



**A Digital Honor Code: Detecting Academic Dishonesty
with Keystroke Dynamics**

A Project Report

submitted by

DEBNATH KUNDU

MT22026

*in partial fulfilment of the requirements
for the award of the degree of*

MASTER OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING
INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

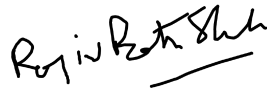
NEW DELHI- 110020

20th May 2024

THESIS CERTIFICATE

This is to certify that the thesis titled **A Digital Honor Code: Detecting Academic Dishonesty with Keystroke Dynamics**, submitted by **Debnath Kundu**, to the Indraprastha Institute of Information Technology, Delhi, for the award of the degree of **Master's of Technology**, is a bona fide record of the research work done by him under my supervision. This thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.



Advisor: Dr. Rajiv Ratn Shah
Thesis Supervisor
Associate Professor, Institute Chair
Professor
Department of CSE and HCD
IIIT Delhi, 110020



Co-Advisor: Dr. Rajesh Kumar
Thesis Supervisor
Assistant Professor
Dept. of Computer Science
Bucknell University, Lewisburg, PA,
USA

Place: New Delhi

Date: 20th May 2024

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who have supported and guided me throughout my M.Tech. thesis work. First and foremost, I would like to thank *Dr. Rajiv Ratn Shah* and *Dr. Rajesh Kumar* for not giving up on me. Their unwavering support and belief in me have been a source of great strength, and their encouragement and patience have been instrumental in completing this work. I am immensely grateful to *Dr. Debajyoti Bera*, who has taught me so much more than algorithms. His wisdom and guidance have profoundly shaped my understanding and approach to this project.

A special thanks to **Atharva Mehta**, who was deeply involved in this project alongside me. His insights and efforts have been invaluable, and working with him has been a rewarding experience. I would also like to extend my heartfelt appreciation to the entire **MIDAS Labs** team: *Mohit (S), Janak, Pijush, Mohit (G), Rishabh, Vani, Ritika, Naman, Avinash, Kritarth, Raj, Aman, and Yaman*. Their camaraderie and assistance have been vital in the journey of this thesis. Each of them has contributed uniquely, and their support has meant the world to me.

Lastly, I am deeply thankful to my friends **Vindhya, Manvendra, and Shambhavi**. Their constant support and encouragement have been a source of strength and motivation throughout this journey. Their friendship has made this challenging process more manageable and enjoyable.

Thank you all for being a part of this incredible experience.

Abstract

As educational assessments migrate to digital platforms, ensuring academic integrity becomes crucial. Traditional plagiarism detection systems struggle to catch instances of intelligent cheating, especially when students use advanced generative AI tools to produce responses. This thesis proposes a novel approach based on keystroke dynamics to distinguish between *bonafide* and *assisted* writing in academic contexts. A dataset was created to record the keystroke patterns of individuals during writing tasks, with and without generative AI assistance. Machine learning and pattern recognition techniques were employed at various levels of detail to determine if unusual typing behaviors can indicate academic dishonesty. The detectors were evaluated under various condition-specific and condition-agnostic scenarios, and the findings reveal noticeable differences in keystroke dynamics recorded for genuine and assisted writing. The thesis outcomes contribute to understanding the interaction between users and generative AI, with implications for enhancing the reliability of digital educational platforms.

Keywords: Keystrokes, cognitive load, plagiarism detection, cheating, education, Pattern Recognition, LGBM Classifier, Deep Learning

Contents

1	Introduction	6
2	Related Works	10
2.1	Cheating and Plagiarism Detection	10
2.2	Emergence of Keystroke Dynamics	10
2.3	Behavior Modeling using Keystrokes	11
2.4	The Current Study	11
3	Background	12
3.1	Keystroke Dynamics	12
3.1.1	Free Writing vs. Fixed Writing in Keystroke Dynamics	12
3.2	Sliding Window Algorithm	14
3.3	Machine Learning methods	15
3.3.1	Light Gradient Boosting (LightGBM)	15
3.3.2	Gradient Boosting Framework	15
3.3.3	Objective Function	16
3.3.4	Tree Growth Algorithm	16
3.3.5	Advantages of LightGBM	17
3.4	Deep Learning methods	17
3.4.1	Long Short-Term Memory (LSTM)	17
3.4.2	Binary Cross Entropy (BCE) Loss Function	20
3.4.3	Cosine Similarity	21
4	Problem Statement	24
4.1	Problem proposed	24
4.1.1	Detailed Explanation	24
5	Methods	26
5.1	Data Collection	26
5.2	Additional Datasets	27
5.3	Feature Extraction	27
5.4	Proposed Framework	29
5.5	Detection Methods	32

5.6	Evaluation Scenarios	34
5.7	Evaluation Metrics	34
6	Results	36
6.0.1	Keyboard-Specific and Keyboard-Agnostic Scenarios	36
6.0.2	Context-Specific and Context-Agnostic Scenarios	36
6.0.3	User-Specific and User-Agnostic Models	37
6.0.4	Dataset-Specific and Dataset-Agnostic Scenarios	37
6.1	Ablation study	38
6.1.1	LGBM Classifier	38
6.1.2	TypeNet	38
6.2	Discussions	41
7	Conclusion and Future Scope	44
7.1	Conclusion and Future Scope	44

List of Figures

1.1	Existing plagiarism detection system framework to "pinpoint word-for-word similarities" only [1, 2].	7
1.2	A Cognitive Model of Writing Process [3]: <i>Proposer</i> includes idea generation and task preparation, identifiable by initial pauses and at sentence boundaries. <i>Translator</i> involves fluency in converting ideas into language, measured by the length of uninterrupted text production sequences, also known as "bursts." <i>Transcriber</i> relates to orthographic proficiency and motor skills, observed through pauses within words and immediate corrections of spelling errors or typos. <i>Evaluator</i> relates to editing and reviewing, identified by jumps to different text parts for extensive edits and replacements.	8
3.1	Keystroke dynamics features include dwell or hold time, which measures the duration a key is pressed, and digraph latency, defined as the time interval between the press or release events of two consecutive keys. [4]	13
3.2	Leaf-wise tree growth strategy in Light Gradient Boosting [5]	17
5.1	210 long Feature Vector for each window	29
5.2	KDE Plots of some frequent words showing a clear difference in the probability function between the two forms of writing	29
5.3	Data preprocessing: we standardize the sequence lengths to a predetermined length M by padding shorter sequences and clipping longer ones, adjusting M during hyperparameter tuning. The value of M ranged between 25-500 with batch size between 32-512 in powers of 2. Sequences with excessive use of the Shift key (over 20%) and those significantly shorter than M (less than 50%) were excluded. Each keystroke sequence includes three attributes: timestamps, keycodes, and key actions (KeyDown, KeyUp), normalized as follows. Timestamps were scaled to a $[0, 1]$ range using min-max normalization. Keycodes are normalized by dividing each by 255. Key actions are binary encoded with KeyDown as 0 and KeyUp as 1.	30
5.4	Proposed Detection Methods: data-driven deep learning model called TypeNet and feature-driven ML classifiers	31
5.5	Sliding Window: across users with varying window size and slide, helps in extracting features from each windows	32

