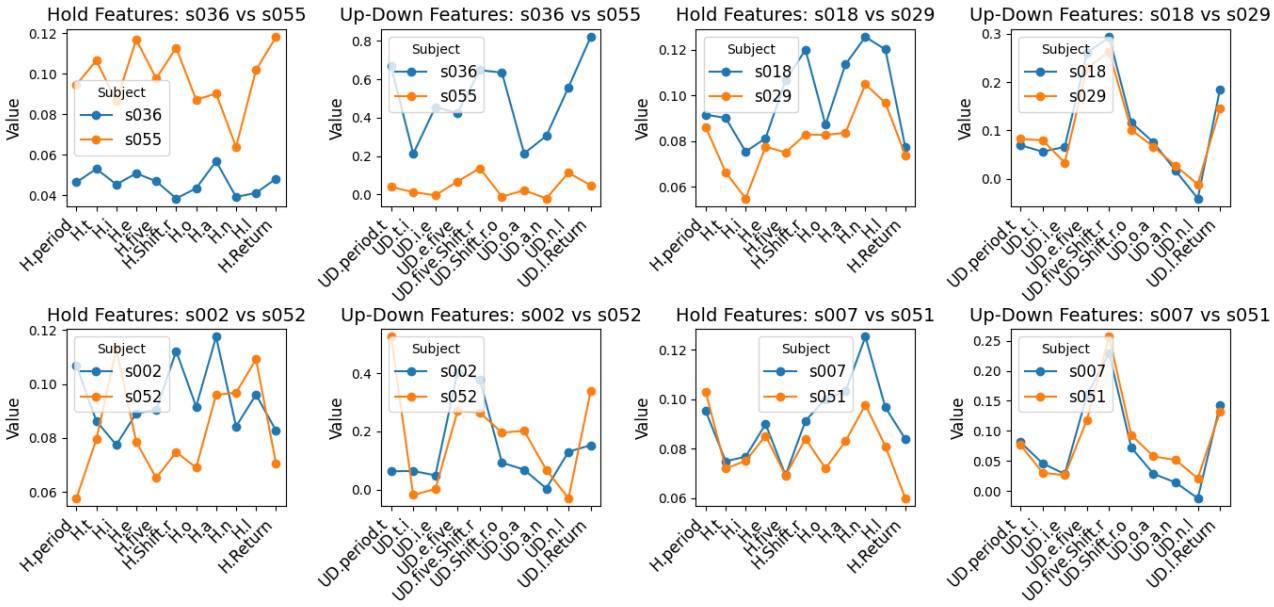


Figure 15: *H and UD feature comparison in CMU dataset*

Furthermore, dimensionality reduction and data visualization techniques are important in machine learning, especially when working with complex datasets. These methods provide valuable insights into similarity relationships within multidimensional data [34]. Fig. 16 presents the results of applying various dimensionality reduction techniques to pairs derived from cosine similarity and Euclidean distance. The techniques include radial basis function (RBF) kernel PCA with *n_components*=2, UMAP with *n_components*=2, *n_neighbors*=10, *min_dist*=0.01, and metric MDS with *n_components*=2, as well as t-SNE with *n_components*=2, *perplexity*=3, *learning_rate*=200, and *n_iter*=3500. The parameters for the dimensionality reduction methods were selected after thorough evaluation and multiple experimental trials to ensure optimal performance.

The Fig. 16 demonstrates that pairs with low similarity or high Euclidean distance can be effectively separated using all dimensionality reduction methods. However, for pairs of similar subjects, these methods are less effective at achieving clear separation. Among the techniques, t-SNE proved to be the most effective in distinguishing patterns for closely related subjects.

In this analysis, t-SNE is used solely for visualization purposes, not for dimensionality reduction. While t-SNE can reduce dimensionality for visualization by projecting high-dimensional data into a 2D or 3D space, it does not retain global structure, making it unsuitable for dimensionality reduction in the feature space used for classification.

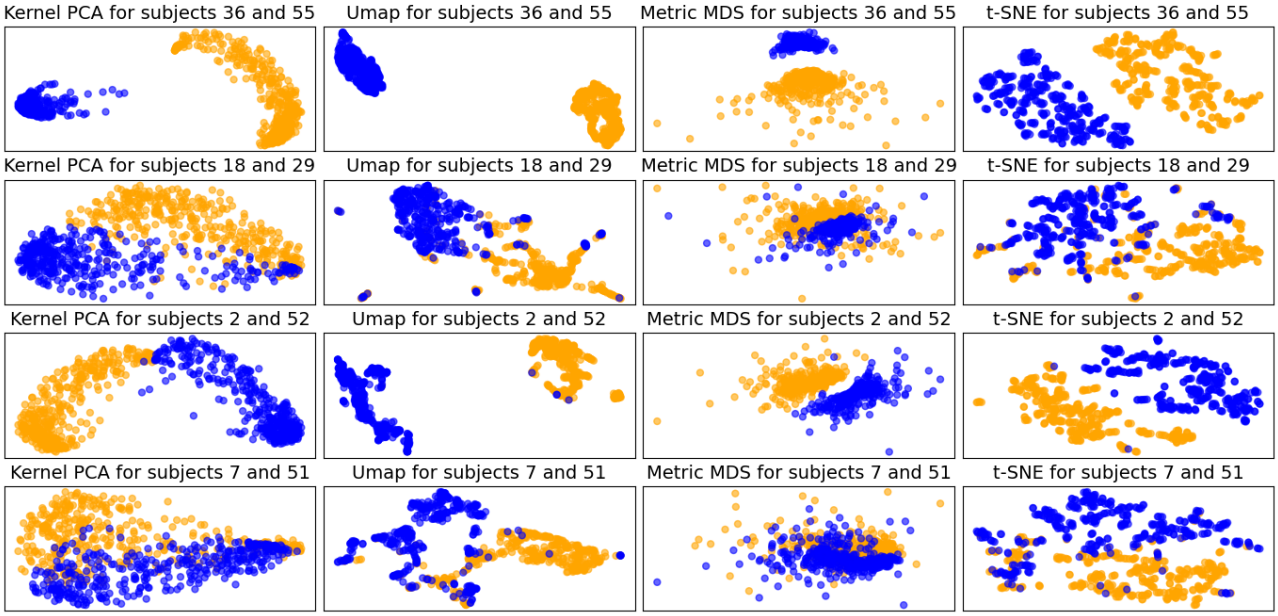


Figure 16: *Multidimensional data in a lower dimensional space for CMU dataset*

Moreover, all dimensionality reduction techniques are applied solely for visualization purposes rather than for dimensionality reduction. Time series data is highly sensitive to transformations that could distort or obscure critical temporal relationships. Reducing the dimensionality of such data risks losing essential information needed to capture the nuanced patterns in keystroke dynamics, which are crucial for accurate classification. By preserving the original feature dimensions, this approach ensures that all time-sensitive characteristics of the data are fully retained and effectively utilized in the analysis.

Therefore, after the dataset was split, six machine learning models and two ensemble methods were applied to evaluate four different versions of the data and assess their performance. Models were chosen based on the research conducted during the literature review. The following machine learning models were chosen: DT, GB, KNN, RF, SVM, XGB. Using the ensemble model proposed in [10] as a baseline, the decision tree model was replaced with a random forest model to improve performance. Compared with the single DT, RF offer greater robustness and accuracy by combining multiple trees through bagging, which reduces overfitting and enhances generalization. Additionally, an XGB model was incorporated into the ensemble, leveraging its efficiency and superior handling of complex data patterns. These modifications were used to create ensemble models for both the voting and stacking methods, enhancing the overall classification.

Hyperparameter tuning for machine learning classification models was conducted using 5-fold cross-validation. The tuned parameters are shown in Table C3. Cross-validation was applied to the training data to evaluate different hyperparameter combinations, and the best set of hyperparameters was selected based on the accuracy scoring metric.

For the CNN model, random search was employed to optimize accuracy and EER values by exploring a wide range of hyperparameter combinations. When dealing with a large number of parameters, random search offers a significant advantage over grid search, as it efficiently identifies models with comparable or superior performance while significantly reducing computational time [14].

A comprehensive analysis of the dataset's structure and preprocessing techniques, including feature extension and normalization, was conducted to evaluate their impact on the performance of machine learning models and ensemble methods. The first model utilized the extended dataset, which included additional features calculated during explanatory data, combined with quantile transformation to normalize the data.

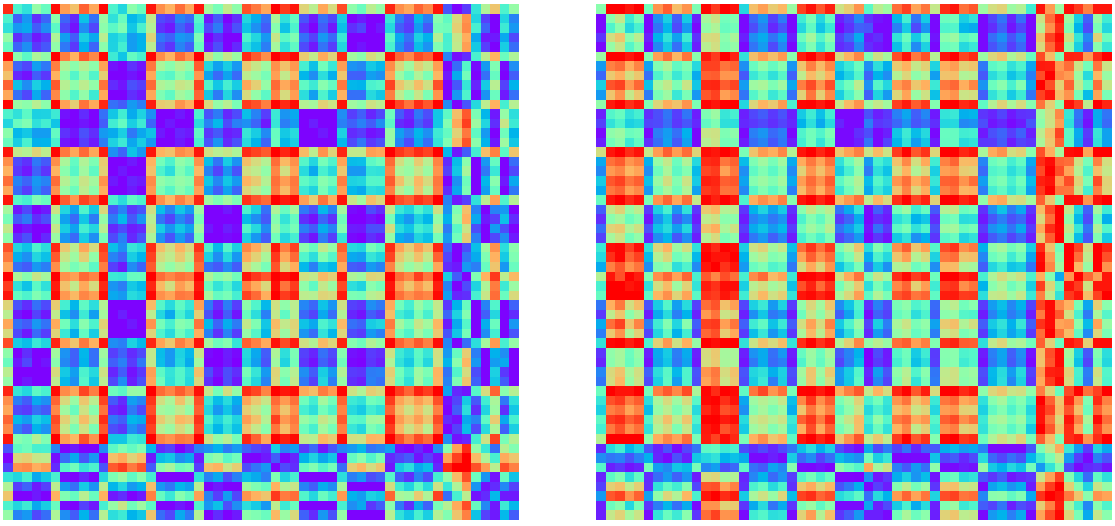
The second model also employed the extended dataset but without applying quantile transformation. The third model used the original dataset, with quantile transformation. Finally, the fourth model used the original dataset without quantile transformation, serving as a baseline.

After testing different feature sets and evaluating the impact of using or not using transformations, the results were very similar between the original and extended datasets across all classifiers. This finding indicates that the additional features do not provide any extra information that the classification models can utilize. However, slightly higher accuracy results were observed with transformations than without transformations (see Table C4).

Table 7: *Performance metrics for different classifiers for CMU dataset*

Classifier	Accuracy	EER	FAR	FRR
DT	0.8962	0.1296	0.0643	0.1532
GB	0.9487	0.0450	0.0260	0.0828
KNN	0.9533	0.0461	0.0507	0.0418
RF	0.9479	0.0402	0.0163	0.0968
SVM	0.9541	0.0355	0.0210	0.0770
Stacking	0.9618	0.0329	0.0160	0.0659
Voting	**0.9648**	**0.0314**	0.0193	0.0550
XGB	0.9485	0.0449	0.0235	0.0866
CNN	0.9139	0.0873	0.0764	0.0978

The following results (see Table 7 and Fig. B2) summarize each model's accuracy, EER, FAR, and FRR calculated on the original dataset with quantile transformation to facilitate model comparison. The highest accuracy and EER performance were achieved by the voting ensemble model, with accuracy reaching 96.48% and an EER of 3.14%. The stacking ensemble method demonstrated the lowest FAR, indicating that it is the least likely to misclassify an impostor as a genuine subject. Meanwhile, the KNN model exhibited the lowest FRR, meaning that it is the least likely to misclassify a genuine user as an impostor.



(a) GASF Image **(b)** GADF Image

Figure 17: *Comparison of GASF and GADF images for subject 2, session 1, repetition 1*

To compare the performance of machine learning and ensemble models on numeric features with image classification using CNN, time-series transformations GASF (17a) and GADF (17b) were applied. These trans-

formations convert numeric features into images, which are then used as inputs for the CNN. To generalize and improve model robustness, Gaussian noise was introduced into the images during preprocessing [29]. This addition helps simulate real-world variability, making the model more robust. Furthermore, Gaussian noise acts as a form of regularization, reducing the risk of overfitting and encouraging the model to focus on meaningful patterns rather than artifacts or noise in the data.

Fig. B1 illustrates the customized architecture of the CNN used for the binary classification of keystroke dynamics time series images.

From Table 7 and Fig. B2, it can be observed that the CNN achieved lower accuracy than did the machine learning and ensemble models. Results of each subject seperately can be seen in Table C7. This performance discrepancy may be attributed to the CNN's complexity, which could make it less suitable for processing consistent and clean data, as it requires more diverse and variable input to fully leverage its capacity.

3.2 KeyRecs Dataset

Unlike the CMU dataset, which focuses on collecting highly accurate data using a predefined strong password, the KeyRecs dataset [21] allowed participants to use their own keyboards. Additionally, the password in KeyRecs consisted solely of letters and had a fixed length of 10 characters: *vpwjkeurkb*.

For data collection, 99 participants were recruited. Each participant completed two sessions, during which they were asked to type the password 100 times.

The dataset captured participants' typing patterns using the following features: DU.key1.key1, DD.key1.key2, DU.key1.key2, UD.key1.key2, and UU.key1.key2 for all characters in the password. These features are consistent with the abbreviations used in the CMU dataset. In total, 46 timing features were extracted, with each feature's timing stored as a floating-point number in seconds.

To ensure data consistency, several modifications were applied to the dataset. First, the DU.key1.key1 feature was renamed to H.key1, as it represents the hold time of a single key. Additionally, discrepancies were identified between some column names and the values they represented, prompting corrections to align the data accurately.

In Table 8, a snippet of the original data is provided, showing the keystroke values for the first row with the timing data for the first three symbols.

Table 8: *Keystroke timing data for participant p001, session 1, repetition 1*

Participant	Session	Repetition	DU.v.v	DD.v.p	DU.v.p	UD.v.p
p001	1	1	0.129	1.917	1.804	2.046
UU.v.p	DU.p.p	DD.p.w	DU.p.w	UD.p.w	UU.p.w	DU.w.w
1.933	0.113	0.719	0.569	0.832	0.682	0.15

The same data values are displayed on the left side of Fig. 18. As seen from the table and the image, some inconsistencies exist. For example, the value of *UD.v.p* is 2.045, which is higher than *DU.v.p*. However, the DU value should be calculated by adding *H.v* (0.129) and *H.p* (0.113) to *UD.v.p*, resulting in *DU.v.p* being 2.288. This discrepancy suggests swapping the DUs and UDs yields the correct values.

A similar issue exists with the DD and UU values. The *DD.v.p* value is 1.917, which should be calculated by adding *H.v* and the adjusted *UD.v.p* value, resulting in 1.933. This value can be observed in *UU.v.p*. If we calculate *UU.v.p* by adding *UD.v.p* and *H.p*, we obtain 1.917, which appears in DD. Therefore, swapping the

column names for DDs and UUs also corrects the inconsistency. The correct order of timings can be seen on the right side of Fig. 18. The colors in the figure show which feature names were swaped between each other.

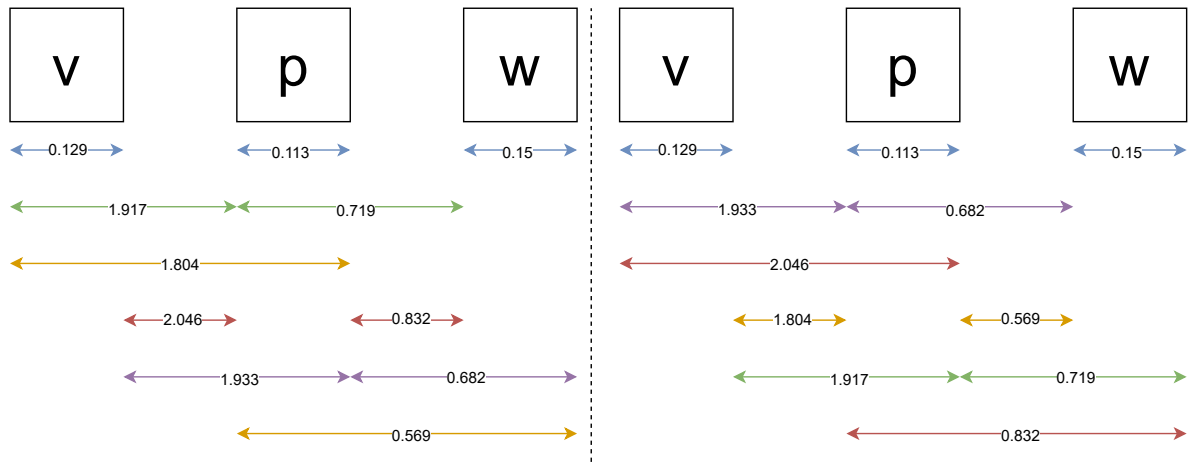


Figure 18: *Changes in keystroke timing data for participant p001, session 1, repetition 1*

The discrepancies were present throughout the dataset for all subjects. To address these inconsistencies, the column names DD and UU, as well as DU and UD, were swapped to reflect the correct timing relationships.

Additionally, trigraphs were calculated as the sum of two DD.key1.key2 features. The final dataset consists of 54 features related to keystroke timing, along with the subject number, session index, and repetition number.

Furthermore, 24 rows were eliminated due to negative values in the H and DD columns, as such values are illogical for these features. Additionally, 12 rows were removed because they exceeded the 20-second threshold for the H and UD features (the threshold was based on the CMU dataset values). All removed rows presented excessive UD timing values, as detailed in Table 9.

