**Is It My Type?**

פרויקט גמר בסייבר:

נושא: AI

שם המגיש: רועי נחשון

תעודת זהות: 327757977

שמות המנחים: ירון יצחקי, תומר טלגם

תאריך הגשה: 14.6.23

Table of Contents

[Introduction 4](#_Toc137387918)

[Initiation 4](#_Toc137387919)

[Overview 4](#_Toc137387920)

[Target Customers 6](#_Toc137387921)

[Goals and Objectives 7](#_Toc137387922)

[Problems and Benefits 8](#_Toc137387923)

[Review of Existing Solutions 10](#_Toc137387924)

[Review of Project Technology 11](#_Toc137387925)

[Caveats and System Setup Restrictions 12](#_Toc137387926)

[Detailed description of the system (characterization) 15](#_Toc137387927)

[Detailed System Description 15](#_Toc137387928)

[*RNN* 15](#_Toc137387929)

[System’s Abilities 16](#_Toc137387930)

[Black-Box checks 17](#_Toc137387931)

[Time Management 18](#_Toc137387932)

[Risk Management 19](#_Toc137387933)

[In-Depth Breakdown of Client-Side Capabilities 19](#_Toc137387934)

[Authentication Experience: 19](#_Toc137387935)

[System Access: 20](#_Toc137387936)

[User Interface Interaction: 20](#_Toc137387937)

[Increased Security: 21](#_Toc137387938)

[Non-Disruptive User Experience: 21](#_Toc137387939)

[Structure / architecture of the project 22](#_Toc137387940)

[hardware description 22](#_Toc137387941)

[Description of Relevant Technology 24](#_Toc137387942)

[Flow Of Information 25](#_Toc137387943)

[Description of the Main Algorithms in the Project 26](#_Toc137387944)

[Description of the Development Environment 27](#_Toc137387945)

[Description of the Testing Environment 28](#_Toc137387946)

[Description of the Communication Protocol 29](#_Toc137387947)

[Description of the system screens 30](#_Toc137387948)

[Description of the Data Structures 31](#_Toc137387949)

[Overview of Weaknesses and Threats 33](#_Toc137387950)

[The Implementation of The Project 33](#_Toc137387951)

[Overview of System Modules and Interrelationships 33](#_Toc137387952)

[Imported Modules/Departments: 33](#_Toc137387953)

[Solving the algorithmic problems 36](#_Toc137387954)

[Full BlackBox Checks 38](#_Toc137387955)

[User’s Guide 39](#_Toc137387956)

[Installation 39](#_Toc137387957)

[Activation 39](#_Toc137387958)

[Personal summary / reflection 39](#_Toc137387959)

[Bibliography 42](#_Toc137387960)

[Appendices 42](#_Toc137387961)

[Code 42](#_Toc137387962)

[create\_train\_files.py 42](#_Toc137387963)

[train\_model.py 45](#_Toc137387964)

[user\_interface.py 48](#_Toc137387965)

[convert\_to\_windows.py 52](#_Toc137387966)

[test\_model.py 53](#_Toc137387967)

# Introduction

## Initiation

### Overview

The Typing Pattern Detection AI is an advanced system designed to analyze and recognize the typing patterns of computer users. It captures and saves these patterns, normalizes the data, and trains a model to identify and differentiate between different typists. The primary purpose of this AI is to detect unauthorized users typing on your computer, providing an additional layer of security and privacy.

#### Project Rationale

The Typing Pattern Detection AI project was initiated to address the growing concerns related to computer security and unauthorized access. Traditional methods like passwords and biometrics have their limitations, and there was a need for an innovative approach to identify and differentiate users based on their unique typing patterns. By leveraging machine learning and artificial intelligence techniques, this project aims to provide an effective solution to detect potential intruders and protect sensitive information.

#### Features and Functionality

The finished product offers the following key features:

1. Typing Pattern Capture: The AI captures and records the typing patterns of users while they interact with the computer.
2. Data Normalization: The captured typing patterns are preprocessed and normalized to remove any biases and ensure consistency in the dataset.
3. Training the Model: The normalized data is then utilized to train a machine learning model capable of recognizing and classifying different typing patterns.
4. User Identification: The trained model can identify and differentiate between authorized users and unauthorized individuals attempting to type on the computer.
5. User Interface (UI): The project includes a user-friendly UI that allows for easy interaction and configuration of the AI system.
6. Training and Testing Files: Separate files are provided for training and testing purposes, enabling the evaluation and refinement of the AI's performance.

#### Motivation and Challenges

There are several reasons why this project was chosen, including:

1. Enhanced Security: By implementing a typing pattern detection system, the project aims to strengthen the security measures in place, offering an additional layer of protection against unauthorized access.
2. User-Friendly Approach: Unlike traditional security methods that often require complex passwords or biometric scans, typing pattern detection offers a more natural and user-friendly way to identify individuals.
3. Potential Real-World Applications: The technology developed in this project could find applications in various domains, such as corporate security, computer forensics, and personal privacy protection.

Challenges that may be encountered during the project include:

1. Data Collection: Gathering a diverse and representative dataset of typing patterns can be challenging, as it requires cooperation from a wide range of users.
2. Noise and Variability: Typing patterns can be influenced by various factors, including external noise, physical conditions, and typing habits, which may introduce variability and affect the accuracy of the AI model.
3. User Adaptability: The AI system needs to adapt to changes in users' typing patterns over time, accounting for natural variations and accommodating improvements or changes in typing style.

By recognizing these challenges and proactively addressing them, the project aims to develop a robust and reliable typing pattern detection AI system.

### Target Customers

The Typing Pattern Detection AI system is intended for individuals and organizations who prioritize computer security and want to add an extra layer of protection to their systems. The system can be particularly beneficial for the following customers:

1. **Individual Users**: Individuals concerned about unauthorized access to their personal computers, especially when working in public spaces or shared environments.
2. **Corporate Organizations**: Companies aiming to enhance the security of their computer networks, safeguard sensitive data, and protect against potential insider threats.
3. **Government Agencies**: Government institutions that handle classified information and require stringent security measures to prevent unauthorized access.
4. **Educational Institutions**: Schools, colleges, and universities that want to ensure secure computer usage by students, faculty, and staff members.
5. **Research Organizations**: Institutes engaged in sensitive research and development work, where protecting intellectual property is crucial.
6. **Cybersecurity Professionals**: Experts in the field of cybersecurity who wish to explore innovative techniques and solutions for intrusion detection and prevention.

The system is designed to be user-friendly and adaptable to different environments, making it accessible to both technical and non-technical users. The customization options available through the user interface allow customers to configure the system according to their specific requirements.

It is important to note that while the Typing Pattern Detection AI can provide an additional layer of security, it should not be considered a standalone solution. It is meant to complement existing security measures and best practices, such as strong passwords, firewalls, and regular system updates.

### Goals and Objectives

The Typing Pattern Detection AI system aims to achieve the following goals:

1. **Enhance Security**: The primary goal of the system is to enhance computer security by detecting unauthorized users attempting to type on the computer. It provides an additional layer of protection against potential intruders and helps prevent unauthorized access to sensitive information.
2. **User Identification**: The system aims to accurately identify and differentiate between authorized users and unauthorized individuals based on their unique typing patterns. This enables proactive measures to be taken when an unauthorized user is detected, such as triggering alerts or initiating security protocols.
3. **Ease of Use**: The system is designed to be user-friendly and accessible to both technical and non-technical users. It provides a seamless and intuitive user interface for easy interaction, configuration, and monitoring of the AI system.
4. **Adaptability**: The system strives to adapt to changes in users' typing patterns over time. It accounts for natural variations and accommodates improvements or changes in typing style without compromising accuracy.
5. **Performance and Accuracy**: The system aims to achieve high performance and accuracy in detecting and classifying typing patterns. By utilizing machine learning techniques and continuously improving the model, it strives to minimize false positives and false negatives.
6. **Flexibility**: The system offers flexibility in terms of customization and integration. Customers can tailor the system to their specific needs and integrate it with existing security infrastructure, providing a comprehensive security solution.
7. **Documentation and Support**: Comprehensive documentation and user support materials are provided to assist customers in understanding and effectively utilizing the system. This includes a detailed README file, user guides, and access to technical support for any queries or issues that may arise.

By achieving these goals, the Typing Pattern Detection AI system aims to provide a reliable and effective solution for detecting unauthorized typing activity and ensuring computer security for individuals and organizations.

### Problems and Benefits

#### Problems:

The Typing Pattern Detection AI system addresses the following problems:

1. **Unauthorized Access**: Unauthorized individuals gaining access to computer systems can pose significant security risks, leading to data breaches, privacy violations, and potential financial or reputational damages.
2. **Weaknesses in Traditional Security Measures**: Traditional security measures like passwords or biometrics have their limitations and can be vulnerable to various attacks, such as password guessing, social engineering, or biometric spoofing.

#### Goals and Benefits:

The Typing Pattern Detection AI system aims to achieve the following benefits:

1. **Enhanced Security**: By accurately detecting unauthorized typing activity, the system provides an additional layer of security to computer systems, significantly reducing the risk of unauthorized access and potential security breaches.
2. **Proactive Intrusion Detection**: The system enables proactive measures to be taken upon detecting unauthorized typing, such as triggering alerts, locking the system, or initiating security protocols. This allows for timely response and mitigates the risk of data compromise or unauthorized activities.
3. **User-Friendly and Non-Intrusive**: Unlike traditional security measures that may require users to remember complex passwords or undergo intrusive biometric scans, the Typing Pattern Detection AI system operates in the background without causing inconvenience to the users. It seamlessly integrates into the user's workflow, requiring no additional effort or conscious authentication.
4. **Improved User Experience**: The system eliminates the need for repeated manual authentication and provides a seamless user experience. Users can access their computers without interruption, while the AI works silently in the background, ensuring their security.
5. **Cost and Time Savings**: Implementing the Typing Pattern Detection AI system can lead to cost and time savings by reducing the need for additional security measures, such as expensive biometric scanners or complex password management systems. It offers a cost-effective and efficient solution for enhancing security.

#### Services Provided:

The Typing Pattern Detection AI system provides the following services:

1. **Real-time Typing Pattern Detection**: The system continuously monitors typing activity in real-time, analyzing and identifying patterns to detect potential unauthorized users.
2. **User Identification and Classification**: The system accurately identifies and classifies users based on their unique typing patterns, distinguishing between authorized and unauthorized individuals.
3. **Alerts and Notifications**: Upon detecting unauthorized typing, the system can generate alerts or notifications, enabling immediate action to be taken to prevent unauthorized access or potential security breaches.
4. **User Interface (UI)**: The system offers a user-friendly interface for easy configuration, monitoring, and management of the AI system. Users can customize settings, view logs, and access relevant information through the UI.
5. **Documentation and Support**: Comprehensive documentation, including a detailed README file and user guides, is provided to assist users in understanding and effectively utilizing the system. Additionally, technical support is available to address any queries or issues that may arise.

By addressing the problems and providing these benefits and services, the Typing Pattern Detection AI system offers an effective, user-friendly, and cost-efficient solution for enhancing computer security and mitigating the risks associated with unauthorized access.