5.6	System architecture: we modified the TypeNet architecture as two LSTM layers, each with a 128-dimensional output, interspersed with batch normalization layers and tanh activation functions. (2) A fully connected layer follows each LSTM to project the embeddings into a consistent 128-dimensional space. (3) Dropout layers are integrated after each LSTM layer to prevent overfitting, with a dropout rate of 0.5 and a recurrent dropout of 0.2. (4) Variable sequence length M and batch size during experiments, exploring ranges from 25 to 500 for M and 32 to 512 for batch size. We experimented with various training durations, noting that performance typically plateaued between 50 to 100 epochs. The Adam optimizer was utilized with a learning rate tailored from 0.0001 to 0.005, and L_2 regularization was included to control overfitting.	33
6.1	LGBM Classifier: A 70-30 train-test split is used for User-Specific training. However, according to Hayes’s cognitive writing model, the final revisions differ from the initial drafts. Therefore, shuffling these segments and retraining for user-specific evaluations helps improve accuracy.	40
6.2	TypeNet: In data-agnostic scenarios, increased data variance in a deep learning model improves performance as the boundary separation between training and testing sets shifts. Models trained on a single dataset do not generalize well to other datasets. However, training on a combination of two datasets enhances accuracy, demonstrating that greater data variance helps the model learn more effectively.	40
6.3	Same window size: PCA plots of the feature set indicate limited separability in the proposed dataset when using 1000 keystrokes with a 300-slide window size.	41
6.4	Varying window size: PCA plots with different window sizes and slides—1000 keystrokes with a 300-slide for Buffalo, 500 keystrokes with a 100-slide for the proposed dataset, and 2000 keystrokes with a 500-slide for the SBU dataset—show good separability. This implies that using different window sizes for various tasks and data collection environments is beneficial.	42

Chapter 1

Introduction

Cheating in online examinations has significantly increased due to rapid advancements in generative AI technology and a growing reliance on virtual educational environments. To maintain the integrity of online exams, testing agencies have implemented several online proctoring measures. These measures include prohibiting students from changing tabs, disabling the copy-paste function, and preventing the minimization of the exam window. These restrictions aim to simulate the controlled environment of in-person exams and ensure a fair testing experience for all participants.

However, these measures do not address all forms of academic dishonesty. Traditional plagiarism detection tools like Turnitin and Urkund evaluate submissions by comparing them to existing sources. While these tools are effective at identifying direct plagiarism, they often fail to detect more sophisticated cheating methods, such as extensive paraphrasing or receiving external help. Despite their limitations, these tools are widely used by academic institutions, publishers, and conference management systems. Figure 1.1 shows a framework diagram for the existing systems which processes a primary input, typically a query or suspicious document (D_q), and an optional reference collection like the web (D). If available, it outputs identified suspicious fragments and their plagiarism sources (S_q). However, they are not effective against more sophisticated forms of cheating, such as text manipulation, extensive paraphrasing, or using assistance from another person [2].

The emergence of advanced language models like *ChatGPT* and *Gemini* has further complicated the landscape. These AI systems can generate high-quality, human-like essays within seconds, presenting new challenges for existing plagiarism detection mechanisms. The ability of these tools to produce tailored, undetectable responses makes them particularly problematic for traditional plagiarism detection software, which relies on direct text comparison.

Cheating in academia also results in idea theft and intellectual property violations. Numerous studies have examined plagiarism in academic settings, detailing various forms and potential detection methods [2, 6–8]. For example, a recent book provides an in-depth linguistic analysis of plagiarism in academic writing [9]. Understanding the specific context is crucial for effectively detecting cheating. In introductory programming classes, cheating with ChatGPT can be identified by comparing student code submissions to those generated by

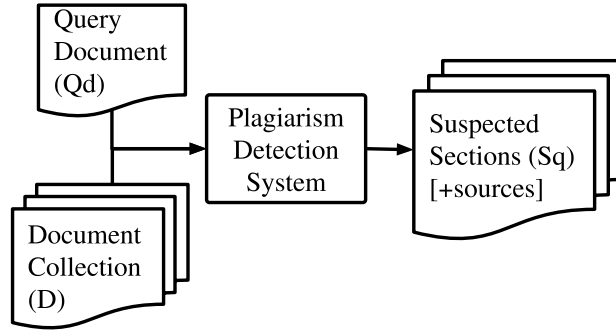


Figure 1.1: Existing plagiarism detection system framework to ”pinpoint word-for-word similarities” only [1, 2].

ChatGPT. Key differences include conciseness, reduced variable duplication, simpler control flow, and a preference for ternary operators in ChatGPT-generated code [10].

To summarize, there is a pressing need to develop more robust plagiarism detection technologies to keep pace with the evolving capabilities of AI-assisted academic dishonesty.

Limited studies connect linguistic patterns of plagiarism with behavioral characteristics. One such characteristic, known as keystroke dynamics, involves analyzing the unique typing rhythms of individuals on devices like computer keyboards, mobile phones, or touch screens. This interaction creates a digital footprint that reflects cognitive qualities specific to each person and task, making it a promising method for personal and behavioral identification. This modality offers several benefits: it is unique, cost-effective, non-intrusive during data collection, prevents replication, and allows for continuous monitoring and authentication [11].

A study by the *Educational Testing Service (ETS)*, the largest private organization for educational testing and assessment, investigates the stability and predictive power of keystroke log features for assessing writing quality. It finds that although these features vary in stability and correlation with human ratings across genres, the most stable features effectively predict essay scores. Furthermore, models using these features show good generalizability across prompts, especially within the same genre [12].

A major concern with online examinations and distance learning is the risk of user impersonation, where someone else falsely represents the examinee. This practice, known as *contract cheating* [13], can be mitigated using keystroke dynamics to uniquely identify students. When students register in the system, their typing data is automatically collected to create templates. These templates are then used to continuously verify the user’s identity during the examination [14, 15].