Table 9: *Deleted rows from KeyRecs dataset*

subject	session	rep	UD.v.p	UD.p.w	UD.w.j	UD.j.k	UD.k.e	UD.e.u	UD.u.r	UD.r.k	UD.k.b
p011	2	2	0.695	**51.323**	0.761	0.481	1.448	0.405	0.323	0.917	0.203
p020	2	4	0.226	0.065	0.427	**-1077833.755**	**1077833.851**	0.097	0.048	**1077834.017**	**-1077833.774**
p025	2	82	0.729	**67.254**	0.937	0.105	0.796	0.028	0.018	12.382	0.318
p042	2	23	**76.321**	0.321	0.010	0.181	-0.031	0.040	-0.020	0.064	0.021
p050	2	95	0.096	**35.890**	0.027	1.129	0.152	0.982	0.085	0.083	0.502
p056	1	26	0.233	**33.902**	0.483	0.713	0.502	0.198	0.153	0.514	0.330
p056	2	56	0.573	0.111	**28.286**	0.134	0.150	0.078	0.046	0.492	0.189
p056	2	81	0.124	0.900	1.038	0.149	0.054	0.079	0.039	**32.094**	0.179
p063	2	2	0.136	0.808	0.641	0.064	0.705	0.337	**20.157**	0.137	0.225
p071	1	20	-0.048	0.400	-0.064	**645.724**	0.032	0.272	0.031	0.544	0.064
p071	2	24	-0.048	0.048	0.256	**21.760**	-0.048	0.000	0.320	-0.032	0.496
p087	1	8	0.073	0.553	0.193	0.224	0.032	0.256	0.097	0.832	**332.605**

The descriptive statistics of the KeyRecs dataset are presented in Table C2. Similar to the CMU dataset, the average duration for UD timings ranges from 0.13 to 0.35 seconds. The minimum values for UD timings range from -0.18 to 0.1 seconds, indicating overlapping key actions where one key is released as the next is pressed. However, the maximum UD values, such as *UD.w.j* at 19.38 seconds, reflect occasional pauses between key releases and subsequent key presses.

H times exhibit low variability, with means ranging from 0.09 to 0.1 seconds and standard deviations of approximately 0.03 seconds. This consistency suggests quick key releases. The relatively small maximum values for H timings, such as *H.k.k.1* with a maximum of only 0.292 seconds, compared to other timing features.

Overall, while H and DD timings demonstrate stable and consistent keystroke durations across users, the variability in UD and UU values reflects the diversity in key transition patterns. This variability is likely influenced by individual differences in typing speed and rhythm.

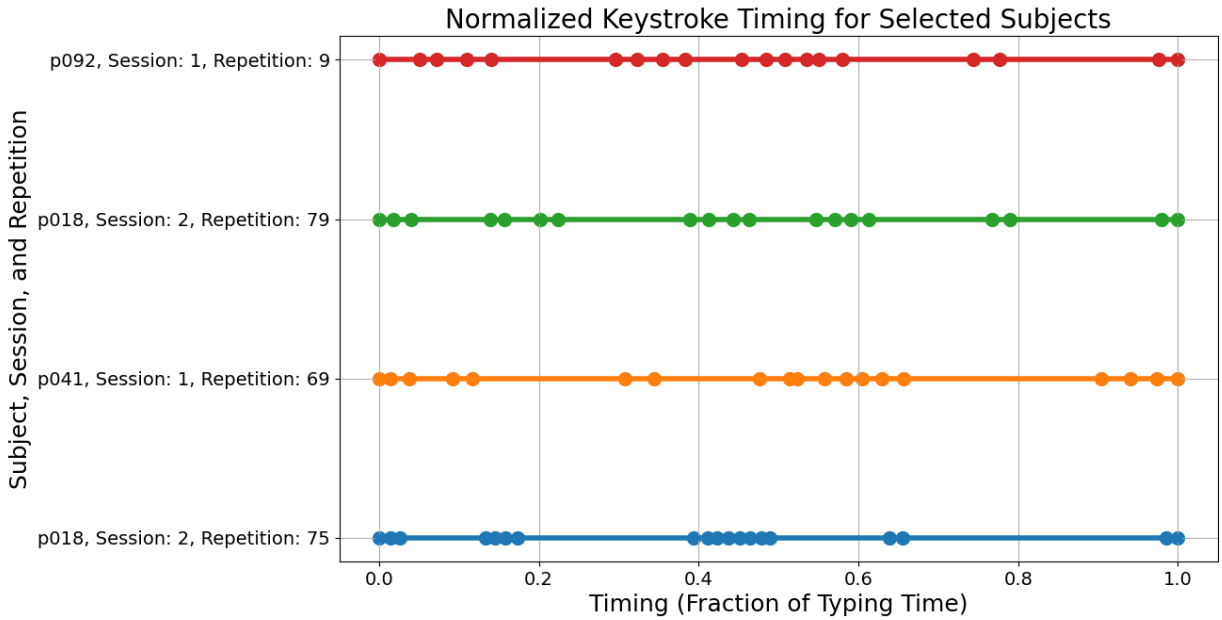


Figure 19: *Subjects normalized timing comparison*

Like in the CMU dataset, fraction graphs were generated to compare users in the KeyRecs dataset, as shown in Fig. 19. The normalized figure reveals notable timing patterns among subjects. For instance, all subjects in the graph exhibit distinct pauses at specific points in the typing sequence. This is particularly evident in subject 18's 79 repetitions, where dots are densely concentrated at certain points, followed by occasional longer intervals between keystrokes.

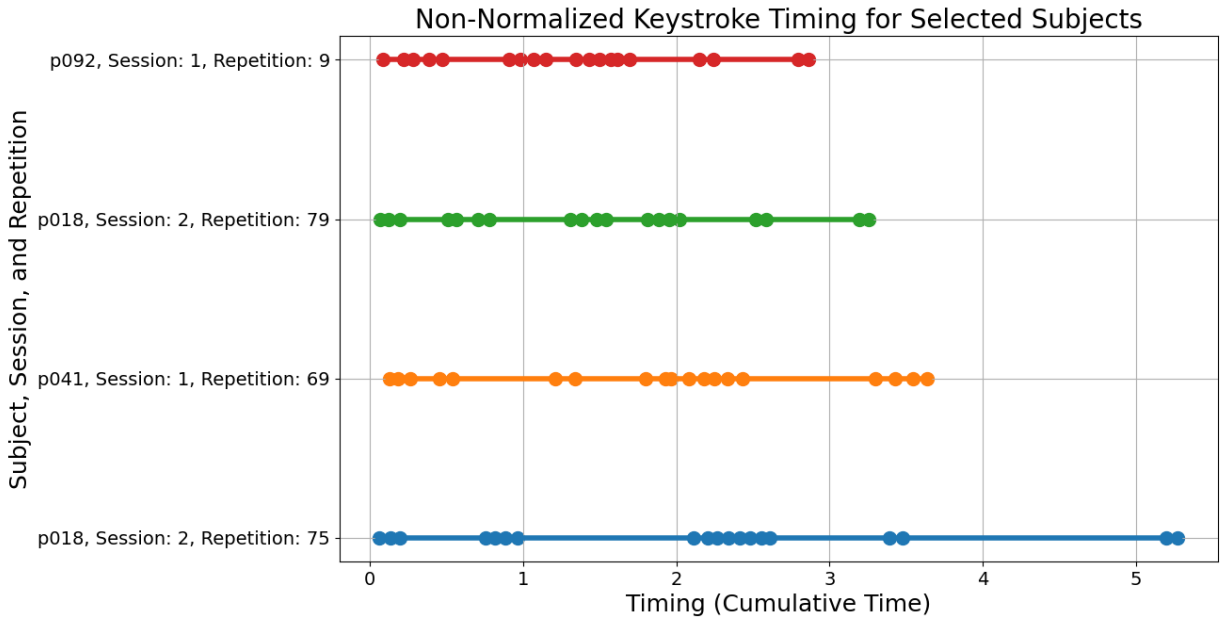


Figure 20: *Subjects timing comparison*

From the normalized patterns, both lines representing subject 18 exhibit very similar keystroke timing patterns, suggesting a consistent typing rhythm across different sessions, as shown in Fig. 20. However, when the raw timing data for the same subject is examined, it is evident that repetition 79 takes significantly longer to complete.

Additionally, subject 92 demonstrates a keystroke pattern similar to that of subject 18 toward the end of the typing sequence, suggesting shared timing characteristics in their final keystrokes. Interestingly, subjects 41 and 92 exhibit comparable patterns at the beginning of their sequences, indicating a similar approach or rhythm in the initial stages of typing. These observations highlight both the individual consistency of typing patterns and the presence of cross-subject similarities in specific parts of the typing sequence.

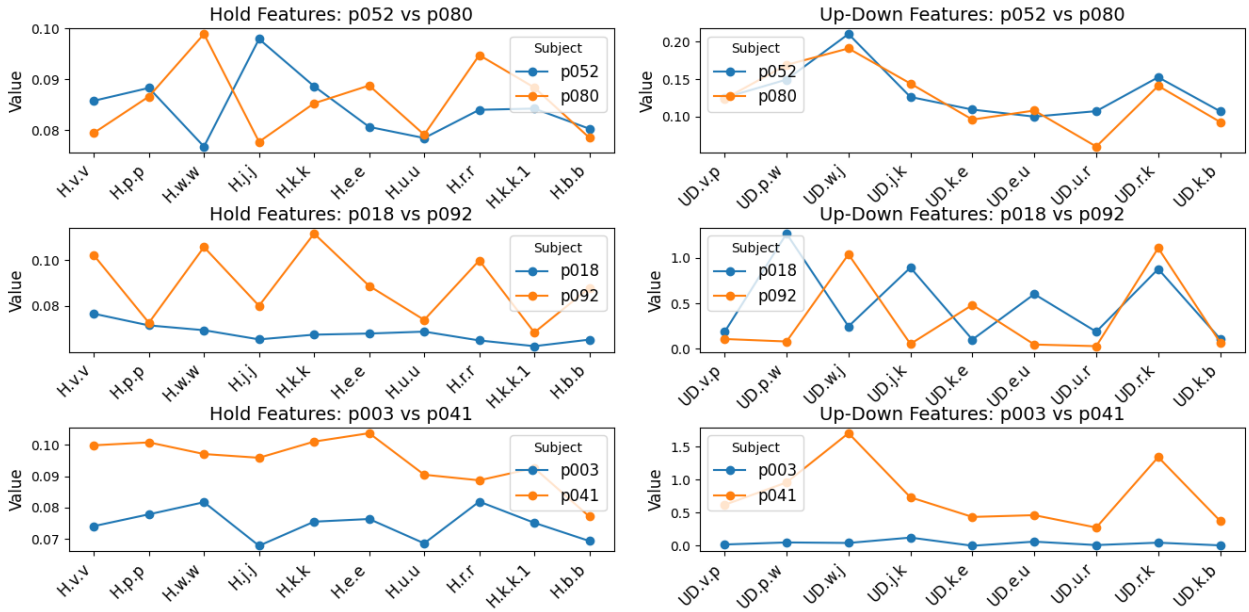


Figure 21: *H and UD feature comparison in KeyRecs dataset*

After calculating the cosine similarity and Euclidean distance between subjects in the KeyRecs dataset, the following pairs were identified: the lowest cosine similarity value was 0.830 between subjects 18 and 92, whereas the highest Euclidean distance was 9.745 between subjects 3 and 41. Moreover, the highest cosine similarity of 0.998 was observed between subjects 52 and 80, and the smallest Euclidean distance, 0.152, was also found between these two subjects.

To further investigate whether the H and UD attribute values align with these similarity metrics, linear graphs were plotted, as shown in Fig. 21.

The resulting linear graphs indicate that all pairs exhibit significant differences, except for participants 52 and 80, which showed relatively similar UD features.

Fig. 22 presents the results of KPCA, UMAP, metric MDS, and t-SNE as dimensionality reduction methods applied to pairs derived from cosine similarity and Euclidean distance, using the same dimension reduction model parameters as those in the CMU dataset. The results show that pairs with low similarity or high Euclidean distance are effectively separated using all dimensionality reduction methods. However, for similar subjects, the t-SNE method proved to be the most effective, although it does not seperate all data points.

Similarly to the CMU dataset, after the KeyRecs dataset was split into training and testing sets, six machine learning models and two ensemble methods were applied across four scenarios: using either the extended dataset or the original dataset, with or without transformation. Hyperparameter tuning was conducted

using random search. The results were similar, demonstrating that the extended version of the dataset does not provide significantly more information. However, transformations slightly improved the results (see Table C5).

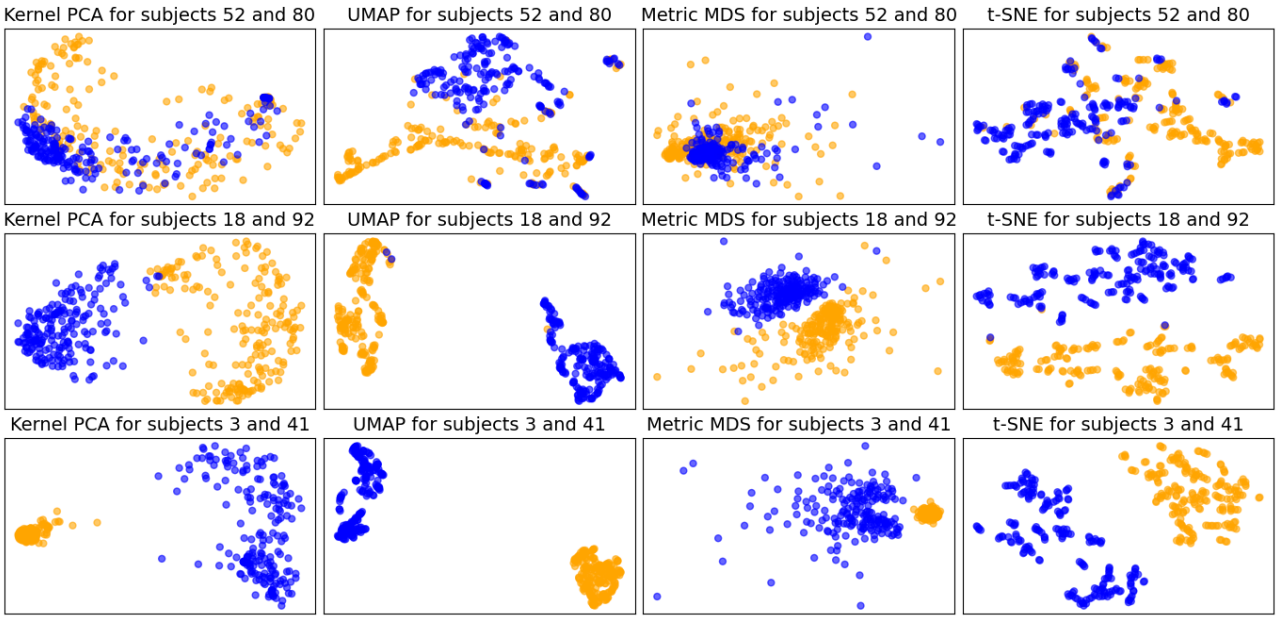


Figure 22: *Multidimensional data in a lower dimensional space for the KeyRecs datset*

The results of the original dataset with transformation (See Table 10 and Fig. B3) show that the best model among the machine learning classifiers and ensemble methods according to accuracy is the voting ensemble, with an accuracy of 86.63%, and while the lowest EER of 11.07% was achieved by the stacking ensemble. However, after running a CNN with GASF and GADF images with additional Gaussian noise as inputs, an accuracy of 90.45% and an EER of 10.06% were achieved, improving on the performance of the machine learning models. Results of each subject seperately can be seen in Table C6. It is also important to note that the FAR and FRR results for the CNN are very similar, at 9.19% and 10.07%, respectively. This indicates a balanced likelihood of the model misclassifying an impostor as a genuine user or vice versa. In contrast, for the machine learning models, the FRR is consistently much higher than the FAR except for KNN—meaning these models are more likely to misclassify a genuine user as an impostor, potentially making the login process less convenient for users.

Table 10: *Performance metrics of different classifiers*

Classifier	Accuracy	EER	FAR	FRR
DT	0.7956	0.2268	0.1335	0.2733
GB	0.8510	0.1352	0.0814	0.2147
KNN	0.8690	0.1266	0.1408	0.1215
RF	0.8531	0.1240	0.0606	0.2307
SVM	0.8608	0.1086	0.0616	0.2147
Stacking	0.8767	0.1163	0.0691	0.1976
Voting	0.8822	0.1107	0.0636	0.2018
XGB	0.8616	0.1202	0.0702	0.2047
CNN	**0.9045**	**0.1006**	0.0919	0.1007

4 Conclusion

This study evaluates and compares machine learning models for user authentication based on keystroke dynamics, leveraging both numerical features and time-series image transformations to classify genuine users and impostors.

The literature review explores various methods for keystroke dynamics-based biometric authentication, analyzing statistical, machine learning, and deep learning approaches. Key findings highlight the effectiveness of traditional models such as k-nearest neighbor (KNN) and support vector machine (SVM) for structured data. In contrast, deep learning methods, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), demonstrate superior performance in free-text and image-based analyses. Challenges identified include user variability, differences in device types, and the influence of emotional states, fatigue, and environmental factors on typing behavior. This study addresses these challenges by exploring innovative time-series-to-image transformations and testing their effectiveness on diverse datasets.

This study not only provides a comprehensive analysis of the widely used CMU dataset, which is often utilized in keystroke dynamics research but also, introduces the novelty of evaluating a relatively new KeyRecs dataset. The CMU dataset offers structured, fixed-text typing samples across multiple sessions, enabling controlled evaluations of machine learning models. In contrast, the KeyRecs dataset provides data from a variety of sources, capturing fixed-text typing behavior across diverse devices. This diversity makes KeyRecs particularly suitable for testing model generalization in real-world scenarios.

Data preprocessing involved cleaning and preparing the datasets to ensure reliable model performance. This included addressing inconsistencies, generating additional features from raw timing data, and applying data transformations. Exploratory analysis revealed that, while subjects shared numerous similarities in their typing patterns, classification was particularly challenging due to closely aligned trends in some features. For example, in the CMU dataset, subjects 18 and 29, as well as subjects 7 and 51, exhibited very similar trends in the keyup-keydown feature. Similarly, in the KeyRecs dataset, subjects 52 and 80 demonstrated closely aligned trends in the same feature. However, subjects with more distinct typing patterns were easier to differentiate, emphasizing the importance of feature separability within the dataset.

Unlike traditional approaches that rely solely on numeric features for classification, this research explores an innovative use of time-series transformations, such as gramian angular summation fields (GASF) and gramian angular difference fields (GADF), to convert numeric keystroke data into visual representations. These transformations facilitated the application of CNNs, enabling a direct comparison between the performance of machine learning and ensemble models on numeric features and CNNs on image-based data. All evaluations were conducted using consistent data cleaning and preprocessing techniques.