### Review of Existing Solutions

The Typing Pattern Detection AI system distinguishes itself from existing solutions and applications in the following ways:

1. **Alternative to Passwords and Biometrics**: While traditional security measures rely on passwords or biometrics for authentication, the Typing Pattern Detection AI system offers a unique approach. It leverages the individual's typing patterns, providing a non-intrusive and user-friendly method of identification, without the need for users to remember complex passwords or undergo biometric scans.
2. **Behavior-Based Authentication**: Unlike static authentication methods, such as passwords or fingerprints, the Typing Pattern Detection AI system focuses on dynamic user behavior. By analyzing the unique patterns and rhythm of an individual's typing, it creates a personalized profile that can be used to differentiate between authorized and unauthorized users.
3. **Continuous Monitoring and Proactive Detection**: The system continuously monitors typing activity in real-time, enabling proactive detection of unauthorized users attempting to access the computer. This offers a proactive security approach compared to conventional methods that rely on periodic authentication.
4. **Adaptability to User Changes**: The Typing Pattern Detection AI system is designed to adapt to changes in users' typing patterns over time. It accommodates natural variations and allows for adjustments in typing style, ensuring accurate identification even as users' behavior evolves.
5. **Customizability and Integration**: The system provides flexibility and customization options, allowing users to configure it according to their specific needs. It can be integrated with existing security infrastructure and complement other security measures, providing a comprehensive security solution.
6. **User-Friendly Interface**: The Typing Pattern Detection AI system offers a user-friendly interface that simplifies system configuration, monitoring, and management. It provides an intuitive and accessible platform for users to interact with the AI system without requiring technical expertise.
7. **Cost-Effective Solution**: Compared to expensive biometric scanners or complex password management systems, the Typing Pattern Detection AI system offers a cost-effective solution for enhancing security. It leverages existing hardware and software resources, reducing the need for additional investments.

While there may be existing solutions that utilize typing patterns for authentication or intrusion detection, the Typing Pattern Detection AI system aims to improve upon them by providing a more robust, adaptable, and user-friendly solution. It leverages the power of machine learning and artificial intelligence techniques to offer enhanced security and proactive identification of unauthorized users typing on the computer.

### Review of Project Technology

The Typing Pattern Detection AI project utilizes a combination of established technologies and techniques. While the concept of analyzing typing patterns is not new, the specific implementation and integration of machine learning and AI make this project innovative. Here is a review of the project's technology:

1. **Machine Learning and AI**: The project leverages machine learning algorithms and AI techniques to analyze and recognize typing patterns. These technologies enable the system to learn from data, classify patterns, and make accurate predictions based on the trained model.
2. **Data Preprocessing and Normalization**: The project employs data preprocessing and normalization techniques to ensure consistency and remove biases in the collected typing pattern data. This step enhances the accuracy and reliability of the AI model.
3. **User Interface (UI)**: The project includes a user-friendly UI that provides an intuitive platform for users to interact with the AI system. The UI allows users to configure settings, view logs, and access relevant information easily.
4. **Training and Testing Files**: Separate training and testing files are provided as part of the project. These files facilitate the training of the AI model on a representative dataset and the evaluation of its performance on unseen data.

### Caveats and System Setup Restrictions

1. **Dataset Diversity**: The accuracy and effectiveness of the Typing Pattern Detection AI system heavily rely on the diversity and quality of the dataset used for training. A more diverse dataset, comprising a wide range of typing styles and scenarios, can enhance the system's ability to generalize and detect unauthorized users accurately.
2. **Hardware and Software Compatibility**: The system's performance may be influenced by the hardware and software setup of the user's computer. Compatibility issues or limitations in processing power and memory may impact the system's ability to capture and analyze typing patterns effectively.
3. **Data Privacy and Ethics**: As the system captures and analyzes user typing patterns, it is essential to address data privacy concerns and ensure compliance with ethical guidelines. User consent and data anonymization should be implemented to protect individual privacy and prevent potential misuse of personal information.
4. **User Adaptability**: The system needs to adapt to changes in users' typing patterns over time. However, it may face challenges in accommodating sudden changes, such as injuries or disabilities that significantly alter typing behavior. Users should be aware of the system's limitations and be prepared to provide alternative authentication methods if needed.
5. **False Positives and False Negatives**: Like any pattern recognition system, the Typing Pattern Detection AI may encounter false positives (identifying authorized users as unauthorized) or false negatives (failing to identify unauthorized users). It is crucial to strike a balance between minimizing false alarms and ensuring accurate detection.

Overall, while the technology utilized in the Typing Pattern Detection AI project may not be entirely new, its implementation and integration provide an innovative approach to enhance computer security. The system setup requires attention to dataset diversity, hardware and software compatibility, data privacy, and user adaptability to maximize the system's effectiveness and mitigate potential limitations.

#### Delimitation of the Project

The Typing Pattern Detection AI project focuses on specific areas related to computer security, networks, and operating systems. It is important to understand the project's scope and the areas it addresses, as well as the areas it does not cover. Here are the delimitations of the project:

#### Areas Covered by the Project:

1. **Computer Security**: The project aims to enhance computer security by detecting unauthorized typing activity and preventing unauthorized access to systems. It focuses on the identification and classification of users based on their typing patterns.
2. **Typing Pattern Analysis**: The project involves the collection, preprocessing, and analysis of typing patterns to develop an AI model capable of recognizing and distinguishing between authorized and unauthorized users. It utilizes machine learning and AI techniques for pattern recognition and prediction.
3. **Network and Operating System Integration**: The project can be integrated into computer networks and operating systems to provide an additional layer of security. It operates at the system level, monitoring typing activity and triggering security measures when unauthorized users are detected.

#### Areas Not Covered by the Project:

1. **Network Protocols and Encryption**: The project does not deal with the intricacies of network protocols, encryption, or data transmission. It focuses solely on the typing pattern analysis and user identification aspects of computer security.
2. **Hardware-level Security**: The project does not address hardware-level security mechanisms, such as physical access controls or tamper-resistant devices. It operates at the software level, utilizing typing patterns as a means of authentication.
3. **Application-Specific Security**: While the project enhances computer security, it does not specifically target the security of individual applications or software programs. It provides a system-level security measure applicable to the entire computer system.
4. **Intrusion Detection and Prevention**: While the project can help detect unauthorized users attempting to type on the computer, it does not cover all aspects of intrusion detection and prevention. It focuses on the typing pattern analysis as a means of identification and prevention of unauthorized access.

It is important to note that the Typing Pattern Detection AI project should be used as a complementary security measure and not as a standalone solution. It should be integrated with other security practices and best practices, such as strong passwords, firewalls, and regular system updates, to ensure comprehensive computer security.

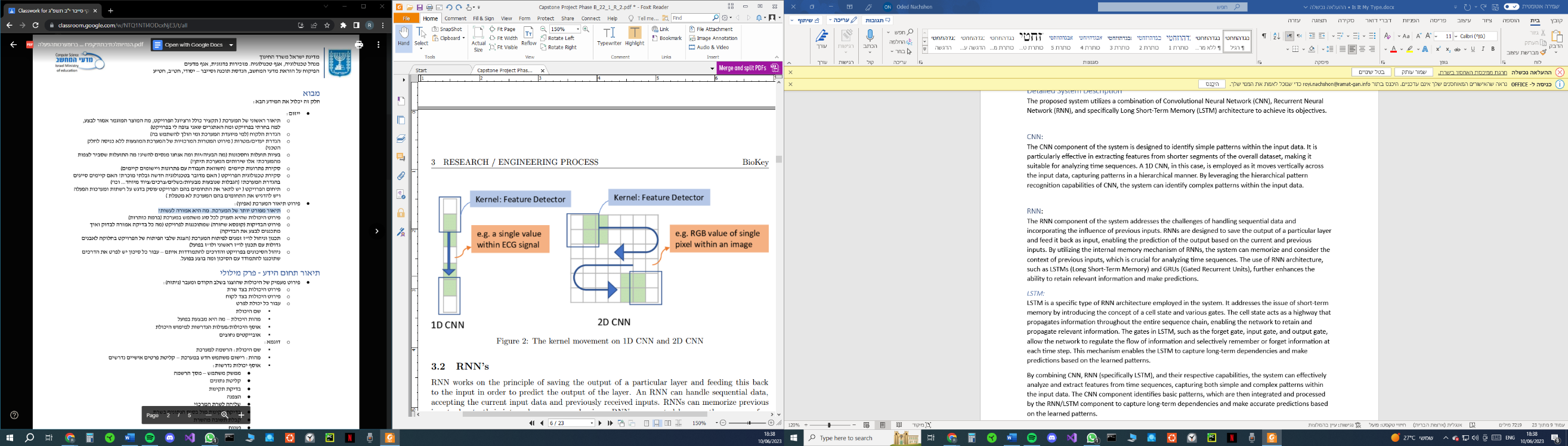
By understanding the delimitations of the project, users can have a clear understanding of the areas it covers and the areas where additional security measures may be required.

## Detailed description of the system (characterization)

### Detailed System Description

The proposed system utilizes a combination of Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and specifically Long Short-Term Memory (LSTM) architecture to achieve its objectives.

#### CNN:

The CNN component of the system is designed to identify simple patterns within the input data. It is particularly effective in extracting features from shorter segments of the overall dataset, making it suitable for analyzing time sequences. A 1D CNN, in this case, is employed as it moves vertically across the input data, capturing patterns in a hierarchical manner. By leveraging the hierarchical pattern recognition capabilities of CNN, the system can identify complex patterns within the input data.

#### RNN:

The RNN component of the system addresses the challenges of handling sequential data and incorporating the influence of previous inputs. RNNs are designed to save the output of a particular layer and feed it back as input, enabling the prediction of the output based on the current and previous inputs. By utilizing the internal memory mechanism of RNNs, the system can memorize and consider the context of previous inputs, which is crucial for analyzing time sequences. The use of RNN architecture, such as LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units), further enhances the ability to retain relevant information and make predictions.

##### LSTM:

תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

התיאור נוצר באופן אוטומטיLSTM is a specific type of RNN architecture employed in the system. It addresses the issue of short-term memory by introducing the concept of a cell state and various gates. The cell state acts as a highway that propagates information throughout the entire sequence chain, enabling the network to retain and propagate relevant information. The gates in LSTM, such as the forget gate, input gate, and output gate, allow the network to regulate the flow of information and selectively remember or forget information at each time step. This mechanism enables the LSTM to capture long-term dependencies and make predictions based on the learned patterns.

By combining CNN, RNN (specifically LSTM), and their respective capabilities, the system can effectively analyze and extract features from time sequences, capturing both simple and complex patterns within the input data. The CNN component identifies basic patterns, which are then integrated and processed by the RNN/LSTM component to capture long-term dependencies and make accurate predictions based on the learned patterns.