We investigate cheating in closed-book online exams, where using external sources constitutes academic dishonesty. When students rely solely on their own reasoning to answer questions, we term this as **bonafide** writing. Conversely, if a student incorporates content from external sources, whether from another person or online, we label this as **assisted** writing. Our approach proposes an extension of keystroke dynamics to distinguish between

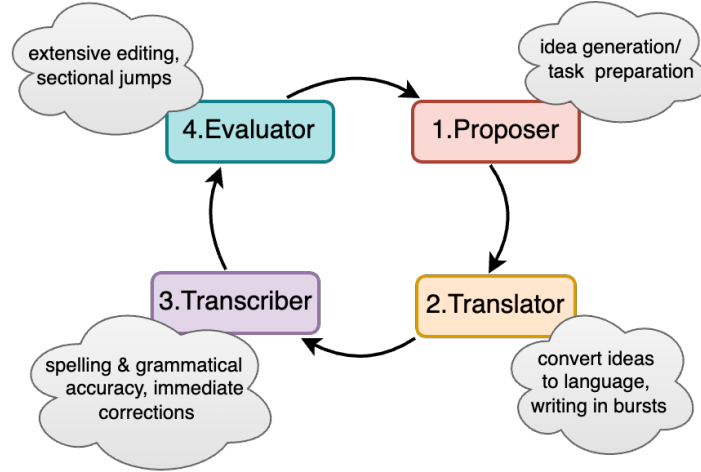


Figure 1.2: **A Cognitive Model of Writing Process** [3]: *Proposer* includes idea generation and task preparation, identifiable by initial pauses and at sentence boundaries. *Translator* involves fluency in converting ideas into language, measured by the length of uninterrupted text production sequences, also known as "bursts." *Transcriber* relates to orthographic proficiency and motor skills, observed through pauses within words and immediate corrections of spelling errors or typos. *Evaluator* relates to editing and reviewing, identified by jumps to different text parts for extensive edits and replacements.

genuine writing and writing influenced by external assistance, including AI. This premise is based on the assumption that the *cognitive effort* required for genuine writing manifests uniquely in typing patterns, distinct from patterns observed when students copy, transcribe, or utilize AI assistance.

Previous research on writing cognition has identified various sub-processes [16]. These processes include task analysis, text planning, idea generation, translating ideas into natural language, typing, revising, copy editing, and reviewing. Figure 1.2 illustrates a simplified version of *Hayes' cognitive writing model* [17], which categorizes writing into four main sub-processes: *proposer*, *translator*, *transcriber*, and *evaluator*. Building on earlier studies concerning writers' time management [13, 17], we hypothesize that genuine writing tasks involve more planning and revising, and less direct copying compared to tasks with writing assistance.

Considering the distinctive characteristics of various writing processes, such as longer breaks during planning, fewer breaks but more keystrokes during translation, and breaks accompanied by word removal during revision, our hypotheses were structured as follows:

- If the *type of task* is the primary factor affecting typing patterns, then we should observe clear differences between genuine and assisted tasks across all users, not just specific individuals.
- If typing patterns are influenced solely by *text complexity and familiarity*, then we

should not detect significant differences between genuine and assisted tasks among users who received similar questions and texts.

- If both the *type of task* and *text complexity/familiarity* impact typing patterns, then variations between the two tasks should be evident regardless of individual user differences. This study also incorporates an analysis concerning keyboard usage.

To investigate variations in typing patterns between two types of writing tasks, we conducted an experiment where participants completed two types of exams: one involved writing original answers without assistance (bonafide writing), and the other allowed the use of the Internet for searching, reading, or transcribing text (assisted writing). Our contributions to the literature include:

- *User-specific comparisons*: Comparing the same users performing different writing tasks.
- *Content-specific comparisons*: Analyzing the same text across different writing tasks.
- *User and Content agnostic comparisons*: Evaluating variations with potential user and content differences.
- *Keyboard agnostic comparisons*: Examining typing patterns of the same users with different keyboards.

If distinct differences in keystroke patterns between bonafide and assisted writing exist, a cluster analysis should reveal two separate groups, each representing a distinct typing activity.

Chapter 2

Related Works

2.1 Cheating and Plagiarism Detection

Despite advancements in detection technologies such as locked PDFs and plagiarism checkers, cheating on exams remains a persistent global problem. Students continuously find ways to circumvent these systems, such as altering texts or employing ghostwriters [18]. In response, researchers have turned to machine learning techniques like autoencoders and Long Short-Term Memory (LSTM) networks to analyze patterns in student assessment data and detect outliers, which may indicate cheating. Methods like combining LSTMs with Kernel Density Estimation (KDE) have proven effective for outlier detection in post-exam analyses, utilizing continuous assessment scores to identify irregular final exam results [19]. Furthermore, LSTMs have been utilized to analyze broader datasets, including students' academic performance, attendance, and behavior, to predict academic outcomes and pinpoint at-risk students [20]. Additional factors such as cognitive levels, seating arrangements, and similarities in exam papers are also considered to assess the likelihood of cheating [21]. More sophisticated techniques involve video surveillance to monitor suspicious physical behaviors [22]. As a preemptive measure, IP address monitoring has been implemented to randomize exam questions when anomalies are detected, further safeguarding the integrity of assessments [23].

2.2 Emergence of Keystroke Dynamics

Keystroke dynamics, extensively studied for security and authentication purposes, have gained increasing importance with the widespread use of computer systems reliant on keyboard input. The unique neuro-physiological aspects underlying keystroke patterns, akin to those in handwritten signatures, serve various purposes. This covert and unobtrusive technique not only enhances continuous user authentication for computer access [24–28] but also bolsters password security by analyzing latency times indicative of a user's typing rhythm to thwart impostor access [29, 30]. Moreover, keystroke biometrics are applied for content de-anonymization to combat fake news during the COVID-19 pandemic [31], distinguish

genuine from deceptive reviews [32], and identify fake social media profiles [33, 34]. Beyond security domains, this technique can ascertain a user’s educational level [35] and emotional state [36], detect Parkinson’s Disease [37], assess stress [38], and monitor mental fatigue to ensure safer working conditions for professionals such as drivers, pilots, surgeons, and first responders [39].

2.3 Behavior Modeling using Keystrokes

Keystroke patterns, originating from standard input devices, inadvertently reveal significant insights into an author’s characteristics, including age, gender [40–42], and handedness [43], even in live chat sessions [44]. These patterns, combined with mouse data, prove invaluable in forensic investigations, providing admissible evidence that streamlines the investigation of unauthorized digital activities, thereby reducing time and costs [45]. Moreover, research suggests that keystroke analysis contributes to lie detection by assessing response consistency with an assumed identity [46]. Further studies delve into how keystrokes reflect the cognitive demands of tasks performed by an unseen typist [47, 48]. These investigations reveal that keystroke dynamics are sensitive to variations in cognitive load and task type, profoundly impacting typing output and offering insights into a typist’s cognitive effort and efficiency [49]. These variations are further elucidated in the literature as free and fixed (transcription) writing [50], with free writing typically characterized by shorter bursts of typing, longer pauses, slower typing speeds, and more revisions than transcription tasks. This distinctive behavior facilitates the classification of task types based on burst patterns and writing speeds [51].