Various machine learning models were assessed for their ability to distinguish between genuine and impostor users. Commonly used classifiers, such as KNN, SVM, and gradient boosting (GB), achieved high accuracy rates of 94–95% and low equal error rates (EERs) ranging from 0.03 to 0.04 on the structured CMU dataset, collected in a controlled environment. The voting ensemble method achieved the highest accuracy of 96.48% and an EER of 0.0314. However, CNNs did not yield further improvements in the CMU dataset. In contrast, on the KeyRecs dataset, these classifiers achieved accuracy levels of 85–86%, while the voting ensemble method reached 88%. CNNs outperformed all other methods on the KeyRecs dataset, achieving an accuracy of up to 90.45% and an EER as low as 0.1006. These results underscore the value of analyzing keystroke dynamics using both numeric features and time-series image representations. This novel methodology enables a fresh

perspective on comparing traditional machine learning models with deep learning approaches for keystroke dynamics.

A key finding of this research is that extended feature sets provided minimal improvement, suggesting that additional features may not always contribute significant new information for model training. On the other hand, the application of data transformations improved performance metrics across both datasets, emphasizing the importance of robust preprocessing in keystroke dynamics research. Furthermore, this study highlights the strengths and weaknesses of different approaches. For instance, machine learning models exhibited higher false rejection rates (FRR) compared to false acceptance rates (FAR), indicating a tendency to reject genuine users more frequently than impostors. In contrast, CNNs demonstrated a more balanced performance between these two metrics, with significantly lower FRR. This balance makes CNNs a promising option for practical biometric systems, as they reduce the inconvenience of rejecting genuine users while maintaining strong security against impostors.

Future research should explore diverse real-world datasets and conduct real-time evaluations under varying conditions, such as different keyboard types and user stress levels, to enhance model robustness and practical applicability. Hybrid approaches that combine numeric and image-based methods could leverage the strengths of both for improved accuracy and robustness. Additionally, alternative time-series-to-image transformation techniques, such as recurrence plots or spectrograms, should be investigated to better capture temporal patterns. Finally, advanced deep learning architectures, such as attention-based models, could provide novel ways to emphasize critical features in keystroke dynamics.

References

[1] A. Abd Al-salam Selami, A. Fadhil. “A Study of the Effects of Gaussian Noise on Image Features.” In: *Kirkuk University Journal / Scientific Studies (1992-0849)* 11 (2016), pages 152–169.

[2] H. AbdelRaouf, S. A. Chelloug, A. Muthanna, N. Semary, K. Amin, M. Ibrahim. “Efficient Convolutional Neural Network-Based Keystroke Dynamics for Boosting User Authentication.” In: *Sensors* 23 (2023). https://doi.org/10.3390/s23104898.

[3] A. Acien, A. Morales, J. V. Monaco, R. Vera-Rodriguez, J. Fierrez. “TypeNet: Deep Learning Keystroke Biometrics.” In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 4 (2022), pages 57–70. https://doi.org/10.1109/TBIOM.2021.3112540.

[4] A. Ahmed, I. Traore. “Biometric Recognition Based on Free-Text Keystroke Dynamics.” In: *IEEE transactions on cybernetics* 44 (2014), pages 458–472. https://doi.org/10.1109/TCYB.2013.2257745.

[5] M. L. Ali, J. V. Monaco, C. C. Tappert, M. Qiu. “Keystroke Biometric Systems for User Authentication.” In: *Journal of Signal Processing Systems* 86 (2017), pages 175–190. https://doi.org/10.1007/s11265-016-1114-9.

[6] O. Alpar. “Biometric keystroke barcoding: A next-gen authentication framework.” In: *Expert Systems with Applications* 177 (2021), page 114980. https://doi.org/https://doi.org/10.1016/j.eswa.2021.114980.

[7] A. A. S. AlQahtani, Z. El-Awadi, M. Min. “A Survey on User Authentication Factors.” In: *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2021, pages 323–328. https://doi.org/10.1109/IEMCON53756.2021.9623159.

[8] J. Alzubi, A. Nayyar, A. Kumar. “Machine learning from theory to algorithms: an overview.” In: *Journal of physics: conference series* 1142 (2018). https://doi.org/10.1088/1742-6596/1142/1/012012.

[9] T. Attaie, J. Caldwell, T. Ward, Y. Yassin, J. Graham, K. Elliot. “Dynamic Keystroke for Authentication with Machine Learning Algorithms.” In: *Proceedings of the 2019 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2019, pages 1138–1143.

[10] B. M. Azanguezet Quimatio, O. F. Yatio Njike, M. Nkenlifack. “User Authentification through Keystroke dynamics based on ensemble learning approach.” In: *CARI 2022 - Colloque Africain sur la Recherche en Informatique et en Mathémathiques Appliquées*. Dschang, Cameroon: HAL, 2022.

[11] N. Bakelman, J. V. Monaco, S.-H. Cha, C. Tappert. “Keystroke Biometric Studies on Password and Numeric Keypad Input.” In: *Proceedings - 2013 European Intelligence and Security Informatics Conference, EISIC 2013* (2013), pages 204–207. https://doi.org/10.1109/EISIC.2013.45.

[12] S. Banerjee, D. Woodard. “Biometric Authentication and Identification Using Keystroke Dynamics: A Survey.” In: *Journal of Pattern Recognition Research* 7 (2012), pages 116–139. https://doi.org/10.13176/11.427.

[13] A. Bastys. *Atraminių vektorių klasifikatorius*. https://kedras.mif.vu.lt/bastys/academic/ATE/biometrika/SVM.pdf. Žiūrėta 2025 m. sausio 2 d.

[14] J. Bergstra, Y. Bengio. “Random Search for Hyper-Parameter Optimization.” In: *The Journal of Machine Learning Research* 13 (2012), pages 281–305.

[15] J. Blomqvist. “Using XGBoost to classify the Beihang Keystroke Dynamics Database.” Master’s thesis. Uppsala University, 2018.

[16] S. Brekke, P. Bours. “Continuous Age Detection using Keystroke Dynamics.” In: *Proceedings of the Norwegian Information Security Conference (NISK)*. Bergen, Norway, 2024. URL: https://www.ntnu.no/ojs/index.php/nikt/article/view/6247.

[17] A. Budžys, O. Kurasova, V. Medvedev. “Deep learning-based authentication for insider threat detection in critical infrastructure.” In: *Artificial Intelligence Review* 57 (2024). https://doi.org/10.1007/s10462-024-10893-1.

[18] H.-C. Chang, J. Li, C.-S. Wu, M. Stamp. “Machine Learning and Deep Learning for Fixed-Text Keystroke Dynamics.” In: *Artificial Intelligence for cybersecurity*. Springer, 2022, pages 309–329. https://doi.org/10.1007/978-3-030-97087-1_13.

[19] Curie. *Curie [AI Model]*. 2024. URL: https://www.aje.com/curie/ (viewed 2025-01-02).

[20] R. K. Das, S. Mukhopadhyay, P. Bhattacharya. “User Authentication Based on Keystroke Dynamics.” In: *IETE Journal of Research* 60 (2014), pages 229–239. https://doi.org/10.1080/03772063.2014.914686.

[21] T. Dias, J. Vitorino, E. Maia, O. Sousa, I. Praça. *RecKeys*. 2023. URL: https://zenodo.org/records/7886743.

[22] A. Dimaratos, D. Pöhn. “Evaluation Scheme to Analyze Keystroke Dynamics Methods.” In: *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP)*. SCITEPRESS, 2024. https://doi.org/10.48550/arXiv.2407.16247.

[23] Elsevier. *Scopus AI*. 2024. URL: https://www.scopus.com/search/form.uri?display=basic#basic (viewed 2025-01-02).

[24] R. Fadul, A. AlShehhi, L. Hadjileontiadis. “Robust remote detection of depressive tendency based on keystroke dynamics and behavioural characteristics.” In: *Scientific Reports* 14 (2024). https://doi.org/10.1038/s41598-024-78489-x.

[25] A. Géron. *Hands-on machine learning with scikit-learn and tensorflow: Concepts*. pages 175-208. Canada: O’Reilly Media, 2017.

[26] W. H. Gomaa, A. A. Fahmy. “A Survey of Text Similarity Approaches.” In: *International Journal of Computer Applications* 68 (2013), pages 13–18. https://doi.org/10.5120/11638-7118.

[27] Grammarly. *Grammarly*. 2024. URL: https://www.grammarly.com (viewed 2025-01-02).

[28] M. Hossin, S. M.N. “A Review on Evaluation Metrics for Data Classification Evaluations.” In: *International Journal of Data Mining & Knowledge Management Process* 5 (2015), pages 1–11. https://doi.org/10.5121/ijdkp.2015.5201.

[29] B. Hu, Y. Zhao, J. He, Q. Liu, R. Chen. “A Classification Method for Airborne Full-Waveform LiDAR Systems Based on a Gramian Angular Field and Convolution Neural Networks.” In: *Electronics* 11 (2022). https://doi.org/10.3390/electronics11244114.

[30] C. Katsini, M. Belk, C. Fidas, N. Avouris, G. Samaras. “Security and Usability in Knowledge-based User Authentication: A Review.” In: *Proceedings of the 20th Pan-Hellenic Conference on Informatics*. New York, NY, USA: Association for Computing Machinery, 2016, pages 1–6. https://doi.org/10.1145/3003733.3003764.

[31] K. Killourhy, R. Maxion. *Carnegue Mellon University Dataset*. 2009. URL: https://www.cs.cmu.edu/~keystroke/#sec2.

[32] K. Killourhy, R. Maxion. “Comparing Anomaly-Detection Algorithms for Keystroke Dynamics.” In: *IEEE/IFIP International Conference on Dependable Systems & Networks* (2009), pages 125–134. https://doi.org/10.1109/DSN.2009.5270346.

[33] V. Kotu, B. Deshpande. *Data Science*. Second. Chapter 11 – Recommendation Engines. United States: Elsevier, 2019.

[34] O. Kurasova, A. Budžys, V. Medvedev. *Exploring Multidimensional Embeddings for Decision Support Using Advanced Visualization Techniques*. MDPI, 2024. https://doi.org/10.3390/informatics11010011.

[35] K. Lis, E. Niewiadomska-Szynkiewicz, K. Dziewulska. “Siamese Neural Network for Keystroke Dynamics-Based Authentication on Partial Passwords.” In: *Sensors* 23 (2023), page 6685. https://doi.org/10.3390/s23156685.

[36] M. Liu, J. Guan. “User Keystroke Authentication Based on Convolutional Neural Network.” In: *Mobile Internet Security*. Springer Singapore, 2019, pages 157–168.

[37] B. Mahesh. “Machine Learning Algorithms -A Review.” In: *International Journal of Science and Research (IJSR)* 9 (2019). https://doi.org/10.21275/ART20203995.

[38] A. Mohammed, R. Kora. “A Comprehensive Review on Ensemble Deep Learning: Opportunities and Challenges.” In: *Journal of King Saud University - Computer and Information Sciences* 35 (2023). https://doi.org/10.1016/j.jksuci.2023.01.014.

[39] E. Monastyrska. “Naujienų pranešimų poveikis vertybinių popierių kainų pokyčiams.” Bachelor’s thesis. Vilnius University, 2023.

[40] F. Monrose, A. Rubin. “Authentication via Keystroke Dynamics.” In: *Proceedings of the ACM Conference on Computer and Communications Security* (2000), pages 48–56. https://doi.org/10.1145/266420.266434.

[41] A. C. Müller, S. Guido. *Introduction to machine learning with Python: a guide for data scientists*. pages 35-44. United States of America: O’Reilly Media, Inc., 2016.

[42] O. Oduntan, I. Adeyanju, F. A.s, O. Obe. “A Comparative Analysis of Euclidean Distance and Cosine Similarity Measure for Automated Essay-Type Grading.” In: *Journal of Engineering and Applied Sciences* 13 (2018), pages 4198–4204. https://doi.org/10.3923/jeasci.2018.4198.4204.

[43] OpenAI. *ChatGPT*. 2024. URL: https://www.openai.com/ (viewed 2025-01-02).

[44] M. Papathanasaki, L. Maglaras, N. Ayres. “Modern Authentication Methods: A Comprehensive Survey.” In: *AI, Computer Science and Robotics Technology* (2022), pages 1–24. https://doi.org/10.5772/acrt.08.

[45] S. P. Patel, S. Upadhyay. “Euclidean distance based feature ranking and subset selection for bearing fault diagnosis.” In: *Expert Systems with Applications* 154 (2020), page 113400. https://doi.org/https://doi.org/10.1016/j.eswa.2020.113400.

[46] H. Risto, O. Graven. “Fixed-text keystroke dynamics authentication data set—collection and analysis.” In: *Annals of Telecommunications* 79 (2024), pages 731–743. https://doi.org/10.1007/s12243-024-01039-z.

[47] S. Roy, J. Pradhan, A. Kumar, D. Adhikary, U. Roy, D. Sinha, R. Pal. “A Systematic Literature Review on Latest Keystroke Dynamics Based Models.” In: *IEEE Access* 10 (2022), pages 92192–92236. https://doi.org/10.1109/ACCESS.2022.3197756.

[48] C. Sahu, M. Banavar. “A nonlinear feature transformation-based multi-user classification algorithm for keystroke dynamics.” In: *2021 55th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2021, pages 1448–1452. https://doi.org/10.1109/IEEECONF53345.2021.9723223.

[49] Scikit-learn Developers. *DecisionTreeClassifier Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier (viewed 2025-01-02).

[50] Scikit-learn Developers. *GradientBoostingClassifier Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (viewed 2025-01-02).

[51] Scikit-learn Developers. *KNeighborsClassifier Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier (viewed 2025-01-02).

[52] Scikit-learn Developers. *RandomForestClassifier Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier (viewed 2025-01-01).

[53] Scikit-learn Developers. *RBF Kernel Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.gaussian_process.kernels.RBF.html (viewed 2025-01-02).

[54] Scikit-learn Developers. *SVC Documentation*. URL: https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (viewed 2025-01-02).

[55] Scikit-learn Developers. *XGBoost Documentation*. URL: https://xgboost.readthedocs.io/en/stable/ (viewed 2025-01-02).

[56] R. Shadman, A. Wahab, M. Manno, M. Lukaszewski, D. Hou, F. Hussain. "Keystroke Dynamics: Concepts, Techniques, and Applications." In: *arXiv* (2023). https://doi.org/10.48550/arXiv.2303.04605.

[57] D. Shanmugapriya, G. Padmavathi. "A Survey of Biometric keystroke Dynamics: Approaches, Security and Challenges." In: *International Journal of Computer Science and Information Security* 5 (2009), pages 115–119. https://doi.org/10.48550/arXiv.0910.0817.

[58] Y. Shi, X. Wang, K. Zheng, S. Cao. "User authentication method based on keystroke dynamics and mouse dynamics using HDA." In: *Multimedia Systems* 29 (2022), pages 1–16. https://doi.org/10.1007/s00530-022-00997-5.

[59] P. S. Teh, A. Teoh, S. Yue. "A Survey of Keystroke Dynamics Biometrics." In: *TheScientificWorldJournal* 2013 (2013), page 408280. https://doi.org/10.1155/2013/408280.

[60] K. P. Tripathi. "A Comparative Study of Biometric Technologies with Reference to Human Interface." In: *International Journal of Computer Applications* 14 (2011). https://doi.org/10.5120/1842-2493.

[61] A. Wahab, D. Hou, S. Schuckers, A. Barbir. "Securing Account Recovery Mechanism on Desktop Computers and Mobile Phones with Keystroke Dynamics." In: *SN Computer Science* 3 (2022). https://doi.org/10.1007/s42979-022-01245-3.

[62] X. Wang, D. Hou. "Enhancing Keystroke Dynamics Authentication with Ensemble Learning and Data Resampling Techniques." In: *Electronics* 13 (2024). https://doi.org/10.3390/electronics13224559.

[63] Y. Wang, H. Huang, C. Rudin, Y. Shaposhnik. "Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMap, and PaCMAP for Data Visualization." In: *Journal of Machine Learning Research* 22 (2020), pages 1–73. https://doi.org/10.48550/arXiv.2012.04456.

[64] Z. Wang, T. Oates. "Imaging Time-Series to Improve Classification and Imputation." In: *arXiv* (2015). https://doi.org/10.48550/arXiv.1506.00327.

[65] C. K. Williams. "On a Connection Between Kernel PCA and Metric Multidimensional Scaling." In: *Advances in Neural Information Processing Systems*. Volume 46. MIT Press, 2002, pages 11–19. https://doi.org/10.1023/A:1012485807823.

Appendix

Appendix A: Artificial intelligence

In this thesis, ChatGPT [43], Grammarly [27], Curie [19], and Scopus [23] were utilized to enhance the quality of the research.

ChatGPT was instrumental in assisting with technical writing, code optimization, and commenting. Additionally, it was used to efficiently create and modify LaTeX tables. For writing, it contributed significantly to improving the report structure, identifying grammar and punctuation errors, rephrasing sentences for better clarity, and suggesting synonyms to avoid repetitive word usage. In code optimization, ChatGPT provided valuable insights by identifying areas where performance and readability could be enhanced. It highlighted redundant or inefficient sections and offered concrete suggestions for improvement. For code commenting, ChatGPT identified parts of the code that required additional explanations to improve comprehensibility, ensuring the code was accessible to future reviewers or collaborators.

Grammarly was used to significantly enhance the quality of my written communication. It ensured that the writing was clear, grammatically correct, and well-structured. Furthermore, its tone suggestions and style enhancements improved the overall professionalism and readability of the content.

Curie played a crucial role in ensuring the coherence of my arguments. It excelled in handling complex academic language, summarizing detailed sections concisely, and suggesting improvements to enhance the logical flow of the report. By leveraging Curie, I achieved greater efficiency in refining technical sections and ensured that the thesis adhered to high academic standards.

Finally, Scopus was utilized to efficiently locate articles relevant to the analyzed topics. Its advanced search capabilities allowed for keyword-based searches, simplifying the process of finding high-quality and relevant literature for this research.