### System’s Abilities

End Users:

1. **Authentication Experience**: End users benefit from a seamless and non-intrusive authentication process. They can simply type on the computer, and the Typing Pattern Detection AI system analyzes their typing patterns to authenticate their identity without requiring additional authentication steps like passwords or biometrics.
2. **System Access**: Authorized end users can gain uninterrupted access to the computer system once their typing patterns are recognized and authenticated by the system. They can perform their tasks and use the system as usual without any interference from the Typing Pattern Detection AI system.
3. **User Interface Interaction**: End users have access to the user interface provided by the Typing Pattern Detection AI system. Through this interface, they can view their typing pattern history, review any security-related notifications or alerts, and modify their own profile settings if granted permission by the system administrator.
4. **Increased Security**: By utilizing the Typing Pattern Detection AI system, end users experience an enhanced level of security. The system helps to prevent unauthorized access by detecting and differentiating between authorized and unauthorized users based on their typing patterns. This reduces the risk of unauthorized individuals gaining access to the system and helps protect sensitive information.
5. **Non-Disruptive User Experience**: The Typing Pattern Detection AI system operates in the background, analyzing typing patterns without interrupting or affecting the end users' normal workflow. End users can continue to type and use the computer system without any noticeable changes or disruptions caused by the system.

It's important to note that the capabilities granted to end users are primarily related to their authentication experience, system access, and user interface interaction. The Typing Pattern Detection AI system aims to provide end users with a convenient and secure authentication method, ensuring a smooth and uninterrupted user experience while maintaining the desired level of computer system security.

### Black-Box checks

1. **Input Validation Test**:
   * Objective: To verify that the system correctly handles different types of input and properly validates the input data.
   * Test Execution: Generate various test cases with different input data, including normal inputs, invalid inputs, and edge cases. Observe how the system handles each input, ensuring it performs appropriate validation and handles errors gracefully.
2. **False Positive/Negative Test**:
   * Objective: To assess the system's ability to minimize false positives and false negatives.
   * Test Execution: Design test cases that include authorized users with different typing behaviors and unauthorized users attempting to mimic authorized users' typing patterns. Measure the system's performance in correctly identifying authorized users and detecting unauthorized users, ensuring a balance between minimizing false positives (legitimate users flagged as unauthorized) and false negatives (unauthorized users not detected).
3. **Performance and Scalability Test**:
   * Objective: To evaluate the system's performance and scalability under varying loads and user scenarios.
   * Test Execution: Simulate different usage scenarios, including varying numbers of concurrent users and typing patterns with different complexities. Measure the system's response time, resource utilization, and accuracy under different load conditions to ensure it performs efficiently and scales well.

### Time Management

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Activity | Planned starting time | Planned finishing time | Actual starting time | Actual finishing time | Notes |
| Initiation | 1.3.23 | 8.3.23 | 3.3.23 | 18.3.23 | I was late |
| Characterization | 12.3.23 | 15.3.23 | 18.3.23 | 20.3.23 | I was late but was closing in |
| Measuring writing times | 22.3.23 | 30.3.23 | 2.4.23 | 3.4.23 | I was late |
| Model | 4.4.23 | 18.4.23 | 4.4.23 | 20.4.23 | I was late |
| POC | 21.4.23 | 25.4.23 | 22.4.23 | 26.4.23 | I was late |
| Final | 27.4.23 | 13.5.23 | 28.4.23 | 10.5.23 | I was late |
| Cleaning | 19.5.23 | 23.5.23 | 12.5.23 | 2.6.23 | I was early |
| ReadMe | 25.5.23 | 27.5.23 | 8.6.23 | 11.6.23 | I was late |

### Risk Management

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| The risk | Detail | Threat level | How to deal | What went down | Date |
| Time management | Unfinished project | Hard | Staying ahead of times | Coding became priority | 1.5.23 |
| Model reliability | Model isn’t reliable | Mid | Read articles | Consult experts | 5.5.23 |
| Bugs and inefficiencies | Clean the code | Easy | Debugging and deleting code | Debugging and deleting code | 3.6.23 |

# In-Depth Breakdown of Client-Side Capabilities

## Authentication Experience:

* **Essence**: This ability enables end users to authenticate their identity through their typing patterns without additional authentication steps.
* **Actions**:
  + Collect and analyze typing patterns in real-time.
  + Compare the observed typing patterns with the user's registered typing profile.
  + Determine the authenticity of the user based on the similarity of typing patterns.
* **Necessary Objects**:
  + Typing pattern data collected from the user.
  + Registered typing profiles of authorized users.

## System Access:

* **Essence**: This ability allows authorized end users to gain uninterrupted access to the computer system once their typing patterns are recognized and authenticated.
* **Actions**:
  + Verify the authenticity of the user's typing patterns.
  + Grant access privileges to authorized users.
  + Monitor typing patterns for any changes or anomalies during the user's session.
* **Necessary Objects**:
  + Typing pattern data of authorized users.
  + Access control settings and privileges.

## User Interface Interaction:

* **Essence**: This ability enables end users to interact with the user interface provided by the Typing Pattern Detection AI system.
* **Actions**:
  + View and access the user interface elements.
  + Navigate through different sections and features of the user interface.
  + Modify personal profile settings if permitted.
* **Necessary Objects**:
  + User interface components and elements.
  + User profile settings and preferences.

## Increased Security:

* **Essence**: This ability enhances the security of the computer system by leveraging typing patterns for user authentication.
* **Actions**:
  + Detect and differentiate between authorized and unauthorized users based on their typing patterns.
  + Generate alerts or trigger security measures upon detecting unauthorized typing activity.
  + Prevent unauthorized access to the system.
* **Necessary Objects**:
  + Typing pattern data for authorized and unauthorized users.
  + Security protocols and measures.

## Non-Disruptive User Experience:

* **Essence**: This ability ensures that the Typing Pattern Detection AI system operates in the background without causing disruptions to the end users' workflow.
* **Actions**:
  + Analyze typing patterns without interrupting the user's normal typing experience.
  + Maintain a seamless authentication process without additional manual steps.
  + Minimize any noticeable changes or disruptions caused by the system.
* **Necessary Objects**:
  + System processes and algorithms for real-time analysis.
  + User feedback and usability testing.

These client-side capabilities in the Typing Pattern Detection AI system provide a seamless and secure authentication experience, enable system access for authorized users, facilitate user interface interaction, enhance overall security, and ensure a non-disruptive user experience. By leveraging typing patterns, the system strengthens computer security and delivers a user-friendly authentication process for end users.

# Structure / architecture of the project

## hardware description

Only a single computer is needed. Nothing more.

1. **Computer System**:
   * This forms the main hardware component on which the Typing Pattern Detection AI system will be deployed.
   * It consists of a CPU, GPU, memory, storage devices, and input/output peripherals.
2. **Database Server**:
   * This component hosts the existing database of users, which contains their typing pattern data.
   * The database server provides storage and retrieval capabilities for user data.
3. **User Data Integration**:
   * The user data integration process involves combining the existing user database with the main user data using the "create\_train\_file.py" Python script.
   * This script retrieves user data from the existing database and merges it with the main user data for further processing.
4. **Data Normalization**:
   * The combined user data, obtained from the integration process, is then passed through a normalization process.
   * The normalization aims to standardize the data, ensuring consistency and compatibility for subsequent processing.
5. **Model Training**:
   * The normalized data is used to train a machine learning model that will analyze and identify typing patterns.
   * The "train\_model.py" Python script is responsible for training the model using the normalized data.
   * The trained model captures the patterns and characteristics of authorized users' typing behavior.
6. **User Interface**:
   * The "user\_interface.py" Python script provides a graphical user interface (UI) for end users to interact with the Typing Pattern Detection AI system.
   * The UI allows users to input their typing patterns for authentication and provides visual feedback on the authentication process.
7. **Model Testing**:
   * The "test\_model.py" Python script utilizes the trained model to analyze and validate the typing patterns of users.
   * It compares the user's input with the patterns stored in the model to determine the authenticity of the user.

The connections between these components involve the flow of data and execution of scripts. The "create\_train\_file.py" script combines the existing user database with the main user data, which is then passed to the "train\_model.py" script for model training. The trained model is saved for later use by the "user\_interface.py" script, which provides the UI for user interaction. The "test\_model.py" script uses the trained model to analyze and verify the typing patterns entered through the UI.

Overall, this architecture allows for the integration of existing user data, training of a machine learning model, and providing a user-friendly interface for authentication using typing pattern analysis.

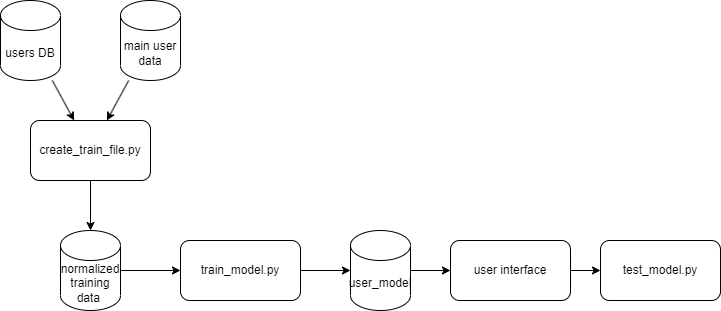
## Description of Relevant Technology

The Typing Pattern Detection AI system utilizes various technologies for its implementation. Here is a description of the relevant technologies:

1. **Programming Language**:
   * The system is developed using the Python programming language.
   * Python provides a wide range of libraries and frameworks for data processing, machine learning, and user interface development.
   * Its simplicity, readability, and extensive community support make it an ideal choice for implementing AI-based systems.
2. **Operating System**:
   * The system can be developed and deployed on multiple operating systems, including Windows, macOS, and Linux distributions.
   * The choice of the operating system depends on the specific requirements and preferences of the users and developers.
3. **Communication**:
   * The Typing Pattern Detection AI system may involve communication between different components.
   * The communication can be achieved through inter-process communication (IPC) mechanisms provided by the operating system, such as sockets, pipes, or shared memory.
   * Additionally, the system may utilize communication protocols, such as HTTP or WebSocket, for communication between the user interface and the backend components.

The selected technologies, such as Python programming language and the choice of operating system, enable the system to effectively process and analyze typing patterns, train models, and provide a user-friendly interface. The use of inter-process communication mechanisms and protocols allows for seamless communication between different components of the system.

## Flow Of Information



## Description of the Main Algorithms in the Project

In the Typing Pattern Detection AI project, the main goal is to develop an algorithm that can analyze and identify typing patterns for user authentication. The algorithmic problem involves finding an effective approach to recognize and differentiate between authorized and unauthorized users based on their typing behavior. Here is an overview of the formulation, existing algorithms, and the chosen solution in the project:

* **Formulation and Analysis of the Algorithmic Problem**:
  + The algorithmic problem in this project revolves around developing a model that can accurately classify and authenticate users based on their typing patterns.
  + It requires analyzing various features of typing behavior, such as key press timings, key hold durations, and inter-key intervals, to capture the unique characteristics of individual users.
  + It is very hard to find anomalies while giving the model enough data to learn the behavior of the subject.
* **Description of Existing Algorithms to Solve the Problem**:
  + Initially, the project explored the use of popular machine learning algorithms such as K-Nearest Neighbors (KNN) and Decision Forests.
  + KNN algorithm attempts to classify users based on the similarity of their typing patterns to previously observed patterns in the training data.
  + Decision Forests (e.g., Random Forest) employ an ensemble of decision trees to classify users based on various features of their typing behavior.
* **Reference to relevant sources:**
* Buffalo dataset: This dataset is collected from 148 subjects in 3 separate sessions. each session takes about 50 minutes and contains about 5.7k keystrokes. There are 28 days in average time intervals between sessions. Four different types of keyboards are used across sessions.
* **Reviewing the Chosen Solution**:
  + Despite attempting the use of KNN and Decision Forest algorithms, they did not yield satisfactory results in accurately identifying and authenticating users based on their typing patterns.
  + As a result, the project moved towards exploring other solutions and experimented with the One-Class Support Vector Machine (SVM) algorithm.
  + However, the One-Class SVM approach also did not meet the desired performance criteria.