2.4 The Current Study

While many studies in this field compare the behavior of genuine students with that of potential impersonators during exam sessions, our research takes a different approach. We focus on distinguishing between authentic writing and writing assisted by AI, all produced by the same author. Additionally, existing research on distinguishing between freely composed and fixed texts has not been tested in real-life exam scenarios. Previous studies have typically centered on uniform, less challenging writing tasks. In contrast, our study assesses the impact of varying cognitive loads by presenting exam questions that demand different levels of cognitive effort to answer. We also introduce a new class of models tailored to detect cheating across different scenarios. These models incorporate both keystroke dynamics and stylometry to capture the intricacies of a typist’s language production. Furthermore, our work sheds light on the feasibility of analyzing typing behavior to generalize findings across unfamiliar typists in entirely different settings, beyond those encountered during model training.

Chapter 3

Background

3.1 Keystroke Dynamics

Keystroke dynamics is a biometric authentication method that analyzes the unique patterns of typing behavior exhibited by individuals when they interact with a keyboard. It involves measuring various characteristics of keystroke inputs, which can include:

1. **Dwell Time:** The duration a key is held down before being released.
2. **Flight Time:** The time interval between releasing one key and pressing the next key.
3. **Typing Speed:** The overall speed at which an individual types.
4. **Error Rate:** The frequency and types of typing errors made.
5. **Rhythm and Sequence Patterns:** The unique patterns in the sequence and timing of keystrokes, including habitual key combinations and typing rhythm.

Keystroke dynamics captures these and other related metrics to create a unique profile for each user. This profile can then be used for continuous or step-wise authentication. Unlike traditional passwords or PINs, keystroke dynamics relies on behavioral characteristics that are difficult to replicate, providing an additional layer of security. This method can be implemented in various applications, including secure access to computer systems, mobile devices, and online services.

By analyzing these patterns, systems can detect anomalies that might indicate unauthorized access, offering enhanced security through continuous monitoring of user behavior.

3.1.1 Free Writing vs. Fixed Writing in Keystroke Dynamics

In the context of keystroke dynamics, free writing and fixed writing represent two distinct modes of typing behavior that are often analyzed to understand an individual's unique typing patterns.

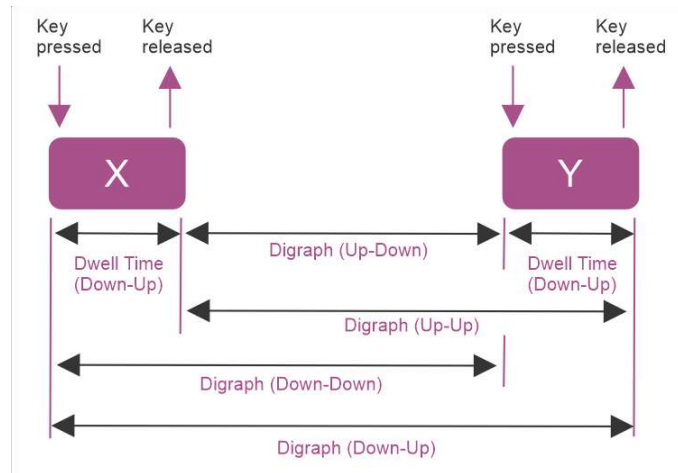


Figure 3.1: Keystroke dynamics features include dwell or hold time, which measures the duration a key is pressed, and digraph latency, defined as the time interval between the press or release events of two consecutive keys. [4]

Free Writing

Free writing refers to a mode of typing that is unrestricted and natural, akin to “*bonafide*” typing. It typically involves the following characteristics:

1. **Unrestricted Typing:** The individual types freely, without any imposed constraints or specific guidelines.
2. **Natural Flow:** The typing behavior closely resembles how a person would type in their everyday activities, such as writing emails or composing text.
3. **Shorter Typing Bursts:** The individual may type in shorter, more spontaneous bursts of text, reflecting their natural thought process.
4. **Longer Pauses:** There may be longer pauses between typing sessions as the individual thinks about what to write next.
5. **More Editing:** The individual often engages in more frequent editing, including corrections, deletions, and rewrites.

Fixed Writing

Fixed writing, on the other hand, refers to a more controlled and structured mode of typing, similar to “*assisted*” typing. It involves the following characteristics:

1. **Transcription Typing:** The individual types by transcribing or copying text from a given source, following specific instructions.
2. **Controlled Environment:** The typing behavior is more regulated, with less spontaneity and more consistency.
3. **Streamlined Writing:** The typing is more continuous and fluid, with fewer interruptions.
4. **Less Variance:** There is less variation in typing speed and rhythm, resulting in a more uniform typing pattern.

Understanding the differences between free writing and fixed writing is essential in keystroke dynamics research, as it allows for a more comprehensive analysis of an individual's typing behavior under varying conditions.

3.2 Sliding Window Algorithm

The sliding window approach is an efficient technique used in various computer science problems to process data in a subset or "window" that slides across the dataset. This method is particularly useful for problems involving subarrays or substrings. A sliding window is defined by two pointers (or indices) that represent the current window's start and end positions. The window slides from the beginning to the end of the dataset, performing computations at each step.

Example: Maximum Sum of k Consecutive Elements Consider an array of integers $A = [1, 3, 2, 5, 8, 7, 10, 12]$ and a window size $k = 3$. We want to find the maximum sum of any k consecutive elements in the array.

1. Initialize the sum of the first window:

$$\text{current_sum} = A[0] + A[1] + A[2] = 1 + 3 + 2 = 6$$

2. Slide the window from the beginning to the end of the array, updating the sum by subtracting the element that is left behind and adding the new element:

$$\text{For } i = 1 : \text{current_sum} = \text{current_sum} - A[0] + A[3] = 6 - 1 + 5 = 10$$

$$\text{For } i = 2 : \text{current_sum} = \text{current_sum} - A[1] + A[4] = 10 - 3 + 8 = 15$$

$$\text{For } i = 3 : \text{current_sum} = \text{current_sum} - A[2] + A[5] = 15 - 2 + 7 = 20$$

$$\text{For } i = 4 : \text{current_sum} = \text{current_sum} - A[3] + A[6] = 20 - 5 + 10 = 25$$

$$\text{For } i = 5 : \text{current_sum} = \text{current_sum} - A[4] + A[7] = 25 - 8 + 12 = 29$$

3. The maximum sum found during this process is 29.

Time Complexity The time complexity of the sliding window approach is $O(n)$, where n is the number of elements in the array. This is because each element is added and removed from the window exactly once. In contrast, a naive approach that recalculates the sum for each window position would have a time complexity of $O(nk)$, where k is the window size. Thus, the sliding window approach provides a significant improvement in efficiency, especially for large datasets.