Appendix B: Supplementary figures

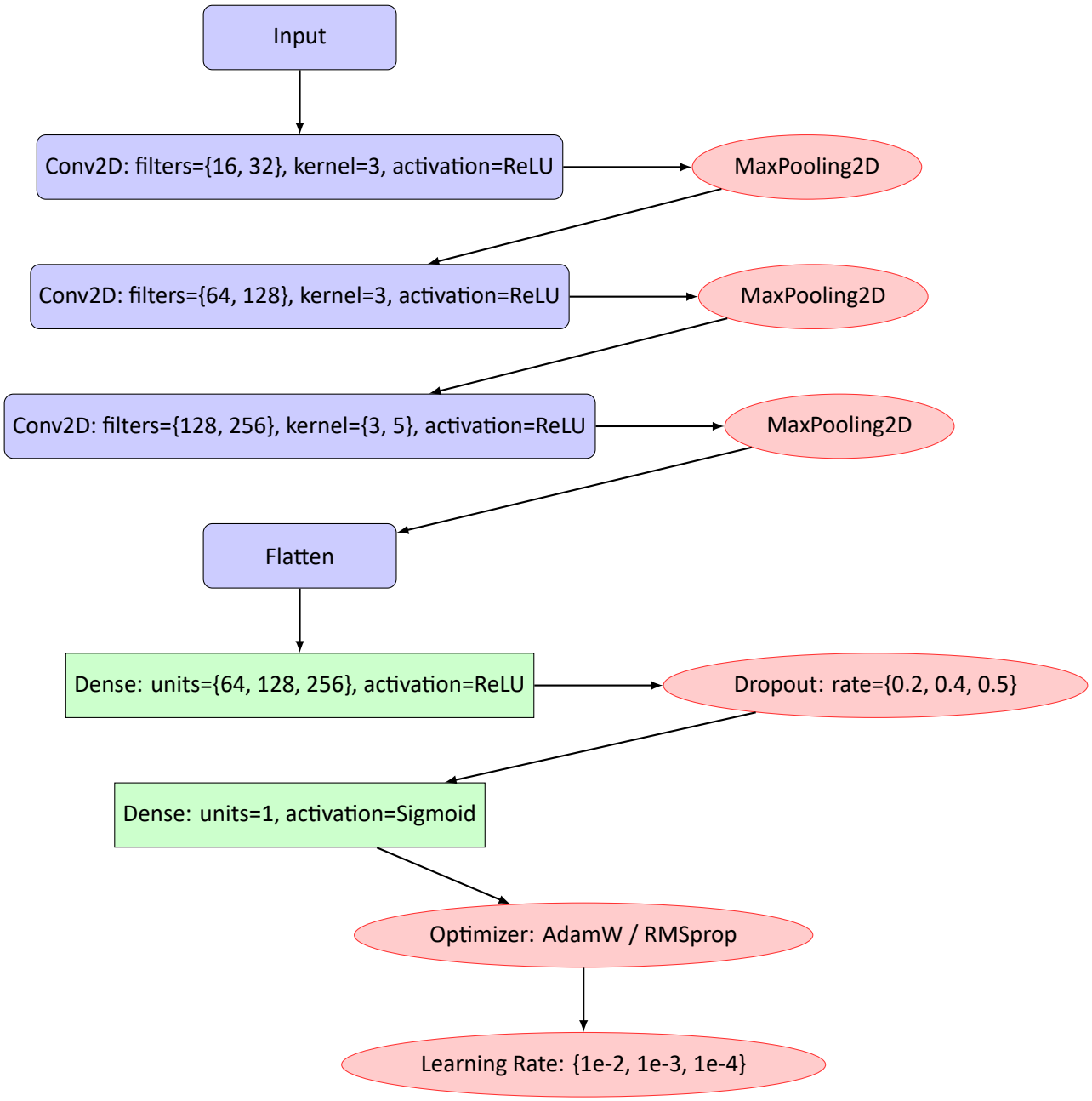


Figure B1: *Architecture of the convolutional neural network model*

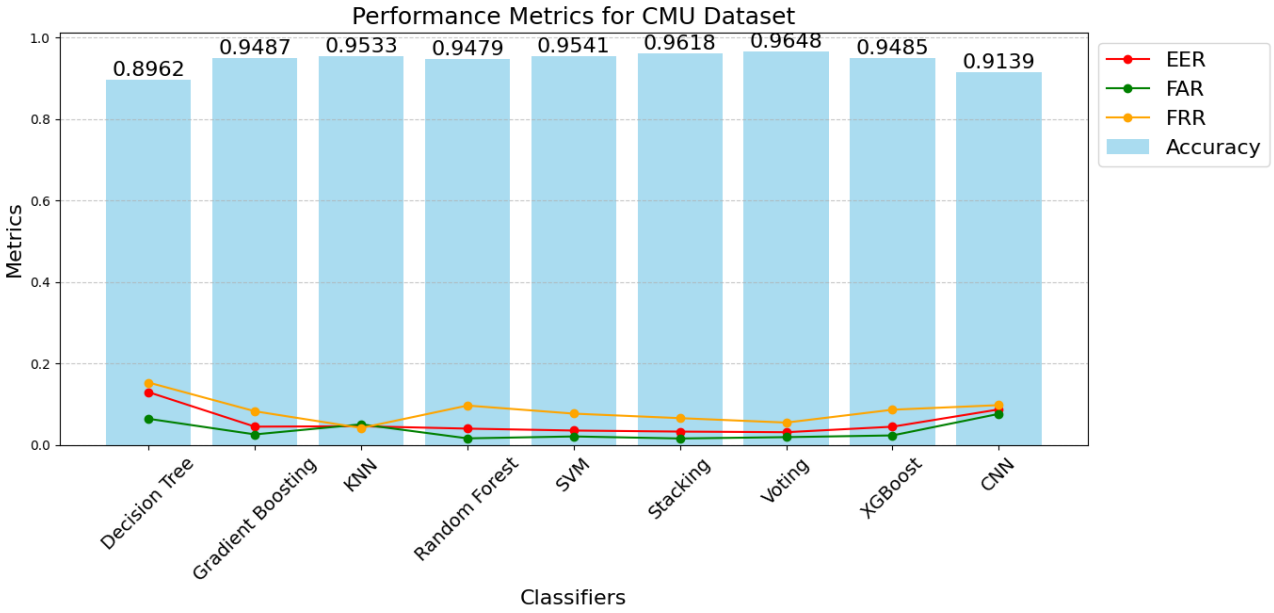


Figure B2: *CMU dataset performance metrics graph*

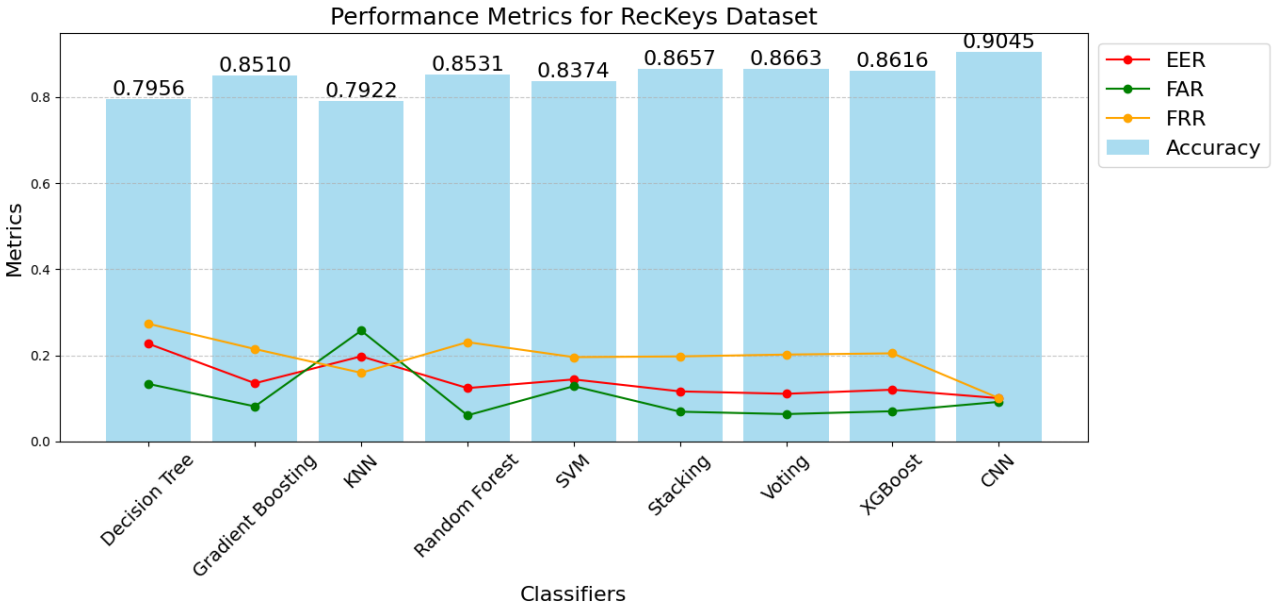


Figure B3: *KeyRecs dataset performance metrics graph for KeyRecs dataset*

Appendix C: Supplementary Tables

Table C1: *CMU keystroke timing statistics*

Feature	Mean	Std	Min	50%	Max
DD.Shift.r.o	0.251	0.174	0.049	0.201	4.152
DD.a.n	0.151	0.107	0.001	0.125	3.328
DD.e.five	0.377	0.265	0.001	0.289	4.962
DD.five.Shift.r	0.439	0.260	0.169	0.378	8.370
DD.i.e	0.158	0.137	0.001	0.121	2.655
DD.l.Return	0.322	0.225	0.008	0.263	5.884
DD.n.l	0.203	0.150	0.001	0.173	4.025
DD.o.a	0.157	0.107	0.001	0.132	2.857
DD.period.t	0.264	0.219	0.019	0.206	12.506
DD.t.i	0.169	0.123	0.001	0.140	4.920
DU.Shift.r.o	0.243	0.177	0.026	0.195	4.105
DU.a.n	0.134	0.104	-0.145	0.113	2.594
DU.e.five	0.365	0.265	-0.111	0.278	4.965
DU.five.Shift.r	0.458	0.263	0.171	0.395	8.373
DU.i.e	0.166	0.142	-0.031	0.129	2.658
DU.l.Return	0.315	0.228	0.035	0.249	5.887
DU.n.l	0.208	0.151	-0.084	0.188	4.045
DU.o.a	0.175	0.115	-0.122	0.147	2.944
DU.period.t	0.256	0.225	-0.076	0.194	12.524
DU.t.i	0.165	0.124	-0.141	0.137	4.899
H.Return	0.088	0.027	0.003	0.085	0.265
H.Shift.r	0.096	0.034	0.001	0.093	0.282
H.a	0.106	0.039	0.004	0.102	2.035
H.e	0.089	0.031	0.002	0.083	0.325
H.five	0.077	0.022	0.001	0.074	0.199
H.i	0.082	0.027	0.003	0.077	0.331
H.l	0.096	0.028	0.004	0.094	0.341
H.n	0.090	0.031	0.004	0.085	0.358
H.o	0.088	0.026	0.007	0.086	0.687
H.period	0.093	0.030	0.001	0.090	0.376
H.t	0.086	0.027	0.009	0.081	0.241
UD.Shift.r.o	0.155	0.182	-0.087	0.102	4.012
UD.a.n	0.044	0.105	-0.236	0.023	2.524
UD.e.five	0.288	0.267	-0.151	0.200	4.883
UD.five.Shift.r	0.362	0.261	0.086	0.302	8.291
UD.i.e	0.077	0.140	-0.160	0.041	2.586

Continued on the next page

Table C1: *CMU keystroke data summary (continued)*

Keystroke	Mean	Std	Min	50%	Max
UD.l.Return	0.226	0.231	-0.125	0.160	5.836
UD.n.l	0.113	0.160	-0.176	0.096	3.978
UD.o.a	0.069	0.109	-0.229	0.044	2.815
UD.period.t	0.171	0.226	-0.236	0.109	12.452
UD.t.i	0.083	0.126	-0.162	0.058	4.800
UU.Shift.r.o	0.243	0.177	0.026	0.195	4.105
UU.a.n	0.134	0.104	-0.145	0.113	2.594
UU.e.five	0.365	0.265	-0.111	0.278	4.965
UU.five.Shift.r	0.458	0.263	0.171	0.395	8.373
UU.i.e	0.166	0.142	-0.031	0.129	2.658
UU.l.Return	0.315	0.228	0.035	0.249	5.887
UU.n.l	0.208	0.151	-0.084	0.188	4.045
UU.o.a	0.175	0.115	-0.122	0.147	2.944
UU.period.t	0.256	0.225	-0.076	0.194	12.524
UU.t.i	0.165	0.124	-0.141	0.137	4.899
tri.Shift.r.o.a	0.408	0.232	0.126	0.335	4.319
tri.a.n.l	0.353	0.218	0.066	0.295	4.259
tri.e.five.Shift.r	0.816	0.442	0.254	0.714	11.317
tri.five.Shift.r.o	0.690	0.376	0.252	0.588	11.031
tri.i.e.five	0.536	0.330	0.087	0.437	5.128
tri.n.l.Return	0.524	0.319	0.108	0.435	7.796
tri.o.a.n	0.308	0.175	0.084	0.258	3.659
tri.period.t.i	0.433	0.276	0.138	0.357	12.684
tri.t.i.e	0.327	0.208	0.085	0.264	6.013

Table C2: *KeyRecs keystroke timing statistics*

Feature	Mean	Std	Min	50%	Max
DD.e.u	0.245	0.259	0.005	0.185	14.772
DD.j.k	0.256	0.261	0.001	0.197	8.861
DD.k.b	0.277	0.213	0.001	0.232	5.730
DD.k.e	0.293	0.350	0.003	0.201	16.554
DD.p.w	0.392	0.436	0.004	0.255	18.715
DD.r.k	0.445	0.422	0.007	0.310	11.384
DD.u.r	0.221	0.208	0.002	0.173	4.981
DD.v.p	0.259	0.246	0.001	0.199	7.466
DD.w.j	0.436	0.514	0.001	0.282	19.503
DU.e.u	0.337	0.262	0.008	0.278	14.871
DU.j.k	0.356	0.260	0.006	0.295	8.990

Continued on the next page

Table C2: *KeyRecs keystroke timing statistics (continued)*

Keystroke	Mean	Std	Min	50%	Max
DU.k.b	0.367	0.217	0.041	0.323	5.784
DU.k.e	0.398	0.351	0.006	0.312	16.642
DU.p.w	0.495	0.437	0.007	0.364	18.843
DU.r.k	0.540	0.422	0.014	0.410	11.465
DU.u.r	0.318	0.212	0.006	0.270	5.056
DU.v.p	0.355	0.247	0.004	0.296	7.542
DU.w.j	0.533	0.519	0.006	0.382	19.763
H.b.b	0.090	0.043	0.002	0.085	4.573
H.e.e	0.105	0.031	0.002	0.101	0.294
H.j.j	0.096	0.041	0.002	0.085	0.548
H.k.k	0.100	0.030	0.003	0.095	0.333
H.k.k.1	0.095	0.029	0.004	0.090	0.292
H.p.p	0.096	0.029	0.002	0.090	0.250
H.r.r	0.096	0.028	0.002	0.093	0.321
H.u.u	0.092	0.028	0.002	0.087	0.266
H.v.v	0.098	0.030	0.002	0.094	0.965
H.w.w	0.103	0.031	0.003	0.098	0.448
UD.e.u	0.141	0.259	-0.111	0.080	14.625
UD.j.k	0.160	0.266	-0.179	0.118	8.749
UD.k.b	0.182	0.214	-0.147	0.140	5.677
UD.k.e	0.193	0.349	-0.144	0.100	16.458
UD.p.w	0.296	0.436	-0.104	0.160	18.624
UD.r.k	0.349	0.423	-0.148	0.210	11.332
UD.u.r	0.130	0.206	-0.100	0.080	4.900
UD.v.p	0.161	0.247	-0.172	0.098	7.391
UD.w.j	0.333	0.514	-0.145	0.178	19.384
UU.e.u	0.232	0.260	-0.045	0.170	14.724
UU.j.k	0.260	0.263	-0.075	0.209	8.878
UU.k.b	0.272	0.215	-0.063	0.228	5.731
UU.k.e	0.298	0.348	-0.012	0.211	16.546
UU.p.w	0.399	0.436	0.004	0.265	18.752
UU.r.k	0.444	0.421	-0.051	0.310	11.413
UU.u.r	0.226	0.208	0.001	0.176	4.975
UU.v.p	0.257	0.247	-0.061	0.196	7.467
UU.w.j	0.429	0.518	-0.073	0.279	19.644
tri.e.u.r	0.467	0.357	0.008	0.377	14.925
tri.j.k.e	0.549	0.441	0.006	0.428	17.050
tri.k.e.u	0.538	0.448	0.008	0.433	16.658
tri.p.w.j	0.828	0.711	0.008	0.619	20.161

Continued on the next page

Table C2: *KeyRecs keystroke timing statistics (continued)*

Keystroke	Mean	Std	Min	50%	Max
tri.r.k.b	0.722	0.469	0.091	0.612	11.741
tri.u.r.k	0.667	0.487	0.013	0.536	11.781
tri.v.p.w	0.651	0.526	0.011	0.504	18.893
tri.w.j.k	0.693	0.588	0.007	0.526	19.655

Table C3: *Hyperparameter tunning*

Hyperparameter	Model	Values
n_neighbors	KNN	[3, 5, 7]
weights	KNN	['uniform', 'distance']
p	KNN	[1, 2, 3]
C	SVM	[0.1, 1, 10]
kernel	SVM	['linear', 'rbf']
degree	SVM	[2,3,4]
n_estimator	RF, XGBoost, GB	[100, 200, 300]
max_depth	RF, DT	[None, 10, 20]
criterion	DT	['gini', 'entropy', 'log_loss']
min_samples_split	DT	[2, 5, 10]
max_depth	XGBoost	[4, 6, 8]
learning_rate	XGBoost	[0.2, 0.3, 0.4]
learning_rate	GB	[0.01, 0.1, 0.2]

Table C4: *Comparison of model performance on original and extended CMU datasets*

Model	Dataset Type	Accuracy	EER
DT	Extended with transformation	0.895	0.130
DT	Extended without transformation	0.896	0.129
DT	Original with transformation	0.896	0.130
DT	Original without transformation	0.894	0.132
GB	Extended with transformation	0.948	0.045
GB	Extended without transformation	0.947	0.046
GB	Original with transformation	0.946	0.046
GB	Original without transformation	0.945	0.046
KNN	Extended with transformation	0.953	0.046
KNN	Extended without transformation	0.910	0.080
KNN	Original with transformation	0.958	0.041
KNN	Original without transformation	0.921	0.064
RF	Extended with transformation	0.947	0.041
RF	Extended without transformation	0.948	0.040
RF	Original with transformation	0.950	0.038
RF	Original without transformation	0.950	0.039
SVM	Extended with transformation	0.954	0.036
SVM	Extended without transformation	0.939	0.058
SVM	Original with transformation	0.954	0.037
SVM	Original without transformation	0.938	0.060
XGB	Extended with transformation	0.949	0.045
XGB	Extended without transformation	0.948	0.045
XGB	Original with transformation	0.947	0.045
XGB	Original without transformation	0.947	0.045
Stacking	Extended with transformation	0.961	0.033
Stacking	Extended without transformation	0.952	0.040
Stacking	Original with transformation	0.962	0.032
Stacking	Original without transformation	0.954	0.038
Voting	Extended with transformation	**0.965**	0.032
Voting	Extended without transformation	**0.957**	0.042
Voting	Original with transformation	0.965	0.032
Voting	Original without transformation	0.959	0.038