## Description of the Development Environment

The development of the Typing Pattern Detection AI system requires specific tools and libraries to facilitate the coding and testing process. Here is a list of the development tools needed for development:

1. **Python**: The development environment is based on the Python programming language, which provides the necessary syntax and features for implementing the system.
2. **Text Editor/IDE**: I wrote my code using Pycharm.
3. **Libraries and Dependencies**: Several libraries and dependencies are needed for the development of the Typing Pattern Detection AI system. These libraries can be installed using the requirements file, which contains the following libraries:
   * random: Used for generating random numbers or samples.
   * h5py: A package to interact with the HDF5 file format.
   * ast: Provides tools for working with Abstract Syntax Trees in Python.
   * os: Provides a way to interact with the operating system, e.g., file handling and directory operations.
   * numpy: A powerful library for numerical computing in Python.
   * scikit-learn: A machine learning library that provides various algorithms and tools for data analysis.
   * pynput: Enables monitoring and controlling input devices, such as keyboards and mice.
   * tkinter: A standard Python GUI toolkit for creating graphical user interfaces.
   * keras: A high-level neural networks library built on top of TensorFlow, used for building and training models.
   * tensorflow: An open-source machine learning framework used for deep learning tasks.

## Description of the Testing Environment

The testing environment for the Typing Pattern Detection AI system requires the following tools and resources:

1. **Testing Data**: A collection of test data representing various typing patterns is needed to evaluate the accuracy and performance of the system. This data can be obtained from real-world users or generated synthetically for controlled testing.
2. **Test Scripts**: Test scripts are created to automate the testing process. These scripts simulate user inputs and evaluate the system's response, checking if the typing patterns are correctly identified and authenticated.

By utilizing the above development tools and setting up a dedicated testing environment, the Typing Pattern Detection AI system can be thoroughly tested for its accuracy, performance, and robustness.

## Description of the Communication Protocol

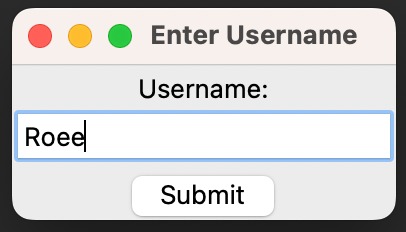
In the Typing Pattern Detection AI system, there is a need for communication between different components, such as the user interface, data processing modules, and the machine learning model. However, in the specific project you described, there is no explicit communication protocol or messaging system mentioned. Therefore, there are no structured messages flowing within the system.

It is important to note that the absence of a communication protocol in your project does not necessarily imply a limitation. The lack of a specific protocol indicates that the components interact directly or rely on function calls and data passing within the program without a standardized messaging format.

Instead of a structured messaging system, the different components of the system interact through function calls, method invocations, and data sharing within the program itself. For example, the user interface module may call functions from the data processing module to process the collected typing data, normalize it, and train the machine learning model. The trained model can then be used for authentication purposes.

Without a specific communication protocol, the system architecture relies on direct interaction and integration of the components, allowing for a more seamless flow of data and control within the program. However, if future expansions or integrations require inter-component communication, it may be necessary to consider implementing a structured communication protocol to ensure efficient and standardized data exchange between the components.

## Description of the system screens

תמונה שמכילה טקסט, צילום מסך, גופן, מולטימדיה

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, צילום מסך, מולטימדיה, גופן

התיאור נוצר באופן אוטומטיThe proposed system consists of three screens, each serving a specific purpose and providing a user interface for interacting with the system.

Username Input Screen: The first screen is a simple interface where the user is prompted to enter their username. It typically consists of a text input field where the user can enter their username and a submit button to proceed to the next screen. This screen serves to identify the user and associate their input with their unique identifier.

Recording Option Screen: The second screen presents the user with two radio buttons, allowing them to choose whether they want to record their input or not. This screen is designed to give the user control over whether their typing patterns will be captured and used by the system. If the user decides not to record their input, the system may display a message to inform them that their input will not be stored.

It's important to note that if the user doesn't exit the system properly, i.e., by submitting their username and choosing the recording option, the system may display a warning or reminder to ensure the user takes the necessary actions.

Output Screen: The third screen is the main output interface of the system. It typically displays the predictions generated by the trained model based on the user's typing patterns. This screen may include an output box where the predicted values or text is displayed, providing the user with real-time feedback or information based on their typing behavior. The screen may also include additional elements, such as a text indicator or visual cues, to indicate the system's decision or classification based on the user's input.

Overall, these three screens provide an intuitive and user-friendly interface for the users to interact with the system, enter their username, choose whether to record their input, and view the predictions or outputs generated by the AI model.

## Description of the Data Structures

The Typing Pattern Detection AI system utilizes various data structures to store and process different types of information. Here is a breakdown of the data structures involved:

1. **Initial Users Database** (txt file):
   * This database contains the typing patterns of multiple users and serves as the initial reference for user authentication.
   * The structure of each record in the database is an array with the following fields:
     + Key 1: The first key pressed by the user.
     + Key 2: The second key pressed by the user.
     + Timing Float 1: A float value representing the timing or duration of the first key press.
     + Timing Float 2: A float value representing the timing or duration of the second key press.
     + Timing Float 3: A float value representing the timing or duration between the first and second key presses.
     + Timing Float 4: A float value representing the timing or duration between the second and third key presses.
   * Example values for a record in the initial users database: **[164, 9, 0.219, 0.124, 0.11, -0.109]**
2. **Main User Database** (txt file):
   * This database contains the typing patterns of the main user who is currently using the system.
   * Similar to the initial users database, each record in this database has the following structure:
     + Key 1: The first key pressed by the main user.
     + Key 2: The second key pressed by the main user.
     + Timing Float 1: A float value representing the timing or duration of the first key press.
     + Timing Float 2: A float value representing the timing or duration of the second key press.
     + Timing Float 3: A float value representing the timing or duration between the first and second key presses.
     + Timing Float 4: A float value representing the timing or duration between the second and third key presses.
   * Example values for a record in the main user database: **[78, 79, 0.14441466331481934, 0.12945318222045898, 0.0945744514465332, -0.04984021186828613]**
3. **Normalized Data** (h5 file):
   * After processing and normalizing the typing patterns, the data is saved in an h5 file format.
   * The structure and organization of the normalized data may depend on the specific requirements and techniques used for normalization.
4. **Trained Model** (h5 file):
   * The trained machine learning model, which is used for identifying and authenticating typing patterns, is saved in an h5 file format.
   * The exact structure and organization of the model depend on the chosen architecture and the framework used for training, such as Keras.

It is important to note that the specific details and structure of the databases and files may vary based on the implementation and requirements of the Typing Pattern Detection AI system. The provided examples give a general idea of the fields and values involved in the data structures but may not reflect the entire complexity of the system's data representation.

## Overview of Weaknesses and Threats

In the Typing Pattern Detection AI system, it is essential to consider potential weaknesses and threats that may impact the system's security and functionality. However, my system is not web-connected and does not involve external network communication, some of the common vulnerabilities and threats related to web applications and network protocols are not applicable.

# The Implementation of The Project תחתית הטופס

## Overview of System Modules and Interrelationships

In the implementation of the Typing Pattern Detection AI system, there are several modules and imported libraries that play a role in the system's functionality. Here is an overview of these modules and their interrelationships:

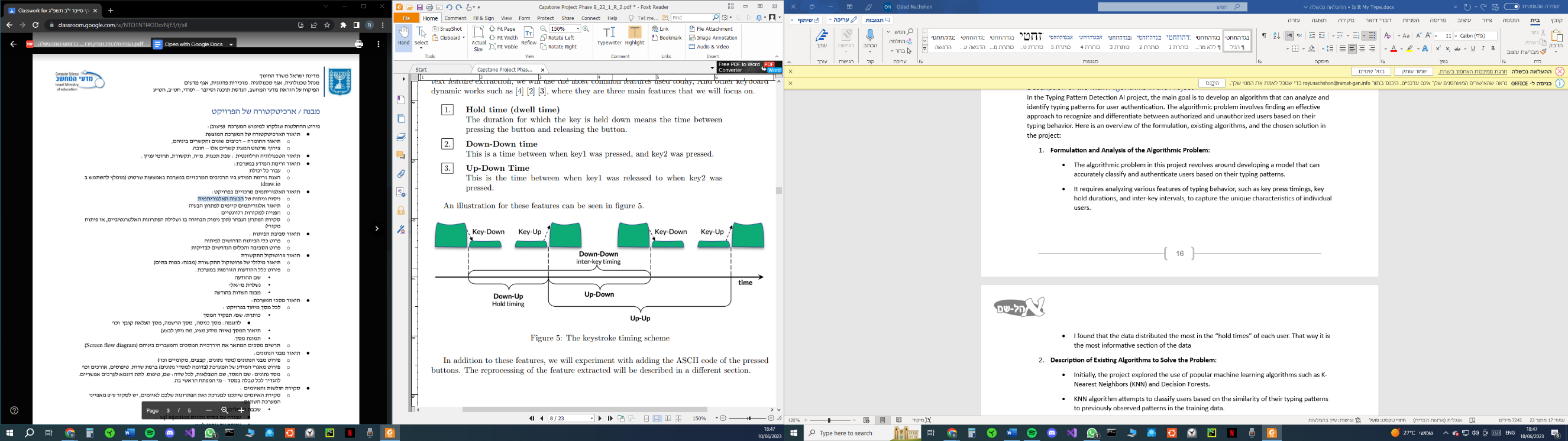
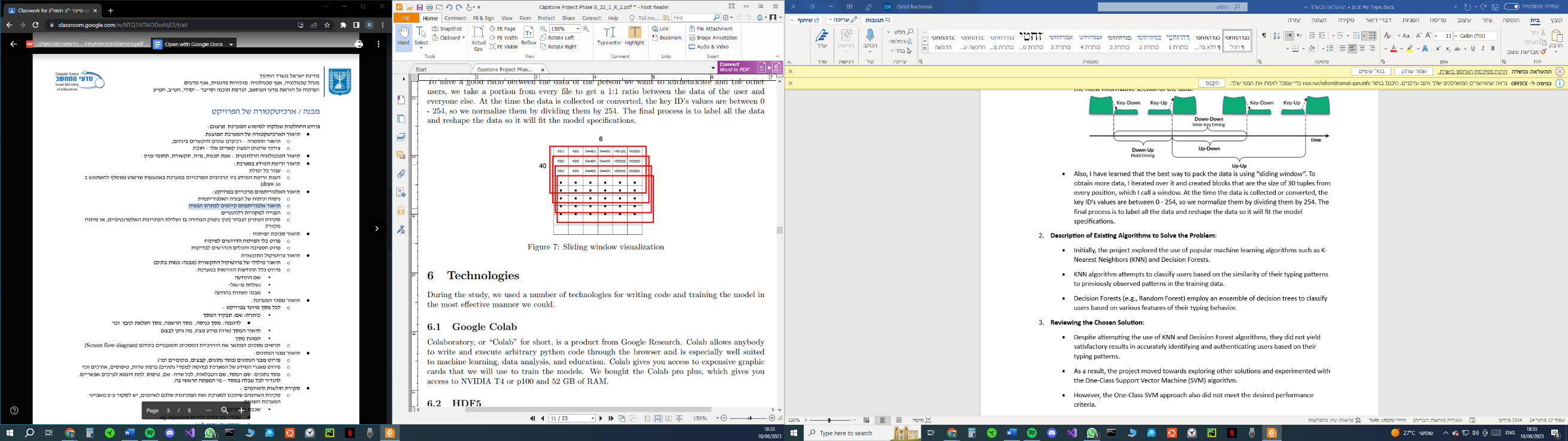
### Imported Modules/Departments:

1. **random**:
   * This module is imported to enable randomization in the system. It is used to slice the data randomly, ensuring randomness in the training and testing datasets.
2. **h5py**:
   * The Hierarchical Data Format version 5 (HDF5) is an open-source file format that supports large, complex, heterogeneous data. HDF5 uses a” file directory” like structure that allows you to organize data within the file in many different structured ways, as you might do with files on your computer.
3. **ast**:
   * The ast module is imported to facilitate the literal evaluation of lines in the data files. It is used to convert string representations of arrays into actual array objects for further processing.
4. **os**:
   * The os module provides functionalities related to the operating system. It is utilized in the system to perform various tasks such as locking the screen after a certain number of false identifications and accessing files on the local system.
5. **numpy**:
   * The numpy library is imported to enable efficient handling and manipulation of multi-dimensional arrays. It is used in the system to arrange and organize the data in a structured format for further analysis and processing.
6. **scikit-learn**:
   * The scikit-learn library is utilized in the system for various machine learning tasks. It provides functionalities for data preprocessing, shuffling of data, and training of machine learning models.
7. **pynput**:
   * The pynput module is imported to enable recording of keyboard events. It allows the system to capture the typing patterns of the user and process them for analysis and authentication purposes.
8. **tkinter**:
   * The tkinter library is used for developing the user interface of the system. It provides GUI components and functionalities, allowing users to interact with the system through a graphical interface.
9. **keras**:
   * The keras library is imported to enable the loading and training of the machine learning model. It provides high-level APIs and utilities for building and training neural networks.
10. **tensorflow**:
    * The tensorflow library is imported to support the functionalities of keras. It is an open-source machine learning framework that provides a computational backend for building and training deep learning models.

These imported modules and libraries enhance the system's capabilities by providing various functionalities required for data manipulation, machine learning, user interface development, and system interaction.

It is important to note that the system does not include additional self-developed modules or departments beyond these imported libraries.

## Solving the algorithmic problems

* I found that the data distributed the most in the “hold times” of each user. That way it is the most informative section of the data.
* Also, I have learned that the best way to pack the data is using “sliding window”. To obtain more data, I iterated over it and created blocks that are the size of 30 tuples from every position, which I call a window. At the time the data is collected or converted, the key ID’s values are between 0 - 254, so we normalize them by dividing them by 254. The final process is to label all the data and reshape the data so it will fit the model specifications.
* **Keras Models Sequential**:

תמונה שמכילה טקסט, צילום מסך, תוכנה, דף אינטרנט

התיאור נוצר באופן אוטומטיFirst, we have one 1D Conv layer with 32 features and a kernel size of 2, then 2 LSTM layers with 32 units and a dropout of 0.5 after each LSTM layer. The results are as follows:

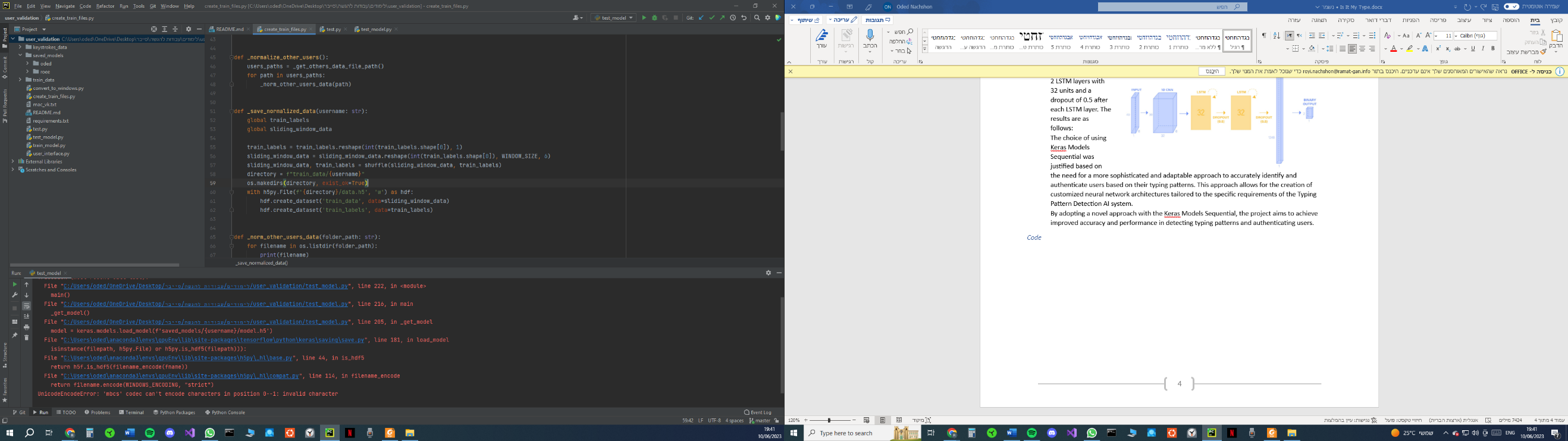
The choice of using Keras Models Sequential was justified based on the need for a more sophisticated and adaptable approach to accurately identify and authenticate users based on their typing patterns. This approach allows for the creation of customized neural network architectures tailored to the specific requirements of the Typing Pattern Detection AI system.

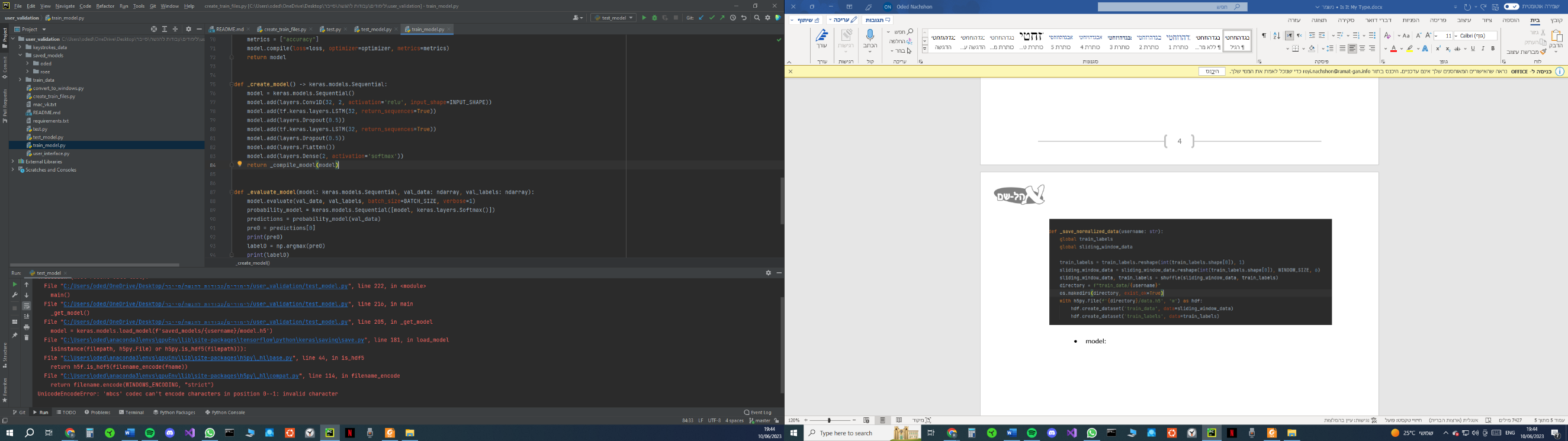
By adopting a novel approach with the Keras Models Sequential, the project aims to achieve improved accuracy and performance in detecting typing patterns and authenticating users.

#### Code

* תמונה שמכילה טקסט, תוכנה, תכונות מולטימדיה, צילום מסך

  התיאור נוצר באופן אוטומטיSliding windows:



* model:

## Full BlackBox Checks

1. **Input Validation Test**:
   * Objective: To verify that the system correctly handles different types of input and properly validates the input data.
   * Test Execution: Generate various test cases with different input data, including normal inputs, invalid inputs, and edge cases. Observe how the system handles each input, ensuring it performs appropriate validation and handles errors gracefully.
   * The edge cases were taken care of, and the system alerts the user to take the necessary actions.
2. **False Positive/Negative Test**:
   * Objective: To assess the system's ability to minimize false positives and false negatives.
   * Test Execution: Design test cases that include authorized users with different typing behaviors and unauthorized users attempting to mimic authorized users' typing patterns. Measure the system's performance in correctly identifying authorized users and detecting unauthorized users, ensuring a balance between minimizing false positives (legitimate users flagged as unauthorized) and false negatives (unauthorized users not detected).
   * The system correctly identified the user’s typing behavior and predicted them right.
3. **Performance and Scalability Test**:
   * Objective: To evaluate the system's performance and scalability under varying loads and user scenarios.
   * Test Execution: Simulate different usage scenarios, including varying numbers of concurrent users and typing patterns with different complexities. Measure the system's response time, resource utilization, and accuracy under different load conditions to ensure it performs efficiently and scales well.

# User’s Guide

## Installation

The system should run on python 3.8. There is an requirements.txt file to make the package installations easier, with the required version for each one.

## Activation

* Once the test\_model.py is activates, a screen of input will show. there the user will have to submit his name.
* Then, another screen will appear, that asks if you want to record the input or not. \*It is important to state that if the user chooses “yes”, the system will not predict the input, but just put it in the files. For the system to predict accurately the user must type 10,000+ characters.
* If the user chooses “no” the system will predict the behavior by the inputted username. The user can choose rather to lock the screen after 10 fails or not by checking the box in the last window.
* The system will detect edge cases such as not user inputs or model ready, and let the user know.