3.3 Machine Learning methods

3.3.1 Light Gradient Boosting (LightGBM)

Light Gradient Boosting Machine (LightGBM) is an advanced implementation of the gradient boosting framework that uses tree-based learning algorithms. It is designed to be efficient and scalable, particularly suitable for large datasets and high-dimensional data. LightGBM constructs trees leaf-wise rather than level-wise, which can result in better accuracy and faster training times.

Gradient-based One-Side Sampling (GOSS) in the LightGBM classifier reduces training time by selectively sampling instances based on their gradients while maintaining accuracy. Additionally, LightGBM's *Exclusive Feature Bundling (EFB)* technique groups mutually exclusive features to reduce dimensionality, enabling faster training without compromising performance.

3.3.2 Gradient Boosting Framework

Gradient Boosting is an ensemble learning method that builds models sequentially, where each subsequent model aims to correct the errors of its predecessor. The main idea is to minimize a loss function by adding a new model $h_m(x)$ at each iteration m that maximizes the overall prediction accuracy. The general form of the prediction model is:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

where:

- $F_m(x)$ is the ensemble model at iteration m ,
- $F_{m-1}(x)$ is the ensemble model at iteration $m - 1$,
- η is the learning rate,
- $h_m(x)$ is the new model added at iteration m .

3.3.3 Objective Function

The objective of LightGBM is to minimize the loss function L , which measures the difference between the predicted and actual values. For a given dataset with N instances $\{(x_i, y_i)\}_{i=1}^N$, where x_i are the features and y_i are the target values, the objective function is given by:

$$L = \sum_{i=1}^N l(y_i, F_{m-1}(x_i) + \eta h_m(x_i)) + \Omega(h_m)$$

where:

- $l(y, \hat{y})$ is the loss function (e.g., mean squared error for regression, log loss for classification),
- $\Omega(h_m)$ is the regularization term to avoid overfitting.

3.3.4 Tree Growth Algorithm

LightGBM uses a leaf-wise tree growth strategy, which differs from the traditional level-wise growth. In the leaf-wise approach, LightGBM splits the leaf with the maximum loss reduction. This results in more complex trees that can lead to better accuracy. However, it also introduces the risk of overfitting, which can be controlled using the maximum depth parameter.

The split gain for a leaf-wise split is calculated as:

$$\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$$

where:

- G_L and G_R are the sums of gradients for the left and right splits,
- H_L and H_R are the sums of Hessians for the left and right splits,
- λ is the regularization parameter for the leaves,
- γ is the regularization term for the number of leaves.

Leaf-wise tree growth strategy can enhance LightGBM's performance in both accuracy and efficiency. Here are some reasons why:

1. The leaf-wise tree growth strategy reduces loss more effectively than the level-wise strategy. It does this by always choosing the leaf with the highest split gain for splitting, ensuring the most optimal split point at each iteration.
2. By limiting the number of leaves or the minimum split gain, the leaf-wise strategy can better prevent overfitting compared to the level-wise approach. It stops tree development when there is no significant improvement.

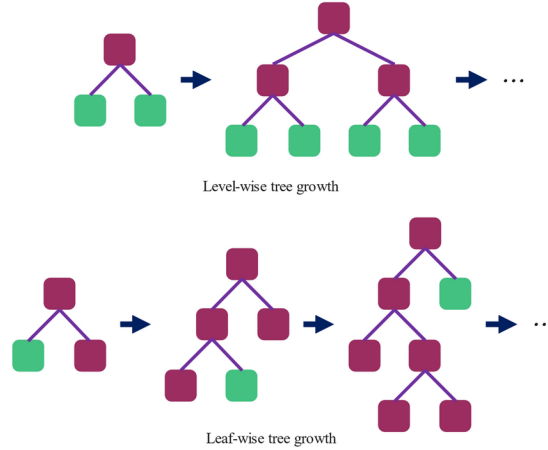


Figure 3.2: Leaf-wise tree growth strategy in Light Gradient Boosting [5]

3. The leaf-wise strategy speeds up training by requiring fewer data instances to be scanned during each iteration. This conserves memory and processing resources.

3.3.5 Advantages of LightGBM

LightGBM has several advantages:

- **Efficiency:** LightGBM is highly efficient in both training speed and memory usage.
- **Scalability:** It can handle large-scale data and high-dimensional feature spaces.
- **Accuracy:** The leaf-wise growth strategy often results in better accuracy compared to level-wise methods.
- **Support for Parallel and GPU Learning:** LightGBM supports parallel learning and can leverage GPU for faster computations.

In summary, LightGBM is a powerful machine learning algorithm that offers high efficiency and accuracy, making it a suitable choice for various prediction tasks in large datasets.

3.4 Deep Learning methods

3.4.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a special type of recurrent neural network (RNN) capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997) and have since become highly effective for a variety of sequence prediction problems.

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are designed to recognize patterns in sequences of data, such as time series or natural language. Traditional RNNs suffer from the problem of vanishing and exploding gradients, making them ineffective for capturing long-term dependencies. LSTMs address these issues by introducing a more complex memory unit.

LSTM Architecture

LSTMs have a unique structure comprising memory cells, which can maintain information in memory for long periods. Each LSTM cell contains three main components: input gate, forget gate, and output gate, which regulate the flow of information.

Input Gate

The input gate controls the extent to which new information flows into the cell state. The equations governing the input gate are:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where:

- i_t is the input gate activation vector at time step t ,
- W_i and W_C are the weight matrices for the input gate and cell state, respectively,
- h_{t-1} is the hidden state vector from the previous time step,
- x_t is the input vector at time step t ,
- b_i and b_C are the bias vectors for the input gate and cell state, respectively,
- σ is the sigmoid activation function,
- $\tanh$ is the hyperbolic tangent activation function.

Forget Gate

The forget gate determines how much of the previous cell state to retain. The equations are:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where:

- f_t is the forget gate activation vector at time step t ,
- W_f is the weight matrix for the forget gate,
- b_f is the bias vector for the forget gate.

Cell State

The cell state is updated based on the input and forget gates:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

where:

- C_t is the cell state vector at time step t ,
- C_{t-1} is the cell state vector from the previous time step,
- $\odot$ denotes element-wise multiplication.

Output Gate

The output gate controls the output of the LSTM cell. The equations are:

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \odot \tanh(C_t) \end{aligned}$$

where:

- o_t is the output gate activation vector at time step t ,
- W_o is the weight matrix for the output gate,
- b_o is the bias vector for the output gate,
- h_t is the hidden state vector at time step t .