Table C5: *Comparison of model performance on original and extended KeyRecs datasets*

Model	Dataset Type	Accuracy	EER	FAR	FRR
DT	Extended with transformation	0.8071	0.2738	0.1101	0.3529
DT	Extended without transformation	0.8026	0.2680	0.1098	0.3667
DT	Original with transformation	0.8060	0.2590	0.1118	0.3532
DT	Original without transformation	0.8132	0.1391	0.1096	0.3361
GB	Extended with transformation	0.8630	0.1381	0.0534	0.2984
GB	Extended without transformation	0.8647	0.1349	0.0524	0.2957
GB	Original with transformation	0.8679	0.1356	0.0569	0.2775
GB	Original without transformation	0.8670	0.1362	0.0565	0.2808
KNN	Extended with transformation	0.8685	0.2163	0.1046	0.1836
KNN	Extended without transformation	0.7959	0.1227	0.1640	0.2816
KNN	Original with transformation	0.8801	0.1948	0.1023	0.1540
KNN	Original without transformation	0.8108	0.1469	0.1616	0.2427
RF	Extended with transformation	0.8547	0.1443	0.0371	0.3543
RF	Extended without transformation	0.8582	0.1411	0.0366	0.3451
RF	Original with transformation	0.8604	0.1408	0.0358	0.3401
RF	Original without transformation	0.8596	0.1205	0.0392	0.3361
SVM	Extended with transformation	0.8675	0.1581	0.0498	0.2924
SVM	Extended without transformation	0.8333	0.1115	0.0905	0.3140
SVM	Original with transformation	0.8761	0.1611	0.0453	0.2760
SVM	Original without transformation	0.8292	0.1376	0.0891	0.3287
XGB	Extended with transformation	0.8664	0.1378	0.0493	0.2967
XGB	Extended without transformation	0.8655	0.1423	0.0501	0.2975
XGB	Original with transformation	0.8610	0.1423	0.0550	0.3013
XGB	Original without transformation	0.8610	0.1181	0.0550	0.3013
Stacking	Extended with transformation	0.8794	0.1381	0.0392	0.2779
Stacking	Extended without transformation	0.8675	0.1130	0.0433	0.3049
Stacking	Original with transformation	0.8830	0.1353	0.0412	0.2635
Stacking	Original without transformation	0.8691	0.1190	0.0441	0.2986
Voting	Extended with transformation	0.8809	0.1496	0.0438	0.2646
Voting	Extended without transformation	0.8587	0.1125	0.0443	0.3288
Voting	Original with transformation	**0.8852**	0.1450	0.0473	0.2454
Voting	Original without transformation	0.8635	0.1450	0.0426	0.3181

Table C6: *KeyRecs dataset CNN results for each subject*

Subject	Acc. (%)	EER	FAR	FRR	Subject	Acc. (%)	EER	FAR	FRR
1	93	0.0773	0.0825	0.065	52	83	0.175	0.1237	0.21
2	91	0.095	0.0515	0.12	53	96	0.0464	0.0773	0.0102

Table C6: *KeyRecs dataset CNN results for each subject (continued)*

Subject	Acc. (%)	EER	FAR	FRR	Subject	Acc. (%)	EER	FAR	FRR
3	99	0.0103	0.0103	0	54	96	0.0412	0.0309	0.0556
5	78	0.2165	0.1495	0.28	55	80	0.22	0.1237	0.28
6	94	0.0619	0.0773	0.045	56	96	0.038	0.056	0.025
7	80	0.18	0.2474	0.145	57	97	0.03	0.0361	0.03
8	98	0.015	0.0206	0.015	58	91	0.1082	0.1392	0.05
9	77	0.2268	0.1959	0.27	59	87	0.13	0.1289	0.14
10	86	0.1392	0.1392	0.135	60	97	0.0309	0.0515	0.01
11	88	0.1263	0.0619	0.1818	61	98	0.0309	0.0103	0.035
12	97	0.0309	0.0464	0.0102	62	90	0.134	0.1598	0.05
13	83	0.1134	0.0258	0.315	63	85	0.1495	0.1598	0.1414
14	93	0.0707	0.0876	0.0606	64	95	0.05	0.0412	0.055
15	96	0.0412	0.0464	0.03	65	88	0.115	0.078	0.1625
16	86	0.135	0.1701	0.105	66	91	0.085	0.066	0.1125
17	93	0.067	0.1082	0.03	67	96	0.0412	0.0412	0.045
18	93	0.08	0.0928	0.055	68	82	0.1753	0.1649	0.195
19	76	0.2268	0.2629	0.22	69	89	0.1443	0.1804	0.05
20	91	0.0909	0.0722	0.1162	70	71	0.1598	0.0515	0.515
21	99	0.0103	0.0103	0	71	88	0.0979	0.0515	0.1818
22	92	0.0876	0.0515	0.11	72	93	0.0825	0.1186	0.03
23	89	0.12	0.17	0.06	73	84	0.1598	0.1031	0.2092
24	90	0.135	0.0464	0.155	74	89	0.12	0.1082	0.12
25	81	0.1804	0.2371	0.1465	75	93	0.0567	0.0979	0.04
26	96	0.0412	0.0412	0.0455	76	91	0.095	0.0876	0.095
27	96	0.0361	0.0361	0.035	77	99	0.02	0.0052	0.02
28	83	0.1598	0.2165	0.12	78	96	0.464	0.0309	0.055
29	80	0.1856	0.1186	0.285	79	95	0.0412	0.0722	0.025
30	86	0.1443	0.1753	0.11	80	94	0.0567	0.0412	0.07
31	95	0.0515	0.0464	0.06	81	80	0.2062	0.3557	0.055
32	89	0.125	0.1649	0.055	82	83	0.1869	0.2784	0.0707
33	91	0.0979	0.0515	0.1364	83	98	0.0309	0.0309	0
34	97	0.0361	0.0206	0.04	84	88	0.1495	0.1804	0.06
35	96	0.0455	0.0567	0.0202	85	96	0.0354	0.0155	0.0556
36	94	0.08	0.0464	0.08	86	98	0.0309	0.0052	0.035
37	88	0.12	0.1546	0.095	87	88	0.1082	0.0619	0.1717
38	87	0.13	0.1186	0.14	88	94	0.06	0.0104	0.16
39	99	0.015	0.0052	0.02	89	83	0.1649	0.1443	0.1919
40	96	0.0361	0.0464	0.0303	90	88	0.1546	0.1959	0.05
41	86	0.145	0.0933	0.22	91	98	0.025	0.0155	0.025
42	98	0.0204	0.0155	0.0204	92	97	0.0309	0.0309	0.0253

Table C6: *KeyRecs dataset CNN results for each subject (continued)*

Subject	Acc. (%)	EER	FAR	FRR	Subject	Acc. (%)	EER	FAR	FRR
43	100	0	0.0052	0	93	94	0.07	0.0361	0.09
44	97	0.0258	0.0515	0.01	94	93	0.065	0.0515	0.08
45	73	0.3041	0.0773	0.455	95	87	0.1443	0.1443	0.125
46	91	0.0825	0.1598	0.03	96	92	0.0825	0.0825	0.085
47	98	0.0206	0.0206	0.0108	97	98	0.025	0.0052	0.035
48	93	0.0725	0.066	0.075	98	80	0.1443	0.0825	0.305
49	86	0.1392	0.1856	0.09	99	85	0.1546	0.134	0.175
50	82	0.1818	0.1959	0.1717	100	85	0	0.0052	0
51	95	0.0606	0.0722	0.0253					

Table C7: *CMU dataset CNN results for each subject*

Subject	Acc. (%)	EER	FAR	FRR	Subject	Acc. (%)	EER	FAR	FRR
2	81	0.1825	0.2220	0.1500	30	97	0.0350	0.0400	0.0350
3	98	0.0125	0.0120	0.0200	31	76	0.2500	0.1520	0.3350
4	91	0.1100	0.1120	0.0625	32	82	0.1940	0.0860	0.3075
5	95	0.0460	0.0400	0.0600	33	88	0.1200	0.0520	0.2075
7	95	0.0525	0.0420	0.0700	35	88	0.1200	0.1180	0.1250
8	94	0.0580	0.0520	0.0725	36	88	0.1200	0.1750	0.1075
10	96	0.0340	0.0160	0.0050	37	86	0.1300	0.0140	0.0425
11	99	0.0100	0.0020	0.0175	39	97	0.0325	0.2740	0.1100
12	99	0.0100	0.0020	0.0175	40	80	0.2075	0.0160	0.0375
13	93	0.0740	0.0760	0.0550	41	97	0.0300	0.0560	0.0075
15	78	0.2275	0.1760	0.2700	43	96	0.0175	0.1000	0.0375
16	74	0.3075	0.1060	0.4600	44	93	0.0800	0.1560	0.1800
18	89	0.1100	0.0120	0.1025	46	93	0.1675	0.0960	0.0900
19	96	0.0640	0.0680	0.0425	47	94	0.0550	0.0540	0.0725
20	93	0.0675	0.0300	0.1125	48	91	0.0580	0.0660	0.1125
21	87	0.1320	0.1180	0.1400	50	99	0.0850	0.0120	0.0100
22	89	0.1080	0.1100	0.1000	51	100	0.0120	0.0120	0.0000
24	90	0.1025	0.1260	0.0700	52	88	0.0000	0.0000	0.0050
25	92	0.0775	0.1080	0.0425	53	99	0.0075	0.0120	0.0050
27	87	0.1380	0.0680	0.2000	55	97	0.0400	0.0200	0.0200
28	94	0.1780	0.0080	0.2425	56	94	0.0600	0.0420	0.0675