# Personal summary / reflection

During the development of this project, I encountered both successes and challenges that contributed to my personal growth as a developer. The work process involved various stages, including data collection, preprocessing, model training, and user interface development.

One of the major successes was successfully implementing the AI model using the Keras library. The model was able to effectively analyze and predict typing patterns based on the collected data. Additionally, the development of the user interface using Tkinter provided a visually appealing and interactive experience for the users.

However, I also faced some challenges and difficulties along the way. One of the challenges was selecting the appropriate machine learning algorithms for the task. Initially, I experimented with KNN and decision trees, but they didn't yield satisfactory results. It required further exploration and research to identify the suitable algorithm, which eventually led to the implementation of a sequential model in Keras.

The learning process throughout the project was extensive. I gained knowledge in areas such as data preprocessing, feature extraction, model training, and user interface development. I had to independently learn and understand concepts related to neural networks, specifically CNN and RNN architectures. This project allowed me to expand my knowledge and skills in machine learning and software development.

To continue the project, I plan to further explore different neural network architectures and optimization techniques to enhance the model's performance. Additionally, I will continue to stay updated with the latest advancements in machine learning and programming languages to incorporate new features and improve the overall system.

Throughout the project, I sought help from experts in the field of machine learning to gain insights and guidance. I also engaged in information sharing and peer learning through online forums and communities, which proved beneficial in overcoming obstacles and discovering new approaches.

In hindsight, if I had the opportunity to implement parts of the project differently, I would invest more time in the initial algorithm selection phase. Exploring a wider range of algorithms and conducting thorough experimentation could have potentially led to better initial results and saved time in the long run.

With additional resources, the project could be improved in various ways. Mostly by improving the user interface. At first it was planned to run on windows, but when I switched to Mac the UI didn’t cooperate. I had to adjust it by a short notice, and I think by having more time I could of improved the look of the final product.

Self-study questions for students to consider:

What other machine learning algorithms could be suitable for analyzing typing patterns?

How can data preprocessing techniques be applied to improve the model's performance?

Are there any other features or factors that could be considered in capturing and analyzing typing behavior?

How can the user interface be further enhanced to provide a more intuitive and user-friendly experience?

What other applications or domains could benefit from the analysis of typing patterns using AI?

I want to thank my friends that were there to help me and guide me when needed. I also want to thank my teachers that they were there around the clock.

# Bibliography

* Stack Overflow
* Geeks For Geeks
* <https://keras.io/guides/sequential_model/>
* GPT

# Appendices

## Code

### create\_train\_files.py

from sklearn.utils import shuffle  
import numpy as np  
import os  
import ast  
import h5py  
import random  
  
PATH = "keystrokes\_data/"  
WINDOW\_SIZE = 30  
train\_labels = np.array([])  
sliding\_window\_data = np.array([])  
  
  
def create\_train\_file(username: str):  
 *"""  
 Creates the h5 file for the model to work with from the txt files.* ***:param*** *username: A string of the user name, to create the files names.* ***:return****: Saves the h5 file.  
 """* \_normalize\_main\_user(username)  
  
 \_normalize\_other\_users()  
  
 \_save\_normalized\_data(username)  
 print("DONE!")  
  
  
def \_normalize\_main\_user(username: str):  
 *"""  
 Normalize the main user txt file.* ***:param*** *username: A string of the user name* ***:return****: None  
 """* user\_data = np.array([])  
 if os.path.exists(PATH + username):  
 for filename in os.listdir(PATH + username):  
 print(filename)  
 filepath = os.path.join(PATH + username, filename)  
 new\_data = np.array(\_get\_file\_data\_type(filepath))  
 user\_data = np.append(user\_data, new\_data)  
  
 \_finish\_norm\_main\_user(user\_data)  
 else:  
 print("No such user files!")  
 quit()  
  
  
def \_finish\_norm\_main\_user(user\_data: np.ndarray):  
 *"""  
 Finishing the normalization of the main user.* ***:param*** *user\_data: An np array of the user's data.* ***:return****: None  
 """* user\_data = user\_data.reshape(int(user\_data.shape[0] / 6), 6)  
 user\_data = \_normalize\_keys\_values(user\_data)  
 \_append\_to\_sw(user\_data)  
 \_append\_to\_labels(1, user\_data)  
  
  
def \_normalize\_other\_users():  
 *"""  
 Extracts the folder paths for the other's data* ***:return****: None  
 """* users\_paths = \_get\_others\_data\_file\_path()  
 for path in users\_paths:  
 \_norm\_other\_users\_data(path)  
  
  
def \_save\_normalized\_data(username: str):  
 *"""  
 Saves the end product of the normalization process.* ***:param*** *username: A string of the user name* ***:return****: Saves as an h5 file, with the username's name.  
 """* global train\_labels  
 global sliding\_window\_data  
  
 train\_labels = train\_labels.reshape(int(train\_labels.shape[0]), 1)  
 sliding\_window\_data = sliding\_window\_data.reshape(int(train\_labels.shape[0]), WINDOW\_SIZE, 6)  
 sliding\_window\_data, train\_labels = shuffle(sliding\_window\_data, train\_labels)  
 directory = f"train\_data/{username}"  
 os.makedirs(directory, exist\_ok=True)  
 with h5py.File(f'{directory}/data.h5', 'w') as hdf:  
 hdf.create\_dataset('train\_data', data=sliding\_window\_data)  
 hdf.create\_dataset('train\_labels', data=train\_labels)  
  
  
def \_norm\_other\_users\_data(folder\_path: str):  
 *"""  
 Extracts the file paths for the other's data* ***:param*** *folder\_path: A path for the folder of the other users DB* ***:return****: None  
 """* for filename in os.listdir(folder\_path):  
 print(filename)  
 \_norm\_others\_values(filename, folder\_path)  
  
  
def \_get\_file\_data\_type(filepath: str) -> np.ndarray:  
 *"""  
 Estimates the type of the txt file data* ***:param*** *filepath: The txt file path to interpret.* ***:return****: The file data as a value  
 """* with open(filepath, 'r') as file:  
 file\_data = file.readlines()  
 for line in file\_data:  
 line = \_remove\_line\_brackets(line)  
 file\_data = ast.literal\_eval(line)  
 return file\_data  
  
  
def \_remove\_line\_brackets(file\_line: str) -> str:  
 *"""  
 Removes brackets* ***:param*** *file\_line: A line from a file's data* ***:return****: The file data without brackets  
 """* file\_line.replace('[', "")  
 file\_line.replace(']', "")  
 return file\_line  
  
  
def \_norm\_others\_values(filename: str, folder\_path: str):  
 *"""  
 Normalize other users' values.* ***:param*** *filename: The file name to norm* ***:param*** *folder\_path: The folder where the file is* ***:return****: None  
 """* filepath = os.path.join(folder\_path, filename)  
 new\_train\_data = np.array(\_get\_file\_data\_type(filepath))  
 new\_train\_data = \_normalize\_keys\_values(new\_train\_data)  
 new\_train\_data = \_cut\_data\_80\_values(new\_train\_data)  
 \_append\_to\_sw(new\_train\_data)  
 \_append\_to\_labels(0, new\_train\_data)  
  
  
def \_append\_to\_labels(user\_type: int, data: np.ndarray):  
 *"""  
 Appends data to the labels foe the data* ***:param*** *user\_type: 1 for main 0 for others* ***:param*** *data: the data to relate the labels to.* ***:return****: None  
 """* global train\_labels  
  
 label = np.empty(int(data.shape[0] - WINDOW\_SIZE))  
 label.fill(user\_type)  
 train\_labels = np.append(train\_labels, label)  
  
  
def \_append\_to\_sw(data: np.ndarray):  
 *"""  
 Append the data to the sliding windows.* ***:param*** *data: The data to append* ***:return****: None  
 """* global sliding\_window\_data  
  
 for line in range(data.shape[0] - WINDOW\_SIZE):  
 sliding\_window\_data = np.append(sliding\_window\_data, data[line:line + WINDOW\_SIZE])  
  
  
def \_cut\_data\_80\_values(data: np.ndarray) -> np.ndarray:  
 *"""  
 Cuts the data and takes 80 lines of it* ***:param*** *data: The data to cut* ***:return****: 80 lines of the data  
 """* cut\_value = random.randint(0, data.shape[0] - 81)  
 return data[cut\_value:cut\_value + 80]  
  
  
def \_normalize\_keys\_values(data: np.ndarray) -> np.ndarray:  
 *"""  
 Normalize the vk of the keys.* ***:param*** *data: The data to norm* ***:return****: The normalized data  
 """* for j in range(0, len(data)):  
 data[j][0] = data[j][0] / 254  
 data[j][1] = data[j][1] / 254  
 return data  
  
  
def \_get\_others\_data\_file\_path() -> list:  
 *"""  
 Gets the file path for the DB data* ***:return****: a list of paths.  
 """* result = []  
 for filename in os.listdir(PATH):  
 if filename.find("user") != -1:  
 filepath = os.path.join(PATH, filename)  
 result.append(filepath)  
 return result

### train\_model.py

import math  
import h5py  
import numpy as np  
import tensorflow as tf  
from tensorflow.keras import layers  
from tensorflow.keras.callbacks import LearningRateScheduler  
from numpy import ndarray  
from tensorflow import keras  
import os  
  
BATCH\_SIZE = 32  
EPOCHS = 150  
INPUT\_SHAPE = (30, 6)  
  
  
def \_get\_data\_from\_files(username: str):  
 *"""  
 Extracts the data from the file* ***:param*** *username: A string of the user name* ***:return****: Tuple of the data and labels  
 """* with h5py.File(f'train\_data/{username}/data.h5', 'r') as hdf:  
 data = hdf.get('train\_data')  
 train\_data = np.array(data)  
 data = hdf.get('train\_labels')  
 train\_labels = np.array(data)  
 return train\_data, train\_labels  
  
  
def \_sort\_data\_by\_label(is\_user: bool, train\_data: ndarray, train\_labels: ndarray) -> tuple:  
 *"""  
 Sorts the data by the labels and the user type.* ***:param*** *is\_user: 1 if user 0 if not* ***:param*** *train\_data: The data to sort* ***:param*** *train\_labels: The data labels* ***:return****: Tuple of the sorted data and labels  
 """* num = int(is\_user)  
 val\_data = np.array([])  
 for i in range(1000):  
 if train\_labels[i] == num:  
 val\_data = np.append(val\_data, train\_data[i])  
  
 val\_label = np.empty(int(val\_data.shape[0] / (INPUT\_SHAPE[0] \* 6)))  
 val\_label.fill(num)  
 return val\_label, val\_data  
  