Advantages of LSTM

LSTMs have several key advantages:

- **Long-Term Dependency Learning:** LSTMs are capable of capturing long-term dependencies in sequence data, making them suitable for tasks like language modeling and time series prediction.
- **Robustness to Vanishing/Exploding Gradients:** The gated structure of LSTMs helps mitigate the vanishing and exploding gradient problems commonly seen in traditional RNNs.
- **Flexibility:** LSTMs can be used in various architectures, including many-to-one, many-to-many, one-to-many, and one-to-one configurations, providing flexibility for different types of sequence tasks.

In summary, LSTM networks are a powerful tool for handling sequential data, offering robustness and flexibility for a wide range of applications.

3.4.2 Binary Cross Entropy (BCE) Loss Function

The Binary Cross Entropy (BCE) loss function is widely used in binary classification tasks. It measures the performance of a classification model whose output is a probability value between 0 and 1. BCE loss is also known as log loss or logistic loss.

Definition

Binary Cross Entropy loss is defined as the negative average of the log of predicted probabilities for each instance. For a binary classification problem, the BCE loss for a single instance is given by:

$$\text{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

where:

- y is the true label (0 or 1),
- $\hat{y}$ is the predicted probability that the instance belongs to the positive class (1).

Formula for BCE Loss

For a dataset with N instances, the overall BCE loss is the average BCE loss across all instances:

$$\text{BCE Loss} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

where:

- N is the number of instances in the dataset,
- y_i is the true label for the i -th instance,
- $\hat{y}_i$ is the predicted probability for the i -th instance.

Interpretation

The BCE loss function penalizes the model more when it makes confident but incorrect predictions. This is because the logarithmic function grows rapidly for probabilities close to 0 or 1, leading to higher loss values for incorrect predictions with high confidence. Conversely, it rewards the model when it makes correct and confident predictions.

Gradient of BCE Loss

The gradient of the BCE loss with respect to the predicted probability $\hat{y}$ is used during the optimization process to update model parameters. The gradient for a single instance is given by:

$$\frac{\partial \text{BCE}}{\partial \hat{y}} = - \left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}} \right)$$

This gradient is used in gradient descent algorithms to minimize the BCE loss by adjusting the model parameters in the direction that reduces the loss.

Advantages of BCE Loss

- **Probabilistic Interpretation:** BCE loss provides a probabilistic interpretation of the model's predictions, making it suitable for tasks where outputs are probabilities.
- **Penalty for Confidence:** It penalizes confident but wrong predictions more heavily, encouraging the model to be more accurate and cautious.
- **Smooth Gradient:** The smooth gradient of the BCE loss function allows for effective optimization using gradient-based methods.

Application in Neural Networks

In neural networks, the BCE loss function is often used in conjunction with a sigmoid activation function in the output layer. The sigmoid function squashes the output of the neural network into the range $(0, 1)$, which can then be interpreted as a probability. The BCE loss is then computed using these probabilities and the true binary labels.

The sigmoid activation function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

where x is the input to the activation function. The output of the sigmoid function is used as $\hat{y}$ in the BCE loss function.

3.4.3 Cosine Similarity

Cosine Similarity is a measure used to determine how similar two vectors are, irrespective of their magnitude. It is often used in text analysis and machine learning to measure the similarity between two non-zero vectors in a multidimensional space.

Definition

Cosine Similarity is defined as the cosine of the angle between two vectors in an inner product space. Mathematically, the cosine similarity between two vectors $\mathbf{A}$ and $\mathbf{B}$ is given by:

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

where:

- $\mathbf{A} \cdot \mathbf{B}$ is the dot product of the vectors $\mathbf{A}$ and $\mathbf{B}$,
- $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the magnitudes (or norms) of the vectors $\mathbf{A}$ and $\mathbf{B}$ respectively.

Formula for Cosine Similarity

Given two vectors $\mathbf{A} = [A_1, A_2, \dots, A_n]$ and $\mathbf{B} = [B_1, B_2, \dots, B_n]$, the cosine similarity is computed as:

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Interpretation

Cosine Similarity measures the cosine of the angle between two vectors. The value ranges from -1 to 1:

- Cosine Similarity = 1 indicates that the two vectors are identical.
- Cosine Similarity = 0 indicates that the two vectors are orthogonal (no similarity).
- Cosine Similarity = -1 indicates that the two vectors are diametrically opposed.

Conversion to Probability

In certain applications, it is useful to convert the cosine similarity to a probability-like measure in the range $[0, 1]$. This can be done by mapping the range $[-1, 1]$ to $[0, 1]$ using the following transformation:

$$P = \frac{c + 1}{2}$$

where:

- c is the cosine similarity value in the range $[-1, 1]$,
- P is the transformed value in the range $[0, 1]$.

This transformation linearly maps the cosine similarity to a probability measure, where:

- $c = -1$ maps to $P = 0$,
- $c = 1$ maps to $P = 1$,
- $c = 0$ maps to $P = 0.5$.

Advantages of Cosine Similarity

- **Magnitude Invariant:** Cosine similarity is independent of the magnitude of the vectors, focusing solely on the orientation.
- **Sparse Vectors:** It is particularly effective for high-dimensional, sparse vectors commonly encountered in text analysis.
- **Efficiency:** The computation is straightforward and efficient, making it suitable for large datasets.

Applications

Cosine similarity is widely used in various domains such as:

- **Text Analysis:** Measuring the similarity between documents or sentences.
- **Recommendation Systems:** Finding items similar to a user's preference.
- **Clustering:** Grouping similar items in clustering algorithms.

Chapter 4

Problem Statement

4.1 Problem proposed

Given the keystroke data vectors $K1$ and $K2$ for users $U1$ and $U2$, we aim to develop a function D to determine if users are cheating. The function D is defined as follows:

$$D(K1, K2) = \begin{cases} 0 & \text{if no cheating is detected} \\ 1 & \text{if cheating is detected} \end{cases}$$

Each keystroke data vector K_i for user U_i is defined as:

$$K_i = (t_{i1}, t_{i2}, \dots, t_{in_i})$$

where t_{ij} represents the j -th keystroke timing for user U_i and n_i is the number of keystrokes recorded for user U_i .

4.1.1 Detailed Explanation

In this context, we are given keystroke data vectors for two users, $K1$ for user $U1$ and $K2$ for user $U2$. These vectors contain the timing information for each keystroke made by the users.

- $K1$ is the keystroke data vector for user $U1$.
- $K2$ is the keystroke data vector for user $U2$.

The function D is designed to analyze these keystroke data vectors and determine whether there is any evidence of cheating. The output of the function D is binary:

- $D(K1, K2) = 0$: Indicates that no cheating is detected.
- $D(K1, K2) = 1$: Indicates that cheating is detected.

Each keystroke data vector K_i is a sequence of keystroke timings for a particular user U_i :

$$K_i = (t_{i1}, t_{i2}, \dots, t_{in_i})$$

Here:

- t_{ij} is the timing of the j -th keystroke for user U_i .
- n_i is the total number of keystrokes recorded for user U_i .