Appendix D: Code

```
import pandas as pd
from sklearn.manifold import MDS, TSNE
from sklearn.preprocessing import StandardScaler
from umap import UMAP
from sklearn.decomposition import KernelPCA
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset from the CSV file
df = pd.read_csv('DSL-StrongPasswordData.csv')
# Define the password used to extract features
password = '.tie5Roanl_'
# Initialize lists to store feature column names
listofUp = []  # Holds UU (key release) features
listofUpDown = [col for col in df.columns if col.startswith('UD')]  # UD (key
↪  press-to-release) features
listofHold = [col for col in df.columns if col.startswith('H')]  # H (key hold
↪  duration) features
listofDown = [col for col in df.columns if col.startswith('DD')]  # DD (key
↪  press-to-press) features
# Generate UU (key release) features based on the password
for i in range(len(password) - 1):
    up = 'UU.'
    current_symbol = password[i]
    next_symbol = password[i + 1]
    # Handle specific cases for symbols and append to listofUp
    if current_symbol == '.':
        listofUp.append(f'{up}period.{next_symbol}')
    elif current_symbol == '5' and next_symbol == 'R':
        listofUp.append(f'{up}five.Shift.r')
    elif next_symbol == '5':
        listofUp.append(f'{up}{current_symbol}.five')
    elif current_symbol == 'R':
        listofUp.append(f'{up}Shift.r.{next_symbol}')
    elif next_symbol == 'R':
        listofUp.append(f'{up}{current_symbol}.Shift.r')
    elif next_symbol == '_':
```

```
        listofUp.append(f'{up}{current_symbol}.Return')
    else:
        listofUp.append(f'{up}{current_symbol}.{next_symbol}')
# Define trigraph features (sequence of three keys pressed in order)
trigraph = [
    'tri.period.t.i', 'tri.t.i.e', 'tri.i.e.five', 'tri.e.five.Shift.r',
    'tri.five.Shift.r.o', 'tri.Shift.r.o.a', 'tri.o.a.n', 'tri.a.n.l',
    'tri.n.l.Return'
]
# Calculate new UU features using UD and H features
for i in range(len(listofUpDown)):
    df[listofUp[i]] = df[listofUpDown[i]] + df[listofHold[i + 1]]
# Generate trigraph features using DD features
for j in range(len(listofDown) - 1):
    df[trigraph[j]] = df[listofDown[j]] + df[listofDown[j + 1]]
# Define Down-Up (DU) features
DownUp = [
    'DU.period.t', 'DU.t.i', 'DU.i.e', 'DU.e.five', 'DU.five.Shift.r',
    'DU.Shift.r.o', 'DU.o.a', 'DU.a.n', 'DU.n.l', 'DU.l.Return'
]
# Calculate Down-Up (DU) features using UD and H features
for j in range(len(listofDown)):
    df[DownUp[j]] = df[listofUpDown[j]] + df[listofHold[j + 1]]
# Extract numerical subject ID from the 'subject' column and add as a new column
df['subject_num'] = df['subject'].str.extract('(\d+)').astype(int)
# Save the modified DataFrame to a new CSV file
df.to_csv('modified_cmu.csv', index=False)
################################################
###########Explanatory data analysis##########
################################################
pd.set_option('display.max_rows', None)
# Descriptive statistics after cleaning data from illogical values
descriptive_stats = df[df.columns.difference(['subject', 'session', 'repetition',
↪  'subject_num'])].describe().T
descriptive_stats = descriptive_stats[['mean', 'std', 'min', '50%', 'max']]
print(descriptive_stats)
################################################
#####################Plots######################
################################################
# Prepare lists of features
listofUpDown = [col for col in df.columns if col.startswith('UD')]
listofHold = [col for col in df.columns if col.startswith('H')]
listofDown = [col for col in df.columns if col.startswith('DD')]
```

```
# Features to use for similarity and distance computations
features = df.columns.difference(['subject', 'sessionIndex', 'rep', 'subject_num'])
# Get unique subjects
subjects = df['subject'].unique()
results = []
# Compute similarity and distance metrics between subjects
for i in range(len(subjects)):
    for j in range(i + 1, len(subjects)):
        subj1 = subjects[i]
        subj2 = subjects[j]
        # Extract subject data
        subject1 = df[df['subject'] == subj1]
        subject2 = df[df['subject'] == subj2]
        # Calculate feature vector means
        avg_vector1 = subject1[features].mean().values
        avg_vector2 = subject2[features].mean().values
        # Compute similarity and distance metrics
        cos_sim = cosine_similarity(avg_vector1.reshape(1, -1),
        ↪  avg_vector2.reshape(1, -1))[0][0]
        euclidean_dist = euclidean(avg_vector1, avg_vector2)
        results.append({
            'Subject1': subj1,
            'Subject2': subj2,
            'Cosine Similarity': cos_sim,
            'Euclidean Distance': euclidean_dist
        })
# Create results DataFrame
results_df = pd.DataFrame(results)
# Function to find maximum and minimum values for a metric
def similar(name, results_df):
    max_val = results_df[name].max()
    min_val = results_df[name].min()
    print(f"\nMax value: \n{results_df[results_df[name] == max_val]}:")
    print(f"\nMin value: \n{results_df[results_df[name] == min_val]}:")
# Print max/min results for each metric
for metric in ['Cosine Similarity', 'Euclidean Distance']:
    print(f"\nResults for {metric}:")
    similar(metric, results_df)
# Function to plot similarity and distance heatmaps
def plot_similarities(results_df):
    fig, axes = plt.subplots(1, 2, figsize=(25, 20))
    # Cosine Similarity Heatmap
```

```
    pivot_cosine = results_df.pivot(index='Subject1', columns='Subject2',
    ↪  values='Cosine Similarity')
    sns.heatmap(
        pivot_cosine, annot=False, cmap='coolwarm',
        cbar_kws={'label': 'Cosine Similarity'}, ax=axes[0]
    )
    axes[0].set_title('Cosine Similarity Between Subjects', fontsize=16)
    axes[0].tick_params(axis='x', labelrotation=90, labelsize=12)
    axes[0].tick_params(axis='y', labelrotation=0, labelsize=12)
    axes[0].set_xlabel('Subject2', fontsize=14)
    axes[0].set_ylabel('Subject1', fontsize=14)
    # Euclidean Distance Heatmap
    pivot_euclidean = results_df.pivot(index='Subject1', columns='Subject2',
    ↪  values='Euclidean Distance')
    sns.heatmap(
        pivot_euclidean, annot=False, cmap='coolwarm',
        cbar_kws={'label': 'Euclidean Distance'}, ax=axes[1]
    )
    axes[1].set_title('Euclidean Distance Between Subjects', fontsize=16)
    axes[1].tick_params(axis='x', labelrotation=90, labelsize=12)
    axes[1].tick_params(axis='y', labelrotation=0, labelsize=12)
    axes[1].set_xlabel('Subject2', fontsize=14)
    axes[1].set_ylabel('Subject1', fontsize=14)

    # Add legends for both subplots
    cbar_cosine = axes[0].collections[0].colorbar
    cbar_cosine.ax.tick_params(labelsize=12)
    cbar_cosine.set_label('Cosine Similarity Scale', fontsize=14)
    cbar_euclidean = axes[1].collections[0].colorbar
    cbar_euclidean.ax.tick_params(labelsize=12)
    cbar_euclidean.set_label('Euclidean Distance Scale', fontsize=14)

    plt.tight_layout()
    plt.show()
# Plot similarity and distance graphs for a subset of subjects
unique_subjects = results_df['Subject1'].unique()
# Dynamically subset subjects
subjects_subset = unique_subjects[0:31]
subjects_subset = np.append(subjects_subset, unique_subjects[44:49])
# Filter results for both Subject1 and Subject2
subset_df = results_df[
    results_df['Subject1'].isin(subjects_subset) &
    ↪  results_df['Subject2'].isin(subjects_subset)
```

```
]
# Plot the similarities
plot_similarities(subset_df)

################Timing#########################
def plot_subject_data(subject_data, subject_of_interest, session_number,
↪  repetition_number, normalized=False):
    # Select columns starting with H or UD
    filtered_columns = subject_data.filter(regex='^(H|UD)').columns
    # Check if non-empty dataframe with info about subject, session and repetition
    if not subject_data.empty:
        # Flaten the timing values into a 1D array for cumulative sum
        timing_values = subject_data[filtered_columns].values.flatten()
        # Calculate cumulative sum
        x_values = np.cumsum(timing_values)
        # Normalization for graph with normalized data (in 0 to 1 range)
        if normalized:
            x_values = (x_values - x_values[0]) / (x_values[-1] - x_values[0])
        # Label containing subject, session, repetition
        y_value = f'{subject_of_interest}, Session: {session_number}, Repetition:
        ↪  {repetition_number}'
        return x_values, [y_value] * len(x_values)
    return None, None
def plot_keystroke_timing(df, subject_combinations, normalized=False):
    plt.figure(figsize=(12, 8))
    # Loop over each subject combination
    for subject_of_interest, session_number, repetition_number in
    ↪  subject_combinations:
        # Filter the data for the specific subject, session, and repetition
        subject_data = df[(df['subject'] == subject_of_interest) &
                          (df['sessionIndex'] == session_number) &
                          (df['rep'] == repetition_number)]
        # Get the timing values for plotting
        x_values, y_values = plot_subject_data(subject_data, subject_of_interest,
        ↪  session_number, repetition_number, normalized=normalized)
        if x_values is not None:
            plt.plot(x_values, y_values, marker='o', linestyle='-', linewidth=4)
            plt.scatter(x_values, y_values, s=100)
    normalization_text = "Normalized" if normalized else "Non-Normalized"
    plt.title(f'{normalization_text} Keystroke Timing for Selected Subjects',
    ↪  fontsize = 20)
    plt.xlabel('Timing (Fraction of Typing Time)' if normalized else 'Timing
    ↪  (Cumulative Time)', fontsize = 18)
```

```
    plt.ylabel('Subject, Session, and Repetition', fontsize = 18)
    plt.xticks(fontsize = 14)
    plt.yticks(fontsize = 14)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
# Define all combinations of subject, session, and repetition
subject_combinations = [
    ('s007', 4, 47), ('s051', 8, 10),
    ('s018', 1, 15), ('s029', 3, 20),
]
# Plot normalized data
plot_keystroke_timing(df, subject_combinations, normalized=True)
# Plot non-normalized data
plot_keystroke_timing(df, subject_combinations, normalized=False)
###################Linear#######################
# Define specific subject pairs for comparison
subject_pairs = [
    ['s036', 's055'],
    ['s018', 's029'],
    ['s002', 's052'],
    ['s007', 's051']
]
# Extract all unique subjects from the defined pairs
all_subjects = [subject for pair in subject_pairs for subject in pair]
# Filter the DataFrame to include only rows corresponding to the selected subjects
df_filtered = df[df['subject'].isin(all_subjects)]
# Compute the mean of each feature for each subject
mean_features_per_subject = df_filtered.groupby('subject').mean()
# Function to plot Hold and Up-Down features side-by-side for each pair of subjects
def plot_side_by_side_hold_updown(df, subject_pairs, hold_features,
↪   updown_features):
    nrows = 2
    ncols = 4
    fig, axes = plt.subplots(nrows, ncols, figsize=(12, 18))
    axes = axes.flatten()
    # Iterate over each pair of subjects
    for i, pair in enumerate(subject_pairs):
        hold_ax = axes[i * 2]
        updown_ax = axes[i * 2 + 1]
        # Plot Hold features for each subject in the pair
        for subject in pair:
            hold_values = df.loc[subject, hold_features]
```

```
            hold_ax.plot(
                hold_values.index, hold_values.values,
                marker='o', linestyle='-', label=f'{subject}'
            )
        # Customize the Hold features plot
        hold_ax.set_xticks(range(len(hold_features)))
        hold_ax.set_xticklabels(hold_features, rotation=45, ha='right', fontsize=12)
        hold_ax.set_title(f'Hold Features: {pair[0]} vs {pair[1]}', fontsize=14)
        hold_ax.set_ylabel('Value', fontsize=12)
        hold_ax.legend(title='Subject', fontsize=12, loc='best')
        # Plot Up-Down features for each subject in the pair
        for subject in pair:
            updown_values = df.loc[subject, updown_features]
            updown_ax.plot(
                updown_values.index, updown_values.values,
                marker='o', linestyle='-', label=f'{subject}'
            )
        updown_ax.set_xticks(range(len(updown_features)))
        updown_ax.set_xticklabels(updown_features, rotation=45, ha='right',
        ↪  fontsize=12)
        updown_ax.set_title(f'Up-Down Features: {pair[0]} vs {pair[1]}',
        ↪  fontsize=14)
        updown_ax.set_ylabel('Value', fontsize=12)
        updown_ax.legend(title='Subject', fontsize=12, loc='upper left')
    plt.tight_layout()
    plt.show()
# Plot Hold and Up-Down features for the defined subject pairs
plot_side_by_side_hold_updown(mean_features_per_subject, subject_pairs, listofHold,
↪  listofUpDown)
################Dimension reduction################
# Define pairs of subject numbers to compare
user_pairs = [[36, 55], [18, 29], [2, 52], [7, 51]]
fig, axes = plt.subplots(len(user_pairs), 4, figsize=(24, 10))
# Iterate through each row (user pair) in the grid
for row, pair in enumerate(user_pairs):
    # Filter the DataFrame to include only the two subjects in the current pair
    df_two_users = df[df['subject_num'].isin(pair)]
    # Extract feature matrix (X) and target labels (y)
    # Drop non-feature columns like 'subject', 'rep', 'subject_num', and
    ↪  'sessionIndex'
    X = df_two_users.drop(columns=['subject', 'rep', 'subject_num', 'sessionIndex'],
    ↪  errors='ignore').values
    y = df_two_users['subject_num'].values
```

```
    # Standardize the feature matrix (mean=0, variance=1)
    X = StandardScaler().fit_transform(X)
    # Define dimensionality reduction methods with their configurations
    projection_methods = {
        'Kernel PCA': KernelPCA(n_components=2, kernel='rbf'),
        'Umap': UMAP(n_components=2, n_neighbors=10, min_dist=0.01),
        'Metric MDS': MDS(n_components=2, metric=True),
        't-SNE': TSNE(n_components=2, perplexity=3, learning_rate=200, n_iter=3500),
    }
    # Map subject numbers to colors for the scatter plot
    color_map = {pair[0]: 'orange', pair[1]: 'blue'}
    colors = np.array([color_map[num] for num in y])
    for col, (name, method) in enumerate(projection_methods.items()):
        # Apply the dimensionality reduction method
        transformed = method.fit_transform(X)
        # Plot the transformed data in the corresponding subplot
        axes[row, col].scatter(
            transformed[:, 0], transformed[:, 1], c=colors, s=25, alpha=0.6
        )
        axes[row, col].set_title(f"{name} for subjects {pair[0]} and {pair[1]}",
        ↪ fontsize=14)
        axes[row, col].set_xticks([]), axes[row, col].set_yticks([])
plt.tight_layout()
plt.show()
from pyts.image import GramianAngularField
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
↪ VotingClassifier, StackingClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
import pandas as pd
import os
import numpy as np
import tensorflow as tf
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import AdamW, RMSprop
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.metrics import classification_report, confusion_matrix, roc_curve
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.preprocessing import QuantileTransformer



data = pd.read_csv("modified_cmu.csv")
# Define the feature columns used for classification
feature_columns = [
    'H.period', 'DD.period.t', 'UD.period.t', 'H.t', 'DD.t.i', 'UD.t.i', 'H.i',
    ↪  'DD.i.e', 'UD.i.e',
    'H.e', 'DD.e.five', 'UD.e.five', 'H.five', 'DD.five.Shift.r', 'UD.five.Shift.r',
    ↪  'H.Shift.r',
    'DD.Shift.r.o', 'UD.Shift.r.o', 'H.o', 'DD.o.a', 'UD.o.a', 'H.a', 'DD.a.n',
    ↪  'UD.a.n', 'H.n',
    'DD.n.l', 'UD.n.l', 'H.l', 'DD.l.Return', 'UD.l.Return', 'H.Return'
]
# Define the classifiers to be evaluated
classifiers = {
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(probability=True),
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'XGBoost': XGBClassifier(eval_metric='logloss'),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Voting': VotingClassifier(estimators=[
            ('rf', RandomForestClassifier()),
            ('svc', SVC(probability=True)),
            ('xgb', XGBClassifier(eval_metric='logloss')),
            ('knn', KNeighborsClassifier())
        ], voting='soft'),
    'Stacking': StackingClassifier(estimators=[
            ('rf', RandomForestClassifier()),
            ('svc', SVC(probability=True)),
            ('xgb', XGBClassifier(eval_metric='logloss')),
            ('knn', KNeighborsClassifier())
        ], final_estimator=LogisticRegression()),
}
# Define the parameter grids for hyperparameter tuning
param_grids = {
```

```
    'KNN': {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance'], 'p': [1,
    ↪ 2]},
    'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']},
    'Random Forest': {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]},
    'Decision Tree': {'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]},
    'XGBoost': {'n_estimators': [100, 200], 'max_depth': [3, 5, 7]},
    'Gradient Boosting': {'n_estimators': [100, 200], 'learning_rate': [0.01, 0.1,
    ↪ 0.2]},
}
# Function to calculate the Equal Error Rate (EER)
def calculate_equal_error_rate(user_scores, impostor_scores):
    predictions = np.concatenate([user_scores, impostor_scores])
    labels = np.concatenate([np.zeros(len(user_scores)),
    ↪ np.ones(len(impostor_scores))])
    fpr, tpr, thresholds = roc_curve(labels, predictions)
    missrates = 1 - tpr
    farates = fpr
    dists = missrates - farates
    idx1 = np.where(dists >= 0, dists, np.inf).argmin()
    idx2 = np.where(dists < 0, dists, -np.inf).argmax()
    if abs(idx1 - idx2) != 1:
        idx2 = idx1 - 1 if idx1 > 0 else idx1 + 1
    x = np.array([missrates[idx1], farates[idx1]])
    y = np.array([missrates[idx2], farates[idx2]])
    a = (x[0] - x[1]) / (y[1] - x[1] - y[0] + x[0])
    eer = x[0] + a * (y[0] - x[0])
    return eer
# Prepare data for a specific subject
def prepare_data_for_subject(X, eval_subject):
    # Training data: genuine users and imposters
    Y_train_genuine = X[(X['subject'] == eval_subject) & (X['sessionIndex'] <= 4)]
    Y_train_impostor = X[(X['subject'] != eval_subject) & (X['sessionIndex'] == 2) &
    ↪ (X['rep'] <= 4)]
    Y_train = pd.concat([Y_train_genuine, Y_train_impostor], ignore_index=True)
    labels_train = np.concatenate([np.ones(len(Y_train_genuine)),
    ↪ np.zeros(len(Y_train_impostor))])
    # Testing data: genuine users and imposters
    Y_test_genuine = X[(X['subject'] == eval_subject) & (X['sessionIndex'] > 4)]
    Y_test_impostor = X[(X['subject'] != eval_subject) & (X['sessionIndex'] == 1) &
    ↪ (X['rep'] <= 4)]
    Y_test = pd.concat([Y_test_genuine, Y_test_impostor], ignore_index=True)
    labels_test = np.concatenate([np.ones(len(Y_test_genuine)),
    ↪ np.zeros(len(Y_test_impostor))])
```

```
    # Feature columns
    feature_columns = Y_train.columns.difference(['subject', 'sessionIndex', 'rep',
    ↪  'subject_num', 'total time'])
    X_train = Y_train[feature_columns].values
    X_test = Y_test[feature_columns].values
    return X_train, labels_train, X_test, labels_test
# Evaluate a classifier using cross-validation
def evaluate_classifier_cv(X_train, y_train, X_test, y_test, classifier,
↪  param_grid):
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(classifier, param_grid, cv=skf, scoring='accuracy',
    ↪  n_jobs=-1)
    grid_search.fit(X_train, y_train)
    best_classifier = grid_search.best_estimator_
    predictions = best_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    # Calculate EER if classifier supports probability prediction
    if hasattr(best_classifier, "predict_proba"):
        scores = best_classifier.predict_proba(X_test)[:, 1]
        user_scores = scores[y_test == 0]
        impostor_scores = scores[y_test == 1]
        eer = calculate_equal_error_rate(user_scores, impostor_scores)
    else:
        eer = None
    # Calculate FAR and FRR
    tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
    far = fp / (fp + tn) if (fp + tn) > 0 else 0
    frr = fn / (fn + tp) if (fn + tp) > 0 else 0
    return accuracy, eer, far, frr
# Evaluate all classifiers for all subjects
def evaluate_all_subjects(data):
    subjects = data['subject'].unique()
    results = []
    for subject in subjects:
        X_train, y_train, X_test, y_test = prepare_data_for_subject(data, subject)
        quantile_transformer = QuantileTransformer(output_distribution='uniform')
        X_train = quantile_transformer.fit_transform(X_train)
        X_test = quantile_transformer.transform(X_test)
        for classifier_name, classifier in classifiers.items():
            param_grid = param_grids.get(classifier_name, {})
            accuracy, eer, far, frr = evaluate_classifier_cv(X_train, y_train,
            ↪  X_test, y_test, classifier, param_grid)
            results.append({
```

```
                'Subject': subject,
                'Classifier': classifier_name,