  
def \_get\_data(username: str) -> tuple:  
 *"""  
 Gets the sorted data from the files* ***:param*** *username: A string of the user name* ***:return****: a list of val1\_labels, val1\_data, val0\_labels, val0\_data, train\_data, train\_labels  
 """* train\_data, train\_labels = \_get\_data\_from\_files(username)  
 val1\_labels, val1\_data = \_sort\_data\_by\_label(True, train\_data, train\_labels)  
 val0\_labels, val0\_data = \_sort\_data\_by\_label(False, train\_data, train\_labels)  
 val1\_data = val1\_data.reshape(int(val1\_labels.shape[0]), INPUT\_SHAPE[0], 6)  
 val0\_data = val0\_data.reshape(int(val0\_labels.shape[0]), INPUT\_SHAPE[0], 6)  
 return val1\_labels, val1\_data, val0\_labels, val0\_data, train\_data, train\_labels  
  
  
def \_train\_model(train\_data: ndarray, train\_labels: ndarray) -> keras.models.Sequential:  
 *"""  
 Trains the model with the labels and the data* ***:param*** *train\_data: The data to train it with* ***:param*** *train\_labels: The data labels* ***:return****: The trained model  
 """* model = \_create\_model()  
 print(model.summary())  
 train\_data = train\_data[1000:]  
 train\_labels = train\_labels[1000:]  
 lrate = LearningRateScheduler(\_step\_decay)  
 callbacks\_list = [lrate]  
 with tf.device('/device:GPU:0'):  
 model.fit(train\_data, train\_labels, batch\_size=BATCH\_SIZE, validation\_split=0.1, callbacks=callbacks\_list,  
 epochs=EPOCHS, shuffle=True)  
 return model  
  
  
def \_step\_decay(epoch) -> float:  
 *"""  
 Gets the decay rate of the model training.* ***:param*** *epoch: A complete iteration of the entire training dataset* ***:return****: The rate  
 """* initial\_lrate = 0.0001  
 drop = 0.1  
 epochs\_drop = 100.0  
 lrate = initial\_lrate \* math.pow(drop, math.floor((1 + epoch) / epochs\_drop))  
 return lrate  
  
  
def \_compile\_model(model: keras.models.Sequential) -> keras.models.Sequential:  
 *"""  
 Compiles the model* ***:param*** *model: The model to compile* ***:return****: The compiled model  
 """* loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)  
 optimizer = keras.optimizers.Adam(learning\_rate=0.0)  
 metrics = ["accuracy"]  
 model.compile(loss=loss, optimizer=optimizer, metrics=metrics)  
 return model  
  
  
def \_create\_model() -> keras.models.Sequential:  
 *"""  
 Creates the model* ***:return****: The new model  
 """* model = keras.models.Sequential()  
 model.add(layers.Conv1D(32, 2, activation='relu', input\_shape=INPUT\_SHAPE))  
 model.add(tf.keras.layers.LSTM(32, return\_sequences=True))  
 model.add(layers.Dropout(0.5))  
 model.add(tf.keras.layers.LSTM(32, return\_sequences=True))  
 model.add(layers.Dropout(0.5))  
 model.add(layers.Flatten())  
 model.add(layers.Dense(2, activation='softmax'))  
 return \_compile\_model(model)  
  
  
def \_evaluate\_model(model: keras.models.Sequential, val\_data: ndarray, val\_labels: ndarray):  
 *"""  
 Evaluates the model* ***:param*** *model: The model to evaluate* ***:param*** *val\_data: The data to evaluate the model with* ***:param*** *val\_labels: The data labels* ***:return****: None  
 """* model.evaluate(val\_data, val\_labels, batch\_size=BATCH\_SIZE, verbose=1)  
 probability\_model = keras.models.Sequential([model, keras.layers.Softmax()])  
 predictions = probability\_model(val\_data)  
 pre0 = predictions[0]  
 print(pre0)  
 label0 = np.argmax(pre0)  
 print(label0)  
  
  
def init\_model(username: str):  
 *"""  
 Creates the model ans initiates the learn process.* ***:param*** *username: A string of the user name* ***:return****: Saves the model as h5 file  
 """* val1\_labels, val1\_data, val0\_labels, val0\_data, train\_data, train\_labels = \_get\_data(username)  
 model = \_train\_model(train\_data, train\_labels)  
 # evaluate\_model(model, val1\_data, val1\_labels)  
 # evaluate\_model(model, val0\_data, val0\_labels)  
 directory = f"saved\_models/{username}"  
 os.makedirs(directory, exist\_ok=True)  
 model.save(f"{directory}/model.h5")  
 print("DONE!")

### user\_interface.py

import os  
import tkinter as tk  
from tkinter import font  
import create\_train\_files  
import train\_model  
  
  
def init() -> tuple:  
 *"""  
 The main of the UI.* ***:return****: Tuple of the window, output\_label, output\_box2, switch\_var, username and the should\_record switch  
  
 """* username = \_get\_username()  
 should\_record = \_open\_choice\_window("Should I record?")  
 \_make\_new\_model\_available(username, should\_record)  
 window = \_create\_window()  
 output\_frame = \_create\_frame\_for\_output\_boxes(window)  
 output\_label = \_create\_a\_status\_label(output\_frame)  
 \_create\_title(output\_frame, "accuracy:")  
 output\_box2 = \_create\_an\_output\_box(output\_frame)  
 switch\_var = \_create\_switch\_button(output\_frame, "lock")  
 return window, output\_label, output\_box2, switch\_var, username, should\_record  
  
  
def \_make\_new\_model\_available(username: str, should\_record: str):  
 *"""  
 if needed creates new model and train files.* ***:param*** *username: A string of the user name* ***:param*** *should\_record: A string of the user's choice* ***:return****: None  
 """* if not os.path.exists(f"saved\_models/{username}") and should\_record == "No":  
 create\_train\_files.create\_train\_file(username)  
 train\_model.init\_model(username)  
  
  
def \_open\_choice\_window(title: str) -> str:  
 *"""  
 Opens a window with two radio buttons (Yes/No)* ***:param*** *title: A string of the title of the window* ***:return****: A string of the users choice  
 """* def \_submit\_answer():  
 nonlocal answer  
 if var.get() == 1:  
 answer = "Yes"  
 else:  
 answer = "No"  
 window.destroy()  
  
 answer = "No"  
  
 window = tk.Tk()  
 window.title(title)  
  
 var = tk.IntVar()  
  
 radio\_yes = tk.Radiobutton(window, text="Yes", variable=var, value=1)  
 radio\_yes.pack()  
  
 radio\_no = tk.Radiobutton(window, text="No", variable=var, value=2)  
 radio\_no.pack()  
  
 button\_submit = tk.Button(window, text="Submit", command=\_submit\_answer)  
 button\_submit.pack()  
  
 window.mainloop()  
  
 return answer  
  
  
def \_get\_username() -> str:  
 *"""  
 Opens a window for the input of the username* ***:return****: A string of the user name  
 """* def \_submit\_username():  
 nonlocal username  
 username = entry.get()  
 window.destroy()  
  
 username = ""  
  
 window = tk.Tk()  
 window.title("Enter Username")  
  
 label = tk.Label(window, text="Username:")  
 label.pack()  
  
 entry = tk.Entry(window)  
 entry.pack()  
  
 button = tk.Button(window, text="Submit", command=\_submit\_username)  
 button.pack()  
  
 window.mainloop()  
  
 return username  
  
  
def \_close\_window(window: tk.Tk):  
 *"""  
 Closes the given window.* ***:param*** *window: The window to close* ***:return****: None  
 """* # Function to close the window  
 window.destroy()  
  
  
def \_create\_window() -> tk.Tk:  
 *"""  
 Creates the main window* ***:return****: The new window  
 """* # Create the main window  
 window = tk.Tk()  
 window.title("Input and Output Window")  
 # Configure the window to open in full screen  
 window.attributes('-topmost', True) # Set the window to be topmost  
 # window.wm\_attributes('-fullscreen', True)  
  
 # Bind the Escape key to close the window  
 window.bind('<Escape>', lambda event: \_close\_window(window))  
  
 return window  
  
  
def \_create\_title(frame: tk.Frame, text: str):  
 *"""  
 Creates a title on the window* ***:param*** *frame: The frame of the title* ***:param*** *text: The text of the title* ***:return****: None  
 """* # Add a title for the input box  
 input\_title\_label = tk.Label(frame, text=text, font=font.Font(size=20, weight='bold'))  
 input\_title\_label.pack(side=tk.TOP)  
  
  
def \_create\_an\_output\_box(output\_frame: tk.Frame) -> tk.Text:  
 *"""  
 Creates an output box* ***:param*** *output\_frame: The frame for the box* ***:return****: The new box  
 """* # Create the first output screen  
 output\_textbox\_font = font.Font(size=14) # Define font size for output text boxes  
 output\_textbox = tk.Text(output\_frame, height=5, width=40, font=output\_textbox\_font)  
 output\_textbox.pack(side=tk.TOP)  
 output\_textbox.configure(state='disabled') # Set output text box as read-only  
 return output\_textbox  
  
  
def \_create\_frame\_for\_output\_boxes(window: tk.Tk) -> tk.Frame:  
 *"""  
 Creates a frame for the output box* ***:param*** *window: The window to create it on* ***:return****: The frame  
 """* # Create a frame for the output screens  
 output\_frame = tk.Frame(window)  
 output\_frame.pack(side=tk.TOP, padx=10, pady=10, anchor='ne')  
 return output\_frame  
  
  
def update\_output\_box(output\_textbox: tk.Text, content: str):  
 *"""  
 Updates the box on the window.* ***:param*** *output\_textbox: The box to update* ***:param*** *content: The text to update it with* ***:return****: None  
 """* # Update the content of the output box  
 output\_textbox.config(state='normal')  
 output\_textbox.delete('1.0', tk.END) # Clear previous content  
 output\_textbox.insert(tk.END, str(content) + "\n")  
 output\_textbox.config(state='disabled')  
  
  
def \_create\_a\_status\_label(output\_frame: tk.Frame) -> tk.Label:  
 *"""  
 Creates the status label* ***:param*** *output\_frame: The frames for the label* ***:return****: The label  
 """* # Create a label to display the text "not sure"  
 output\_label = tk.Label(output\_frame, text="not user", font=font.Font(size=24, weight='bold'), fg="red")  
 output\_label.pack(side=tk.TOP)  
 return output\_label  
  
  
def update\_status\_label(output\_label: tk.Label, content: str, color: str):  
 *"""  
 Updates the label by the given arguments* ***:param*** *output\_label: The label to update* ***:param*** *content: What to write* ***:param*** *color: What color* ***:return****: None  
 """* # Update the content and color of the output label  
 output\_label.config(text=str(content), fg=color)  
  
  
def \_create\_switch\_button(output\_frame: tk.Frame, text: str) -> tk.IntVar:  
 *"""  
 Creates the switch button on the main window* ***:param*** *output\_frame: The frame for the switch* ***:param*** *text: The text of the switch* ***:return****: The switch  
 """* # Create a switch button below the output box  
 switch\_var = tk.IntVar()  
 switch\_button = tk.Checkbutton(output\_frame, text=text, variable=switch\_var, font=font.Font(size=20))  
 switch\_button.pack(side=tk.TOP)  
 return switch\_var