The function D compares these sequences to detect patterns or anomalies that may indicate cheating. This comparison could involve statistical analysis, pattern recognition, or other computational techniques to assess the similarity or discrepancy between the two keystroke data vectors.

Chapter 5

Methods

5.1 Data Collection

One might draw parallels between distinguishing answers written with or without assistance and discerning between fixed and free-form text. However, our emphasis lies squarely within the examination context. Hence, we curated our dataset to encompass various question sections tailored to the cognitive effort demanded from the writer in formulating their responses.

Following the ethical guidelines established by the Institutional Review Board (specifics to be disclosed upon dataset release), we conducted a study involving 40 students from a prominent research university. These participants underwent two typing sessions: the first session involved typing without external aids (*referred to as bonafide typing-Session 1*), while the second session allowed the use of the Internet and chatGPT on their mobile phones (*referred to as assisted typing-Session 2*). The student population consisted of 65

Prior research suggests that writing patterns differ between familiar and unfamiliar topics [52]. When addressing an unfamiliar topic, individuals often pause longer to review their text and organize their thoughts, utilizing the backspace key more frequently to maintain coherence. Conversely, writing about a familiar topic typically proceeds more swiftly and consistently [51]. To accommodate this variation, the study was structured into two parts:

1. **Opinion-based Section:** This section comprised two open-ended essay-type questions concerning societal, environmental, and current affairs topics. Participants were instructed to argue either for or against the given motion. For instance, "*Is AI a promising tool that can revolutionize industries and improve human life, or does it carry significant ethical risks, including job displacement and loss of control?*"
2. **Fact-based Section:** This section contained two scenario-based descriptive questions centered on high school-level science concepts. Participants were tasked with explaining their observations and justifying the underlying principles. For example, "*You are given a mixture of sand and salt. Design an experiment to separate the two components using appropriate laboratory equipment and techniques. Explain your step-by-step procedure and the underlying principles of the separation process.*"

Each participant was required to compose and type answers to four questions per session, details of which will be disclosed later through a separate webpage. Participants were instructed to provide responses of at least 300 characters for each question. Two distinct sets of questions were administered for Session 1 and Session 2, with each question categorized under one of the six cognitive load levels outlined in the report published by DARPA [53]. A participant’s typing session concluded after addressing all questions, typically spanning 30 to 45 minutes. Throughout this period, participants had the opportunity to edit their responses using the Delete or Backspace keys. A keystroke sensor recorded each key’s press and release, capturing event type (key-down or key-up), key code, and event time. These timestamps, generated by an external reference clock, maintain an accuracy within ± 200 microseconds. This dataset is denoted as the ”*Proposed dataset*” throughout the remainder of this paper.

5.2 Additional Datasets

We present our findings based on several existing keystroke datasets that closely align with the scenario outlined in this paper. Below are descriptions of each dataset utilized:

- *SBU dataset* [54]: This dataset comprises both truthful and deceptive writings from two domains: restaurant business reviews and essays on two social interest topics: (a) gun control and (b) gay marriage. Participants were also required to retype their typing.
- *Buffalo dataset* [55]: It encompasses keystroke data from 157 subjects engaged in transcribing fixed text and freely answering questions across three sessions using four different keyboards. To our knowledge, this dataset is unique in its recording of the keyboard types used by participants.

Table 5.1: Comparison of all datasets

Dataset	# Users	Free writing	Fixed writing	Context information	Keyboard information	Cognitive Load wise Questions	AI-assisted writing
SBU	196	✓	✓	✓	✗	✗	✗
Buffalo	148	✓	✓	✗	✓	✗	✗
Proposed	34	✓	✓	✗	✗	✓	✓

5.3 Feature Extraction

During rapid typing, users often press the next key before releasing the previous one, resulting in overlapping events in the keystroke logs. This overlapping pattern offers insights into

typing behavior. To process the raw data, we aligned each "Key Down" (KD) event with the corresponding "Key Up" (KU) event, searching within the next 20 entries. As the alphabet case is inconsequential at this stage, we standardized all keys to uppercase. Subsequently, we compiled a list of these synchronized KD-KU pairs, which were then used to analyze typing latencies. Depending on the desired level of granularity, features were extracted at three different levels - *Character*, *Word*, and *Window* - and analyzed in both time and frequency domains. Illustrated in Figure 5.4, we propose a framework of comparisons at these levels:

- *Character level*: At this level, we delve into key latencies through monographs and digraphs, as mentioned in the literature [56]. This involves analyzing the average Key Hold Times (KHT) for the 90 most commonly used keys, including special and non-printable characters. We also calculate the average Key Interval Times (KIT) for the top 30 and last 20 key pairs. Keys outside these groups are excluded, and unused keys among the selected 90 are assigned a value of 0. These 140 attributes are then employed for various ML-based classifiers.
- *Word level*: Here, we gather data on various aspects of typing, including the time required to type each word, delays between words, and average typing times for words of different lengths. We also track the occurrence of backspaces and key presses like Shift, Alt, and Ctrl, along with average pause duration and occurrences of long pauses before and after punctuation marks. These measures result in 27 distinct features, which are further analyzed using mean, standard deviation, and kurtosis to train and assess various ML classifiers [57].
- *Window level*: In this phase, we combine each sentence with the typing speeds for each word, aiming to detect subtle signs of assisted writing by analyzing both linguistic elements and variations in typing pauses. This analysis evaluates each paragraph for several metrics, including the total number of characters, words, and sentences and averages for word and sentence length. Additionally, we assess the count of nouns, verbs, modals, and modifiers per paragraph. User-specific comparisons are then conducted using various machine learning and deep learning models.

Here is the entire feature set. The feature set is divided into atomic, pausality, linguistic, and count-based features extracted from raw keystroke data.

Explanation:

- **Atomic Features**: Basic metrics such as the duration a key is pressed and the interval between key presses.
- **Pausality Features**: Characteristics related to pauses in typing, such as the duration of pauses, the number of keystrokes between pauses, and pauses before and after punctuation marks.
- **Linguistic Features**: Metrics related to the typed textual content, including the number of characters, words, sentences, and the frequency of specific parts of speech like nouns and verbs.