                'Accuracy': accuracy,
                'EER': eer,
                'FAR': far,
                'FRR': frr
            })
    return pd.DataFrame(results)

# Run evaluation across all subjects
results_df = evaluate_all_subjects(data)
# Calculate and display average results for all classifiers
average_results = results_df.groupby(['Classifier']).agg({'Accuracy': 'mean', 'EER':
↪  'mean', 'FAR': 'mean', 'FRR': 'mean'}).round(4)
print(average_results)

##############################GASF and GADF###################################
# Initialize MinMaxScaler for normalization to [-1, 1]
scaler = MinMaxScaler(feature_range=(-1, 1))
# Initialize GASF and GADF transformations
gasf = GramianAngularField(method='summation')
gadf = GramianAngularField(method='difference')
# Define selected features and output directory
selected_features = [
    'H.period', 'DD.period.t', 'UD.period.t', 'H.t', 'DD.t.i', 'UD.t.i', 'H.i',
    ↪  'DD.i.e', 'UD.i.e',
    'H.e', 'DD.e.five', 'UD.e.five', 'H.five', 'DD.five.Shift.r', 'UD.five.Shift.r',
    ↪  'H.Shift.r',
    'DD.Shift.r.o', 'UD.Shift.r.o', 'H.o', 'DD.o.a', 'UD.o.a', 'H.a', 'DD.a.n',
    ↪  'UD.a.n', 'H.n',
    'DD.n.l', 'UD.n.l', 'H.l', 'DD.l.Return', 'UD.l.Return', 'H.Return'
]
output_dir = "gasf_gadf_images_cmu/"
os.makedirs(output_dir, exist_ok=True)
# Function to generate a single GASF and GADF image per subject
def generate_gasf_gadf_images(data, output_dir, split_ratio=0.5):
    subjects = data['subject'].unique()
    for subject in subjects:
        subject_data = data[data['subject'] == subject]
        for (session, repetition), group in subject_data.groupby(['session',
        ↪  'repetition']):
                # Concatenate selected features into a single time series
                time_series = group[selected_features].values.flatten()
```

```
                # Normalize the concatenated time series to [-1, 1]
                time_series_scaled = scaler.fit_transform(time_series.reshape(-1,
                ↪  1)).reshape(1, -1)
                # Generate GASF and GADF images
                gaf_image = gasf.fit_transform(time_series_scaled)[0]
                gadf_image = gadf.fit_transform(time_series_scaled)[0]
                # Normalize images to [0, 1] for saving
                gaf_image = (gaf_image - gaf_image.min()) / (gaf_image.max() -
                ↪  gaf_image.min())
                gadf_image = (gadf_image - gadf_image.min()) / (gadf_image.max() -
                ↪  gadf_image.min())
                # Save the images in the specified output directory
                gaf_filename =
                ↪  f"{subject}_session{session}_rep{repetition}_gasf.png"
                gadf_filename =
                ↪  f"{subject}_session{session}_rep{repetition}_gadf.png"
                plt.imsave(os.path.join(output_dir, gaf_filename), gaf_image,
                ↪  cmap='rainbow')
                plt.imsave(os.path.join(output_dir, gadf_filename), gadf_image,
                ↪  cmap='rainbow')
# Generate a single GASF and GADF image per subject
generate_gasf_gadf_images(data, output_dir)
##############################CNN#############################################
# Define the main tuning directory
tuning_directory = 'cmu'
# Define image size and directories
output_dir = "gasf_gadf_images_cmu/"
image_size = (64, 64)
# Define a helper function to add Gaussian noise
def add_gaussian_noise(image, mean=0.0, stddev=0.05):
    noise = np.random.normal(mean, stddev, image.shape)
    return np.clip(image + noise, 0.0, 1.0)
# Helper function to load images for specific sessions and repetitions
def load_images(subject_code, sessions, reps, label):
    images = []
    labels = []
    for session in sessions:
        for rep in reps:
            for image_type in ['gasf', 'gadf']:
                filename = f"{subject_code}_session{session}_rep{rep +
                ↪  1}_{image_type}.png"
                path = os.path.join(output_dir, filename)
                if os.path.exists(path):
```

```
                    img = load_img(path, target_size=image_size, color_mode="rgb")
                    img_array = img_to_array(img) / 255.0
                    img_array = add_gaussian_noise(img_array)
                    images.append(img_array)
                    labels.append(label)
    return np.array(images), np.array(labels)
# Get unique subject codes from filenames
subject_codes = set(filename.split("_")[0] for filename in os.listdir(output_dir) if
↪  filename.startswith("s"))
# Function to build the CNN model
def build_model(hp):
    model = Sequential()
    model.add(Conv2D(
        filters=hp.Choice('filters_1', values=[16, 32]),
        kernel_size=hp.Choice('kernel_size_1', values=[3]),
        activation='relu', input_shape=(image_size[0], image_size[1], 3)))
    model.add(MaxPooling2D())
    model.add(Conv2D(
        filters=hp.Choice('filters_2', values=[64, 128]),
        kernel_size=hp.Choice('kernel_size_2', values=[3]),
        activation='relu'))
    model.add(MaxPooling2D())
    model.add(Conv2D(
        filters=hp.Choice('filters_3', values=[128, 256]),
        kernel_size=hp.Choice('kernel_size_3', values=[3, 5]),
        activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(units=hp.Choice('units_dense', values=[64, 128, 256]),
    ↪  activation='relu'))
    model.add(Dropout(rate=hp.Choice('dropout_dense', values=[0.2, 0.4, 0.5])))
    model.add(Dense(1, activation='sigmoid'))
    optimizer_choice = hp.Choice('optimizer', ['AdamW', 'RMSprop'])
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    if optimizer_choice == 'AdamW':
        optimizer = AdamW(learning_rate=learning_rate)
    else:
        optimizer = RMSprop(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
    ↪  metrics=['accuracy'])
    return model
# Loop through each subject as the genuine user
accuracies = []
```

```
eers = []
fars = []
frrs = []
for genuine_user_code in subject_codes:
    print(f"Evaluating for subject {genuine_user_code} as the genuine user...")
    # Load data for genuine user and imposters
    train_images_genuine, train_labels_genuine = load_images(genuine_user_code, [1],
    ↪  range(100), 1)
    test_images_genuine, test_labels_genuine = load_images(genuine_user_code, [2],
    ↪  range(100), 1)
    train_images_imposter = []
    train_labels_imposter = []
    test_images_imposter = []
    test_labels_imposter = []
    for subject_code in subject_codes:
        if subject_code != genuine_user_code:
            imp_train_images, imp_train_labels = load_images(subject_code, [2],
            ↪  range(1), 0)
            imp_test_images, imp_test_labels = load_images(subject_code, [1],
            ↪  range(1), 0)
            train_images_imposter.extend(imp_train_images)
            train_labels_imposter.extend(imp_train_labels)
            test_images_imposter.extend(imp_test_images)
            test_labels_imposter.extend(imp_test_labels)
    train_images_imposter = np.array(train_images_imposter)
    train_labels_imposter = np.array(train_labels_imposter)
    test_images_imposter = np.array(test_images_imposter)
    test_labels_imposter = np.array(test_labels_imposter)
    # Combine genuine and imposter data
    X_train = np.concatenate((train_images_genuine, train_images_imposter), axis=0)
    y_train = np.concatenate((train_labels_genuine, train_labels_imposter), axis=0)
    X_test = np.concatenate((test_images_genuine, test_images_imposter), axis=0)
    y_test = np.concatenate((test_labels_genuine, test_labels_imposter), axis=0)
    # Set the random seed for reproducibility
    seed = 1
    np.random.seed(seed)
    tf.random.set_seed(seed)
    # Define unique tuning directory for each subject
    subject_tuning_dir = Path(tuning_directory) / f'tuning_{genuine_user_code}'
    # Check if the subject's tuner data exists
    if subject_tuning_dir.exists() and any(subject_tuning_dir.iterdir()):
        print(f"Tuner data for {genuine_user_code} found. Restoring previous
        ↪  tuner.")
```

```
        overwrite_tuner = False
    else:
        print(f"No tuner data for {genuine_user_code} or data is corrupt. Starting
        ↪ fresh tuning.")
        overwrite_tuner = True
    # Initialize the tuner for the current subject
    tuner = kt.RandomSearch(
        build_model,
        objective='val_accuracy',
        directory=tuning_directory,
        project_name=f'tuning_{genuine_user_code}',
        seed=seed,
        overwrite=overwrite_tuner  # Decide whether to overwrite or restore previous
        ↪ tuning
    )
    if overwrite_tuner:
        tuner.search(X_train, y_train, epochs=20, validation_data=(X_test, y_test),
        ↪ verbose=0)
    # Retrieve the best model
    best_model = tuner.get_best_models(num_models=1)[0]
    # Evaluate on the test set and calculate metrics
    y_pred = (best_model.predict(X_test) > 0.5).astype("int32")
    acc = np.mean(y_pred.flatten() == y_test)
    accuracies.append(acc)
    # Get confusion matrix for FAR and FRR calculation
    conf_matrix = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = conf_matrix.ravel()
    # Calculate FAR and FRR
    far = fp / (fp + tn) if (fp + tn) > 0 else 0  # FAR: False positives / All
    ↪ negatives
    frr = fn / (fn + tp) if (fn + tp) > 0 else 0  # FRR: False negatives / All
    ↪ positives
    fars.append(far)
    frrs.append(frr)
    # Get scores for EER calculation
    scores = best_model.predict(X_test).flatten()
    user_scores = scores[y_test == 0]
    impostor_scores = scores[y_test == 1]
    eer = calculate_equal_error_rate(user_scores, impostor_scores)
    eers.append(eer)
    print(f"Classification Report for {genuine_user_code}:\n")
    print(classification_report(y_test, y_pred, target_names=['Imposter',
    ↪ 'Genuine']))
```

```
    print(f"Confusion Matrix for {genuine_user_code}:\n{conf_matrix}\n")
    print(f"False Acceptance Rate (FAR): {far:.4f}")
    print(f"False Rejection Rate (FRR): {frr:.4f}")
    print(f"Equal Error Rate (EER) for {genuine_user_code}: {eer:.4f}\n")
# Calculate and print the average metrics across all subjects
average_accuracy = np.mean(accuracies)
average_eer = np.mean(eers)
average_far = np.mean(fars)
average_frr = np.mean(frrs)
print(f"Average Test Accuracy across all subjects: {average_accuracy * 100:.2f}%")
print(f"Average Equal Error Rate (EER) across all subjects: {average_eer:.4f}")
print(f"Average False Acceptance Rate (FAR) across all subjects: {average_far:.4f}")
print(f"Average False Rejection Rate (FRR) across all subjects: {average_frr:.4f}")

from sklearn.manifold import MDS, TSNE
from sklearn.preprocessing import StandardScaler
from umap import UMAP
from sklearn.decomposition import KernelPCA
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt
import seaborn as sns
##############################################
##########Data cleaning#######################
##############################################
df = pd.read_csv('fixed-text1.csv')
df.rename(columns={'participant':'subject'}, inplace=True)
# Create a numeric variable
df['subject_num'] = df['subject'].str.extract('(\d+)').astype(int)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
# Rename columns from DU to H
def modify_DU_columns(df):
    # Starts with DU (group 1), followed by . (group 2), followed by one symbol
    ↪  (group 3),
    # followed by . followed by the same symbol as in group 3,
    # then we replace DU by H followed by everything after group 2
    df.columns = df.columns.str.replace(r'^(DU)(\.(.)(\.\3)?)$', r'H\2', regex=True)
    # Starts with DU (group 1), followed by . (group 2), followed by one symbol
    ↪  (group 3),
    # followed by . followed by the same symbol as in group 3, followed by any
    ↪  symbol
```

```
    # then we replace DU by H followed by everything after group 2
    df.columns = df.columns.str.replace(r'^(DU)(\.(.)(\.\3).(.))', r'H\2',
    ↪  regex=True)
    return df
df = modify_DU_columns(df)
# List of column names starting with DD
dd_columns = [col for col in df.columns if col.startswith('DD')]
# List of column names starting with UU
uu_columns = [col for col in df.columns if col.startswith('UU')]
# Swapping values between columns starting with DD and UU
for dd_col, uu_col in zip(dd_columns, uu_columns):
    df[dd_col], df[uu_col] = df[uu_col].copy(), df[dd_col].copy()
# List of column names starting with DU
du_columns = [col for col in df.columns if col.startswith('DU')]
# List of column names starting with UD
ud_columns = [col for col in df.columns if col.startswith('UD')]
# Swapping values between columns starting with DU and UD
for du_col, ud_col in zip(du_columns, ud_columns):
    df[du_col], df[ud_col] = df[ud_col].copy(), df[du_col].copy()

password = "vpwjkeurkb"

# Creating lists of column names
listofUpDown = [col for col in df.columns if col.startswith('UD')]
listofUp = [col for col in df.columns if col.startswith('UU')]
listofDown = [col for col in df.columns if col.startswith('DD')]
listofHold = [col for col in df.columns if col.startswith('H')]
listofDownUp = [col for col in df.columns if col.startswith('DU')]
listofTrigraph = ['tri.v.p.w', 'tri.p.w.j', 'tri.w.j.k', 'tri.j.k.e', 'tri.k.e.u',
↪  'tri.e.u.r', 'tri.u.r.k', 'tri.r.k.b']
# Creating columns with trigraph values
for j in range(len(listofDown) - 1):
    df[listofTrigraph[j]] = df[listofDown[j]] + df[listofDown[j + 1]]

##################################################
############Explanatory data analysis############
##################################################
# Descriptive statistics of all the columns capturing the keystroke timing
descriptive_stats = df[df.columns.difference(['subject', 'session', 'repetition',
↪  'subject_num'])].describe().T
descriptive_stats = descriptive_stats[['mean', 'std', 'min', '50%', 'max']]
# Delete all rows that have values higher then 20 in columns H and UD
numeric_cols = df.filter(regex=r'^(H|UD)').select_dtypes(include=np.number)
```

```
# deleted_rows = df[~(numeric_cols <= 20).all(axis=1)]
df = df[(numeric_cols <= 20).all(axis=1)]
# Delete all rows that have negative values in columns H and DD
numeric_cols1 = df.filter(regex=r'^(H|DD)').select_dtypes(include=np.number)
# deleted_rows1 = df[~(numeric_cols >= 0).all(axis=1)]
df = df[(numeric_cols1 >= 0).all(axis=1)]
df = df[df["subject"] != "p004"]
# Descriptive statistics after cleaning data from illogical values
descriptive_stats = df[df.columns.difference(['subject', 'session', 'repetition',
↪  'subject_num'])].describe().T
descriptive_stats = descriptive_stats[['mean', 'std', 'min', '50%', 'max']]
print(descriptive_stats)
df.to_csv(r'C:\Users\eveli\PycharmProjects\Magistrinis\modified_fixed-text.csv',
↪  index=False)
#########################################################
######################Plots##############################
#########################################################
# Prepare lists of features
listofUpDown = [col for col in df.columns if col.startswith('UD')]
listofHold = [col for col in df.columns if col.startswith('H')]
listofDown = [col for col in df.columns if col.startswith('DD')]
# Features to use for similarity and distance computations
features = df.columns.difference(['subject', 'sessionIndex', 'rep', 'subject_num'])
# Get unique subjects
subjects = df['subject'].unique()
results = []
# Compute similarity and distance metrics between subjects
for i in range(len(subjects)):
    for j in range(i + 1, len(subjects)):
        subj1 = subjects[i]
        subj2 = subjects[j]
        # Extract subject data
        subject1 = df[df['subject'] == subj1]
        subject2 = df[df['subject'] == subj2]
        # Calculate feature vector means
        avg_vector1 = subject1[features].mean().values
        avg_vector2 = subject2[features].mean().values
        # Compute similarity and distance metrics
        cos_sim = cosine_similarity(avg_vector1.reshape(1, -1),
        ↪  avg_vector2.reshape(1, -1))[0][0]
        euclidean_dist = euclidean(avg_vector1, avg_vector2)
        results.append({
            'Subject1': subj1,
```

```
            'Subject2': subj2,
            'Cosine Similarity': cos_sim,
            'Euclidean Distance': euclidean_dist
        })
# Create results DataFrame
results_df = pd.DataFrame(results)
# Function to find maximum and minimum values for a metric
def similar(name, results_df):
    max_val = results_df[name].max()
    min_val = results_df[name].min()
    print(f"\nMax value: \n{results_df[results_df[name] == max_val]}:")
    print(f"\nMin value: \n{results_df[results_df[name] == min_val]}:")
# Print max/min results for each metric
for metric in ['Cosine Similarity', 'Euclidean Distance']:
    print(f"\nResults for {metric}:")
    similar(metric, results_df)
# Function to plot similarity and distance heatmaps
def plot_similarities(results_df):
    # Increase figure size for better readability
    fig, axes = plt.subplots(1, 2, figsize=(25, 20))
    # Cosine Similarity Heatmap
    pivot_cosine = results_df.pivot(index='Subject1', columns='Subject2',
    ↪  values='Cosine Similarity')
    sns.heatmap(
        pivot_cosine, annot=False, cmap='coolwarm',
        cbar_kws={'label': 'Cosine Similarity'}, ax=axes[0]
    )
    axes[0].set_title('Cosine Similarity Between Subjects', fontsize=16)
    axes[0].tick_params(axis='x', labelrotation=90, labelsize=12)
    axes[0].tick_params(axis='y', labelrotation=0, labelsize=12)
    axes[0].set_xlabel('Subject2', fontsize=14)
    axes[0].set_ylabel('Subject1', fontsize=14)
    # Euclidean Distance Heatmap
    pivot_euclidean = results_df.pivot(index='Subject1', columns='Subject2',
    ↪  values='Euclidean Distance')
    sns.heatmap(
        pivot_euclidean, annot=False, cmap='coolwarm',
        cbar_kws={'label': 'Euclidean Distance'}, ax=axes[1]
    )
    axes[1].set_title('Euclidean Distance Between Subjects', fontsize=16)
    axes[1].tick_params(axis='x', labelrotation=90, labelsize=12)
    axes[1].tick_params(axis='y', labelrotation=0, labelsize=12)
    axes[1].set_xlabel('Subject2', fontsize=14)
```

```
    axes[1].set_ylabel('Subject1', fontsize=14)
    # Add legends for both subplots
    cbar_cosine = axes[0].collections[0].colorbar
    cbar_cosine.ax.tick_params(labelsize=12)
    cbar_cosine.set_label('Cosine Similarity Scale', fontsize=14)
    cbar_euclidean = axes[1].collections[0].colorbar
    cbar_euclidean.ax.tick_params(labelsize=12)
    cbar_euclidean.set_label('Euclidean Distance Scale', fontsize=14)
    plt.tight_layout()
    plt.show()
# Plot similarity and distance graphs for a subset of subjects
unique_subjects = results_df['Subject1'].unique()
# Dynamically subset subjects
subjects_subset = unique_subjects[0:31]
if len(unique_subjects) > 46:
    subjects_subset = np.append(subjects_subset, unique_subjects[44:49])
# Filter results for both Subject1 and Subject2
subset_df = results_df[
    results_df['Subject1'].isin(subjects_subset) &
    ↪ results_df['Subject2'].isin(subjects_subset)
]
# Plot the similarities
plot_similarities(subset_df)
######################Timing##################
def plot_subject_data(subject_data, subject_of_interest, session_number,
↪ repetition_number, normalized=False):
    # Select columns starting with H or UD
    filtered_columns = subject_data.filter(regex='^(H|UD)').columns
    # Check if non-empty dataframe with info about subject, session and repetition
    if not subject_data.empty:
        # Flaten the timing values into a 1D array for cumulative sum
        timing_values = subject_data[filtered_columns].values.flatten()
        # Calculate cumulative sum
        x_values = np.cumsum(timing_values)
        # Normalization for graph with normalized data (in 0 to 1 range)
        if normalized:
            x_values = (x_values - x_values[0]) / (x_values[-1] - x_values[0])
        # Label containing subject, session, repetition
        y_value = f'{subject_of_interest}, Session: {session_number}, Repetition:
        ↪ {repetition_number}'
        return x_values, [y_value] * len(x_values)
    return None, None
def plot_keystroke_timing(df, subject_combinations, normalized=False):
```

```
    plt.figure(figsize=(12, 8))
    # Loop over each subject combination
    for subject_of_interest, session_number, repetition_number in
    ↪ subject_combinations:
        # Filter the data for the specific subject, session, and repetition
        subject_data = df[(df['subject'] == subject_of_interest) &
                          (df['session'] == session_number) &
                          (df['repetition'] == repetition_number)]
        # Get the timing values for plotting
        x_values, y_values = plot_subject_data(subject_data, subject_of_interest,
        ↪ session_number, repetition_number, normalized=normalized)
        if x_values is not None:
            plt.plot(x_values, y_values, marker='o', linestyle='-', linewidth=4)
            plt.scatter(x_values, y_values, s=100)
    normalization_text = "Normalized" if normalized else "Non-Normalized"
    plt.title(f'{normalization_text} Keystroke Timing for Selected Subjects',