### convert\_to\_windows.py

*"""  
A dictionary that converts mac vk to windows, since the DB is window supported.  
"""*macToPCDict = {47: 190, 41: 186, 30: 221, 33: 219, 42: 220, 27: 189, 44: 191, 36: 13, 51: 8, 48: 9, 56: 16, 57: 20,  
 53: 27, 49: 32, 116: 33, 121: 34, 119: 35, 115: 36, 123: 37, 126: 38, 124: 39, 125: 40, 114: 47, 29: 48,  
 18: 49, 19: 50, 20: 51, 21: 52, 23: 53, 22: 54, 26: 55, 28: 56, 25: 57, 0: 65, 11: 66, 8: 67, 2: 68,  
 14: 69, 3: 70, 5: 71, 4: 72, 34: 73, 38: 74, 40: 75, 37: 76, 46: 77, 45: 78, 31: 79, 35: 80, 12: 81,  
 15: 82, 1: 83, 17: 84, 32: 85, 9: 86, 13: 87, 7: 88, 16: 89, 6: 90, 82: 96, 83: 97, 84: 98, 85: 99,  
 86: 100, 87: 101, 88: 102, 89: 103, 91: 104, 92: 105, 122: 112, 120: 113, 99: 114, 118: 115, 96: 116,  
 97: 117, 98: 118, 100: 119, 101: 120, 109: 121, 103: 122, 111: 123, 105: 124, 107: 125, 113: 126}

### test\_model.py

from tensorflow import keras  
import convert\_to\_windows  
import user\_interface as ui  
import tkinter as tk  
import numpy as np  
from pynput import keyboard  
from time import time  
import sys  
import threading  
import collections  
import os  
import platform  
  
q = collections.deque() # queue for the windows prediction calculation  
dwell = [] # The list of key dwell times  
startTimes = np.zeros(254) # Saves the time in witch a key was pressed  
startTyping = 0 # start time  
DownDown = [] # DownDown array  
virtualKeysID = [] # key ID array  
last\_key\_press\_time = 0 # a variable used to calculate the duration between keystrokes  
count = 0 # number of tuples inserted  
model: keras.models.Sequential  
sem = threading.Semaphore(0) # semaphore to count the number of windows in the queue  
mutex = threading.Semaphore(1) # semaphore to protect the insert and pop operations on the queue  
output\_label: tk.Label  
output\_textbox2: tk.Text  
window: tk.Tk  
switch: tk.IntVar  
username: str  
should\_record: str  
fail\_count = 0  
  
  
def \_on\_press(key):  
 *"""  
 On press function* ***:param*** *key: The key that was pressed* ***:return****: None  
 """* global last\_key\_press\_time  
 global startTyping  
 global count  
  
 current\_time = time()  
  
 \_is\_first\_typed(current\_time)  
  
 last\_key\_press\_time = current\_time  
 vk = \_get\_virtual\_key(key)  
 startTimes[vk] = current\_time  
 virtualKeysID.append(vk / 254)  
 sys.stdout.flush()  
  
  
def \_is\_first\_typed(current\_time: float):  
 *"""  
 Record the time the first key was pressed* ***:param*** *current\_time: The time of the event* ***:return****: None  
 """* global startTyping  
 if startTyping == 0:  
 startTyping = current\_time  
 if last\_key\_press\_time != 0:  
 DownDown.append(current\_time - last\_key\_press\_time)  
  
  
def \_on\_release(key) -> bool:  
 *"""  
 On release function* ***:param*** *key: The key that was release* ***:return****: False if the key is escape  
 """* global count  
  
 \_append\_times(key)  
  
 if count > 30:  
 \_append\_to\_queue(count)  
  
 count += 1  
  
 if key == keyboard.Key.esc:  
 return False  
  
  
def \_append\_times(key):  
 *"""  
 Append the event times to the corresponding lists* ***:param*** *key: The key that was triggered* ***:return****: None  
 """* current\_time = time()  
 vk = \_get\_virtual\_key(key)  
 start = startTimes[vk]  
 startTimes[vk] = 0  
 dwell.append(current\_time - start)  
  
  
def \_append\_to\_queue(num: int):  
 *"""  
 Appends to queue for the sliding window size* ***:param*** *num: The num to append* ***:return****: None  
 """* mutex.acquire()  
 q.append(num)  
 mutex.release()  
 sem.release()  
  
  
def \_get\_virtual\_key(key) -> int:  
 *"""  
 Gets the correct windows virtual key code for the key* ***:param*** *key: The key to get its vk* ***:return****: The windows vk of the key  
 """* key\_value = key.vk if hasattr(key, 'vk') else key.value.vk  
 if platform.system() == "Darwin":  
 try:  
 key\_value = convert\_to\_windows.macToPCDict[key\_value]  
 except KeyError:  
 key\_value = -1  
 return key\_value  
  
  
def \_predict\_and\_print(position\_number: int):  
 *"""  
 Calculates the vector and predicts* ***:param*** *position\_number: The position of the window* ***:return****: None  
 """* global dwell  
 global DownDown  
 global virtualKeysID  
 global model  
  
 dwell\_chunk = np.array(dwell[position\_number - 31:position\_number])  
 down\_down\_chunk = np.array(DownDown[position\_number - 31:position\_number])  
 up\_down\_chunk = down\_down\_chunk - dwell\_chunk  
  
 final\_vector = \_get\_final\_vector(down\_down\_chunk, dwell\_chunk, position\_number, up\_down\_chunk, virtualKeysID)  
  
 final\_vector = np.array(final\_vector)  
 final\_vector = final\_vector.reshape(1, 30, 6)  
 if should\_record == "No":  
 predictions = model.predict(x=final\_vector, verbose=1)  
 \_print\_predictions(predictions)  
  
  
def \_print\_predictions(predictions: list):  
 *"""  
 Prints the predictions on the screen* ***:param*** *predictions: The prediction to print* ***:return****: None  
 """* global fail\_count  
  
 for prediction in predictions:  
 print(prediction)  
 ui.update\_output\_box(output\_textbox2, prediction)  
 \_check\_if\_to\_lock(prediction)  
  
  
def \_check\_if\_to\_lock(prediction):  
 *"""  
 Checks if to lock the computer after 10 false predictions* ***:param*** *prediction: The predictions* ***:return****: None  
 """* global fail\_count  
 if prediction[1] > 0.5:  
 ui.update\_status\_label(output\_label, "user", "green")  
 fail\_count = 0  
 else:  
 ui.update\_status\_label(output\_label, "not user", "red")  
 if switch.get() == 1 and fail\_count == 15:  
 \_turn\_off()  
 fail\_count = 0  
 elif switch.get() == 1:  
 fail\_count += 1  
  
  
def \_turn\_off():  
 *"""  
 Turns off the computer* ***:return****: None  
 """* os.system("rundll32.exe user32.dll,LockWorkStation") # windows  
 os.system("osascript -e 'tell application \"System Events\" to keystroke \"q\" using {control down, command down}'")  
  
  
def \_get\_final\_vector(down\_down\_chunk: np.ndarray, dwell\_chunk: np.ndarray, position\_number: int,  
 up\_down\_chunk: np.ndarray, virtual\_keys\_id: list) -> list:  
 *"""  
 Calculates the final vector to predict with* ***:param*** *down\_down\_chunk: Keyboard timing list* ***:param*** *dwell\_chunk: Keyboard timing list* ***:param*** *position\_number: The position in the sliding windows* ***:param*** *up\_down\_chunk: Keyboard timing list* ***:param*** *virtual\_keys\_id: virtual key codes list* ***:return****: list of the final vector  
 """* final\_vector = []  
 index = position\_number - 31  
 for i in range(len(dwell\_chunk) - 1):  
 vector\_to\_append = (  
 virtual\_keys\_id[i + index], virtual\_keys\_id[i + 1 + index], dwell\_chunk[i], dwell\_chunk[i + 1],  
 down\_down\_chunk[i],  
 up\_down\_chunk[i])  
 final\_vector.append(vector\_to\_append)  
 if should\_record == "Yes":  
 \_make\_new\_user\_files(final\_vector[-1])  
 return final\_vector  
  
  
def \_make\_new\_user\_files(final\_vector: tuple):  
 *"""  
 Makes new user txt files* ***:param*** *final\_vector: the data to append* ***:return****: writes in the files  
 """* directory = f"keystrokes\_data/{username}"  
 if os.path.exists(directory):  
 new\_data = \_arrange\_new\_data(final\_vector)  
 new\_data = \_get\_data\_to\_write(directory, new\_data)  
 \_write\_in\_file(directory, str(new\_data))  
 else:  
 os.makedirs(directory, exist\_ok=True)  
 new\_data = "["  
 new\_data += str(\_arrange\_new\_data(final\_vector))  
 new\_data += "]"  
 \_write\_in\_file(directory, str(new\_data))  
  
 ui.update\_output\_box(output\_textbox2, "put")  
  
  
def \_arrange\_new\_data(new\_data: tuple) -> tuple:  
 *"""  
 Arranges new data to match the DB format.* ***:param*** *new\_data: The data to arrange* ***:return****: The arranged data  
 """* result = [int(new\_data[0] \* 254), int(new\_data[1] \* 254)]  
 for i in range(2, len(new\_data)):  
 result.append(new\_data[i])  
 return tuple(result)  
  
  
def \_get\_data\_to\_write(directory: str, final\_vector: tuple) -> str: # [(1,3,,3),()...]  
 *"""  
 If the file existed, combines the old data with the new one.* ***:param*** *directory: The file's path* ***:param*** *final\_vector: The data to append* ***:return****: The new combined data  
 """* with open(f"{directory}/data.txt", "r") as file:  
 old\_data = file.read().split("]")  
 new\_data = old\_data[0]  
 new\_data += ", "  
 new\_data += (str(final\_vector))  
 new\_data += "]"  
 return new\_data  
  
  
def \_write\_in\_file(directory: str, new\_data: str):  
 *"""  
 Writes in the given file.* ***:param*** *directory: The file to write* ***:param*** *new\_data: The mode of the method* ***:return****: None  
 """* with open(f"{directory}/data.txt", "w") as file:  
 file.write(new\_data)  
  
  
def \_predictions\_thread():  
 *"""  
 The prediction thread handling.* ***:return****: None  
 """* while True:  
 sem.acquire()  
 mutex.acquire()  
 x = q.popleft()  
 mutex.release()  
 \_predict\_and\_print(x)  
  
  
def \_get\_model():  
 *"""  
 Gets the model by the username* ***:return****: None  
 """* global model  
 if should\_record == "No":  
 if os.path.exists(f'saved\_models/{username}/model.h5'):  
 model = keras.models.load\_model(f'saved\_models/{username}/model.h5')  
 else:  
 print("No such model in the systems!")  
 quit()  
  
  
def main():  
 *"""  
 Test the system. it run the UI, and if needed creates train files and runs the train model* ***:return****: None  
 """* global output\_label, output\_textbox2, switch, username, should\_record  
 threading.Thread(target=\_predictions\_thread).start()  
 with keyboard.Listener(on\_press=\_on\_press, on\_release=\_on\_release) as listener:  
 root, output\_label, output\_textbox2, switch, username, should\_record = ui.init()  
 \_get\_model()  
 root.mainloop()  
 listener.start()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()