    ↪ fontsize=20)
    plt.xlabel('Timing (Fraction of Typing Time)' if normalized else 'Timing
    ↪ (Cumulative Time)', fontsize=18)
    plt.ylabel('Subject, Session, and Repetition', fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
# Define all combinations of subject, session, and repetition
subject_combinations = [
    ('p018', 2, 75), ('p041', 1, 69),
    ('p018', 2, 79), ('p092', 1, 9),
]
# Plot normalized data
plot_keystroke_timing(df, subject_combinations, normalized=True)
# Plot non-normalized data
plot_keystroke_timing(df, subject_combinations, normalized=False)
########################Linear#########################
subject_pairs = [
    ['p052', 'p080'],
    ['p018', 'p092'],
    ['p003', 'p041']
]
# Extracting data for specific subjects
all_subjects = [subject for pair in subject_pairs for subject in pair]
df_filtered = df[df['subject'].isin(all_subjects)]
```

```
# Compute mean of each feature per subject
mean_features_per_subject = df_filtered.groupby('subject').mean()
def plot_side_by_side_hold_updown(df, subject_pairs, hold_features,
↪  updown_features):
    nrows = 3
    ncols = 2
    fig, axes = plt.subplots(nrows, ncols, figsize=(12, 18))
    axes = axes.flatten()
    # Iterate over each pair of subjects
    for i, pair in enumerate(subject_pairs):
        hold_ax = axes[i * 2]
        updown_ax = axes[i * 2 + 1]
        # Plot Hold features for each subject in the pair
        for subject in pair:
            hold_values = df.loc[subject, hold_features]
            hold_ax.plot(
                hold_values.index, hold_values.values,
                marker='o', linestyle='-', label=f'{subject}'
            )
        # Customize the Hold features plot
        hold_ax.set_xticks(range(len(hold_features)))
        hold_ax.set_xticklabels(hold_features, rotation=45, ha='right', fontsize=12)
        hold_ax.set_title(f'Hold Features: {pair[0]} vs {pair[1]}', fontsize=14)
        hold_ax.set_ylabel('Value', fontsize=12)
        hold_ax.legend(title='Subject', fontsize=12, loc='best')
        # Plot Up-Down features for each subject in the pair
        for subject in pair:
            updown_values = df.loc[subject, updown_features]
            updown_ax.plot(
                updown_values.index, updown_values.values,
                marker='o', linestyle='-', label=f'{subject}'
            )
        updown_ax.set_xticks(range(len(updown_features)))
        updown_ax.set_xticklabels(updown_features, rotation=45, ha='right',
        ↪  fontsize=12)
        updown_ax.set_title(f'Up-Down Features: {pair[0]} vs {pair[1]}',
        ↪  fontsize=14)
        updown_ax.set_ylabel('Value', fontsize=12)
        updown_ax.legend(title='Subject', fontsize=12, loc='upper left')
    plt.tight_layout()
    plt.show()
# Plot Hold and Up-Down features for the defined subject pairs
```

```
plot_side_by_side_hold_updown(mean_features_per_subject, subject_pairs, listofHold,
↪ listofUpDown)
##################################################Dimension
↪ reduction############################################################
user_pairs = [[52, 80], [18, 92], [3, 41]]
fig, axes = plt.subplots(len(user_pairs), 4, figsize=(24, 10))
# Iterate through each row (user pair) in the grid
for row, pair in enumerate(user_pairs):
    # Filter the DataFrame to include only the two subjects in the current pair
    df_two_users = df[df['subject_num'].isin(pair)]
    # Extract feature matrix (X) and target labels (y)
    # Drop non-feature columns like 'subject', 'rep', 'subject_num', and
    ↪ 'sessionIndex'
    X = df_two_users.drop(columns=['subject', 'rep', 'subject_num', 'sessionIndex'],
    ↪ errors='ignore').values
    y = df_two_users['subject_num'].values
    # Standardize the feature matrix (mean=0, variance=1)
    X = StandardScaler().fit_transform(X)
    # Define dimensionality reduction methods with their configurations
    projection_methods = {
        'Kernel PCA': KernelPCA(n_components=2, kernel='rbf'),
        'Umap': UMAP(n_components=2, n_neighbors=10, min_dist=0.01),
        'Metric MDS': MDS(n_components=2, metric=True),
        't-SNE': TSNE(n_components=2, perplexity=3, learning_rate=200, n_iter=3500),
    }
    # Map subject numbers to colors for the scatter plot
    color_map = {pair[0]: 'orange', pair[1]: 'blue'}
    colors = np.array([color_map[num] for num in y])
    for col, (name, method) in enumerate(projection_methods.items()):
        # Apply the dimensionality reduction method
        transformed = method.fit_transform(X)
        # Plot the transformed data in the corresponding subplot
        axes[row, col].scatter(
            transformed[:, 0], transformed[:, 1], c=colors, s=25, alpha=0.6
        )
        axes[row, col].set_title(f"{name} for subjects {pair[0]} and {pair[1]}",
        ↪ fontsize=14)
        axes[row, col].set_xticks([]), axes[row, col].set_yticks([])
plt.tight_layout()
plt.show()
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
↪  VotingClassifier, StackingClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
import pandas as pd
import os
import numpy as np
import tensorflow as tf
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import AdamW, RMSprop
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.metrics import classification_report, confusion_matrix, roc_curve
from pathlib import Path

# Load data
data = pd.read_csv("modified_fixed-text.csv")
print(data.columns)
# Define feature columns
feature_columns = [
       'H.v.v', 'DD.v.p','UD.v.p', 'H.p.p', 'DD.p.w', 'UD.p.w','H.w.w', 'DD.w.j',
       ↪  'UD.w.j', 'H.j.j', 'DD.j.k',
       'UD.j.k', 'H.k.k', 'DD.k.e', 'UD.k.e', 'H.e.e', 'DD.e.u', 'UD.e.u', 'H.u.u',
       ↪  'DD.u.r', 'UD.u.r', 'H.r.r',
       'DD.r.k', 'UD.r.k', 'H.k.k.1', 'DD.k.b', 'UD.k.b', 'H.b.b'
]
param_grids = {
    'KNN': {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance'], 'p': [1,
    ↪  2, 3]},
    'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'degree':[2,3,4]},
    'Random Forest': {'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20]},
    'Decision Tree': {'criterion': ['gini', 'entropy', 'log_loss'], 'max_depth':
    ↪  [None, 10, 20], 'min_samples_split': [2, 5, 10]},
    'XGBoost': {'n_estimators': [100, 200, 300], 'max_depth': [4, 6, 8],
    ↪  'learning_rate': [0.2, 0.3, 0.4]},
    'Gradient Boosting': {'n_estimators': [100, 200, 300], 'learning_rate': [0.01,
    ↪  0.1, 0.2]},
}
# Base classifiers
```

```
classifiers = {
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(probability=True),
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'XGBoost': XGBClassifier(eval_metric='logloss'),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Voting': VotingClassifier(estimators=[
        ('rf', RandomForestClassifier()),
        ('svc', SVC(probability=True)),
        ('xgb', XGBClassifier(eval_metric='logloss')),
        ('knn', KNeighborsClassifier())
    ], voting='soft'),
    'Stacking': StackingClassifier(estimators=[
        ('rf', RandomForestClassifier()),
        ('svc', SVC(probability=True)),
        ('xgb', XGBClassifier(eval_metric='logloss')),
        ('knn', KNeighborsClassifier())
    ], final_estimator=LogisticRegression()),
}
# EER calculation function
def calculate_equal_error_rate(user_scores, impostor_scores):
    predictions = np.concatenate([user_scores, impostor_scores])
    labels = np.concatenate([np.zeros(len(user_scores)),
    ↪  np.ones(len(impostor_scores))])
    fpr, tpr, thresholds = roc_curve(labels, predictions)
    missrates = 1 - tpr
    farates = fpr
    dists = missrates - farates
    idx1 = np.where(dists >= 0, dists, np.inf).argmin()
    idx2 = np.where(dists < 0, dists, -np.inf).argmax()
    if abs(idx1 - idx2) != 1:
        idx2 = idx1 - 1 if idx1 > 0 else idx1 + 1
    x = np.array([missrates[idx1], farates[idx1]])
    y = np.array([missrates[idx2], farates[idx2]])
    a = (x[0] - x[1]) / (y[1] - x[1] - y[0] + x[0])
    eer = x[0] + a * (y[0] - x[0])
    return eer
# Data preparation for each subject
def prepare_data_for_subject(X, eval_subject):
    Y_train_genuine = X[(X['subject'] == eval_subject) & (X['session'] <= 1)]
    Y_train_impostor = X[(X['subject'] != eval_subject) & (X['session'] == 2) &
    ↪  (X['repetition'] <= 1)]
```

```
    Y_train = pd.concat([Y_train_genuine, Y_train_impostor], ignore_index=True)
    labels_train = np.concatenate([np.ones(len(Y_train_genuine)),
    ↪  np.zeros(len(Y_train_impostor))])

    Y_test_genuine = X[(X['subject'] == eval_subject) & (X['session'] > 1)]
    Y_test_impostor = X[(X['subject'] != eval_subject) & (X['session'] == 1) &
    ↪  (X['repetition'] <= 1)]
    Y_test = pd.concat([Y_test_genuine, Y_test_impostor], ignore_index=True)
    labels_test = np.concatenate([np.ones(len(Y_test_genuine)),
    ↪  np.zeros(len(Y_test_impostor))])

    feature_columns = Y_train.columns.difference(['subject', 'session',
    ↪  'repetition', 'subject_num', 'total time'])
    X_train = Y_train[feature_columns].values
    X_test = Y_test[feature_columns].values
    return X_train, labels_train, X_test, labels_test
# Evaluate classifiers
def evaluate_classifier_cv(X_train, y_train, X_test, y_test, classifier,
↪  param_grid):
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(classifier, param_grid, cv=skf, scoring='accuracy',
    ↪  n_jobs=-1)
    grid_search.fit(X_train, y_train)
    best_classifier = grid_search.best_estimator_
    predictions = best_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    if hasattr(best_classifier, "predict_proba"):
        scores = best_classifier.predict_proba(X_test)[:, 1]
        user_scores = scores[y_test == 0]
        impostor_scores = scores[y_test == 1]
        eer = calculate_equal_error_rate(user_scores, impostor_scores)
    else:
        eer = None
    # Calculate FAR and FRR
    tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
    far = fp / (fp + tn) if (fp + tn) > 0 else 0
    frr = fn / (fn + tp) if (fn + tp) > 0 else 0
    return accuracy, eer, far, frr
# Evaluate all subjects
def evaluate_all_subjects(data):
    subjects = data['subject'].unique()
    results = []
    for subject in subjects:
```

```
        X_train, y_train, X_test, y_test = prepare_data_for_subject(data, subject)
        for classifier_name, classifier in classifiers.items():
            param_grid = param_grids.get(classifier_name, {})
            accuracy, eer, far, frr = evaluate_classifier_cv(X_train, y_train,
            ↪  X_test, y_test, classifier, param_grid)
            results.append({
                'Subject': subject,
                'Classifier': classifier_name,
                'Accuracy': accuracy,
                'EER': eer,
                'FAR': far,
                'FRR': frr
            })
    return pd.DataFrame(results)
# Run evaluation
results_df = evaluate_all_subjects(data)
average_results = results_df.groupby(['Classifier']).agg({'Accuracy': 'mean', 'EER':
↪  'mean', 'FAR': 'mean', 'FRR': 'mean'}).round(4)
print(average_results)
##############################GASF and GADF###################################
# Initialize MinMaxScaler for normalization to [-1, 1]
scaler = MinMaxScaler(feature_range=(-1, 1))
# Initialize GASF and GADF transformations
gasf = GramianAngularField(method='summation')
gadf = GramianAngularField(method='difference')
# Define selected features and output directory
selected_features = [
        'H.v.v', 'DD.v.p', 'UD.v.p', 'H.p.p', 'DD.p.w', 'UD.p.w',
        'H.w.w', 'DD.w.j', 'UD.w.j', 'H.j.j', 'DD.j.k', 'UD.j.k',
        'H.k.k', 'DD.k.e', 'UD.k.e', 'H.e.e', 'DD.e.u', 'UD.e.u',
        'H.u.u', 'DD.u.r', 'UD.u.r', 'H.r.r', 'DD.r.k', 'UD.r.k',
        'H.k.k.1', 'DD.k.b', 'UD.k.b', 'H.b.b'
]
output_dir = "gasf_gadf_images_final/"
os.makedirs(output_dir, exist_ok=True)
# Function to generate a single GASF and GADF image per subject
def generate_gasf_gadf_images(data, output_dir, split_ratio=0.5):
    subjects = data['subject'].unique()
    for subject in subjects:
        subject_data = data[data['subject'] == subject]
        for (session, repetition), group in subject_data.groupby(['session',
        ↪  'repetition']):
                # Concatenate selected features into a single time series
```

```
                time_series = group[selected_features].values.flatten()
                # Normalize the concatenated time series to [-1, 1]
                time_series_scaled = scaler.fit_transform(time_series.reshape(-1,
                ↪  1)).reshape(1, -1)
                # Generate GASF and GADF images
                gaf_image = gasf.fit_transform(time_series_scaled)[0]
                gadf_image = gadf.fit_transform(time_series_scaled)[0]
                # Normalize images to [0, 1] for saving
                gaf_image = (gaf_image - gaf_image.min()) / (gaf_image.max() -
                ↪  gaf_image.min())
                gadf_image = (gadf_image - gadf_image.min()) / (gadf_image.max() -
                ↪  gadf_image.min())
                # Save the images in the specified output directory
                gaf_filename =
                ↪  f"{subject}_session{session}_rep{repetition}_gasf.png"
                gadf_filename =
                ↪  f"{subject}_session{session}_rep{repetition}_gadf.png"
                plt.imsave(os.path.join(output_dir, gaf_filename), gaf_image,
                ↪  cmap='rainbow')
                plt.imsave(os.path.join(output_dir, gadf_filename), gadf_image,
                ↪  cmap='rainbow')
# Generate a single GASF and GADF image per subject
generate_gasf_gadf_images(data, output_dir)
##############################CNN#############################################
# Define the main tuning directory
tuning_directory = 'fixed'
# Define image size and directories
output_dir = "gasf_gadf_images_final/"
image_size = (64, 64)
# Define a helper function to add Gaussian noise
def add_gaussian_noise(image, mean=0.0, stddev=0.05):
    noise = np.random.normal(mean, stddev, image.shape)
    return np.clip(image + noise, 0.0, 1.0)
# Helper function to load images for specific sessions and repetitions
def load_images(subject_code, sessions, reps, label):
    images = []
    labels = []
    for session in sessions:
        for rep in reps:
            for image_type in ['gasf', 'gadf']:
                filename = f"{subject_code}_session{session}_rep{rep +
                ↪  1}_{image_type}.png"
                path = os.path.join(output_dir, filename)
```

```
                if os.path.exists(path):
                    img = load_img(path, target_size=image_size, color_mode="rgb")
                    img_array = img_to_array(img) / 255.0
                    img_array = add_gaussian_noise(img_array)
                    images.append(img_array)
                    labels.append(label)
    return np.array(images), np.array(labels)
# Get unique subject codes from filenames
subject_codes = set(filename.split("_")[0] for filename in os.listdir(output_dir) if
↪  filename.startswith("p"))
# Function to build the CNN model
def build_model(hp):
    model = Sequential()
    model.add(Conv2D(
        filters=hp.Choice('filters_1', values=[16, 32]),
        kernel_size=hp.Choice('kernel_size_1', values=[3]),
        activation='relu', input_shape=(image_size[0], image_size[1], 3)))
    model.add(MaxPooling2D())
    model.add(Conv2D(
        filters=hp.Choice('filters_2', values=[64, 128]),
        kernel_size=hp.Choice('kernel_size_2', values=[3]),
        activation='relu'))
    model.add(MaxPooling2D())
    model.add(Conv2D(
        filters=hp.Choice('filters_3', values=[128, 256]),
        kernel_size=hp.Choice('kernel_size_3', values=[3, 5]),
        activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(units=hp.Choice('units_dense', values=[64, 128, 256]),
    ↪  activation='relu'))
    model.add(Dropout(rate=hp.Choice('dropout_dense', values=[0.2, 0.4, 0.5])))
    model.add(Dense(1, activation='sigmoid'))
    optimizer_choice = hp.Choice('optimizer', ['AdamW', 'RMSprop'])
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    if optimizer_choice == 'AdamW':
        optimizer = AdamW(learning_rate=learning_rate)
    else:
        optimizer = RMSprop(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
    ↪  metrics=['accuracy'])
    return model
# Loop through each subject as the genuine user
```

```
accuracies = []
eers = []
fars = []
frrs = []
for genuine_user_code in subject_codes:
    print(f"Evaluating for subject {genuine_user_code} as the genuine user...")
    # Load data for genuine user and imposters
    train_images_genuine, train_labels_genuine = load_images(genuine_user_code, [1],
    ↪  range(100), 1)
    test_images_genuine, test_labels_genuine = load_images(genuine_user_code, [2],
    ↪  range(100), 1)
    train_images_imposter = []
    train_labels_imposter = []
    test_images_imposter = []
    test_labels_imposter = []
    for subject_code in subject_codes:
        if subject_code != genuine_user_code:
            imp_train_images, imp_train_labels = load_images(subject_code, [2],
            ↪  range(1), 0)
            imp_test_images, imp_test_labels = load_images(subject_code, [1],
            ↪  range(1), 0)
            train_images_imposter.extend(imp_train_images)
            train_labels_imposter.extend(imp_train_labels)
            test_images_imposter.extend(imp_test_images)
            test_labels_imposter.extend(imp_test_labels)
    train_images_imposter = np.array(train_images_imposter)
    train_labels_imposter = np.array(train_labels_imposter)
    test_images_imposter = np.array(test_images_imposter)
    test_labels_imposter = np.array(test_labels_imposter)
    # Combine genuine and imposter data
    X_train = np.concatenate((train_images_genuine, train_images_imposter), axis=0)
    y_train = np.concatenate((train_labels_genuine, train_labels_imposter), axis=0)
    X_test = np.concatenate((test_images_genuine, test_images_imposter), axis=0)
    y_test = np.concatenate((test_labels_genuine, test_labels_imposter), axis=0)
    # Set the random seed for reproducibility
    seed = 1
    np.random.seed(seed)
    tf.random.set_seed(seed)
    # Define unique tuning directory for each subject
    subject_tuning_dir = Path(tuning_directory) / f'tuning_{genuine_user_code}'
    # Check if the subject's tuner data exists
    if subject_tuning_dir.exists() and any(subject_tuning_dir.iterdir()):
```

```
        print(f"Tuner data for {genuine_user_code} found. Restoring previous
        ↪ tuner.")
        overwrite_tuner = False
    else:
        print(f"No tuner data for {genuine_user_code} or data is corrupt. Starting
        ↪ fresh tuning.")
        overwrite_tuner = True
    # Initialize the tuner for the current subject
    tuner = kt.RandomSearch(
        build_model,
        objective='val_accuracy',
        directory=tuning_directory,
        project_name=f'tuning_{genuine_user_code}',
        seed=seed,
        overwrite=overwrite_tuner  # Decide whether to overwrite or restore previous
        ↪ tuning
    )
    if overwrite_tuner:
        tuner.search(X_train, y_train, epochs=20, validation_data=(X_test, y_test),
        ↪ verbose=0)
    # Retrieve the best model
    best_model = tuner.get_best_models(num_models=1)[0]
    # Evaluate on the test set and calculate metrics
    y_pred = (best_model.predict(X_test) > 0.5).astype("int32")
    acc = np.mean(y_pred.flatten() == y_test)
    accuracies.append(acc)
    # Get confusion matrix for FAR and FRR calculation
    conf_matrix = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = conf_matrix.ravel()
    # Calculate FAR and FRR
    far = fp / (fp + tn) if (fp + tn) > 0 else 0  # FAR: False positives / All
    ↪ negatives
    frr = fn / (fn + tp) if (fn + tp) > 0 else 0  # FRR: False negatives / All
    ↪ positives
    fars.append(far)
    frrs.append(frr)
    # Get scores for EER calculation
    scores = best_model.predict(X_test).flatten()
    user_scores = scores[y_test == 0]
    impostor_scores = scores[y_test == 1]
    eer = calculate_equal_error_rate(user_scores, impostor_scores)
    eers.append(eer)
    print(f"Classification Report for {genuine_user_code}:\n")
```

```
    print(classification_report(y_test, y_pred, target_names=['Imposter',
    ↪  'Genuine']))
    print(f"Confusion Matrix for {genuine_user_code}:\n{conf_matrix}\n")
    print(f"False Acceptance Rate (FAR): {far:.4f}")
    print(f"False Rejection Rate (FRR): {frr:.4f}")
    print(f"Equal Error Rate (EER) for {genuine_user_code}: {eer:.4f}\n")
# Calculate and print the average metrics across all subjects
average_accuracy = np.mean(accuracies)
average_eer = np.mean(eers)
average_far = np.mean(fars)
average_frr = np.mean(frrs)
print(f"Average Test Accuracy across all subjects: {average_accuracy * 100:.2f}%")
print(f"Average Equal Error Rate (EER) across all subjects: {average_eer:.4f}")
print(f"Average False Acceptance Rate (FAR) across all subjects: {average_far:.4f}")
print(f"Average False Rejection Rate (FRR) across all subjects: {average_frr:.4f}")
```