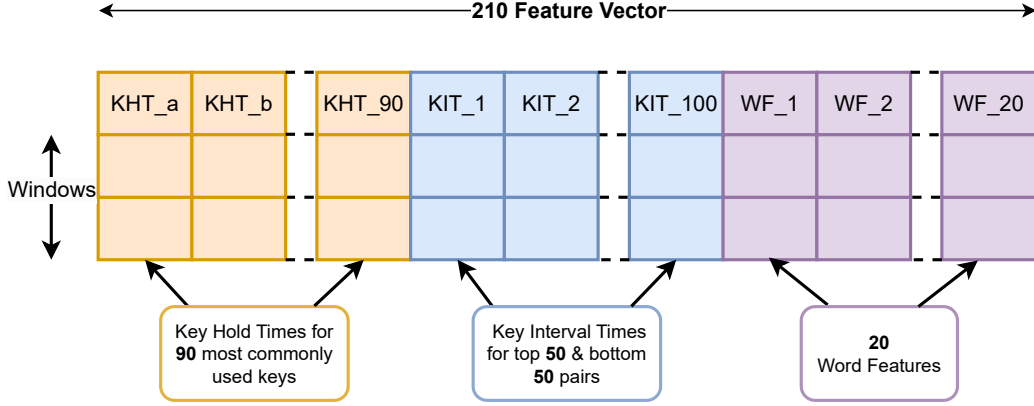


Figure 5.1: 210 long Feature Vector for each window

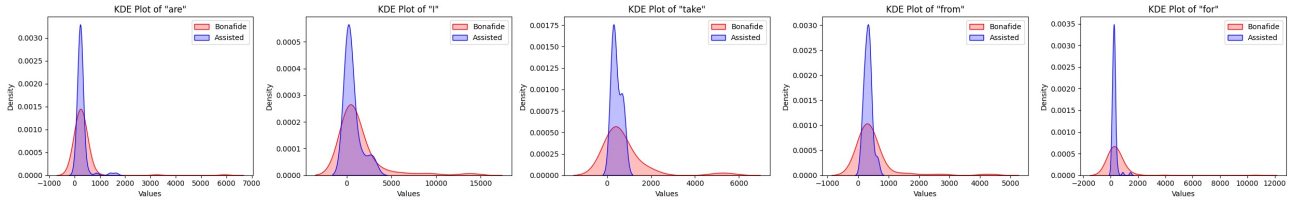


Figure 5.2: **KDE Plots** of some frequent words showing a clear difference in the probability function between the two forms of writing

- **Count-Based Features:** Counts of specific actions or key presses, such as the number of special characters, backspace usage, and typing speed for bursts of characters.

We have designed various evaluation scenarios depending on specific application contexts, detailed in later document sections.

5.4 Proposed Framework

We propose detecting cheating using two approaches: a data-driven deep learning model called TypeNet and a feature-driven machine learning model called LGBM classifier.

- **TypeNet (Data-Driven Deep Learning Model):** This approach leverages deep learning techniques to analyze raw keystroke data directly. By learning complex patterns and representations from the data itself, TypeNet can detect subtle anomalies that may indicate cheating without relying on manually engineered features. Figure 5.3
- **LGBM Classifier (Feature-Driven Machine Learning Model):** This approach uses manually designed features extracted from keystroke data, such as timing and

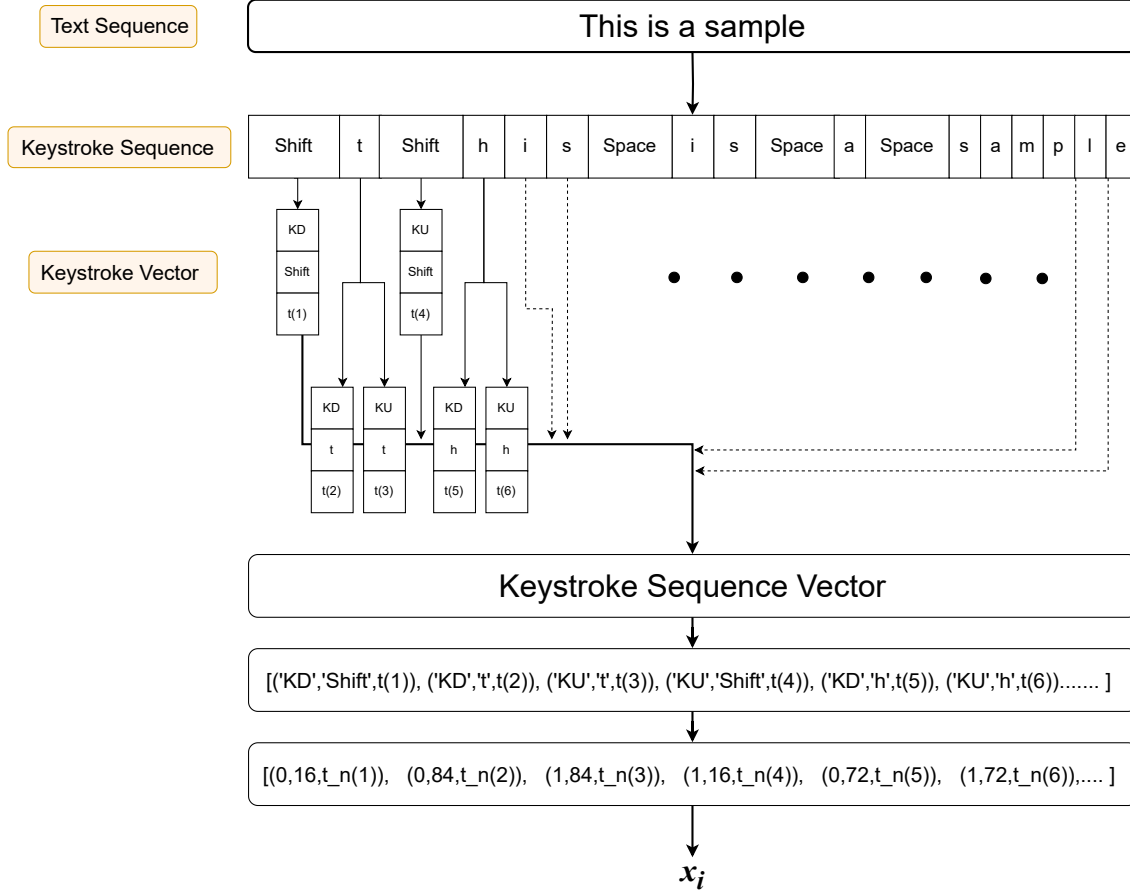


Figure 5.3: **Data preprocessing:** we standardize the sequence lengths to a predetermined length M by padding shorter sequences and clipping longer ones, adjusting M during hyperparameter tuning. The value of M ranged between 25-500 with batch size between 32-512 in powers of 2. Sequences with excessive use of the Shift key (over 20%) and those significantly shorter than M (less than 50%) were excluded. Each keystroke sequence includes three attributes: timestamps, keycodes, and key actions (KeyDown, KeyUp), normalized as follows. Timestamps were scaled to a $[0, 1]$ range using min-max normalization. Keycodes are normalized by dividing each by 255. Key actions are binary encoded with KeyDown as 0 and KeyUp as 1.

Feature Type	Names
Atomic	KHT, KIT
Pausality	Pauses after long words, Duration between pauses, Keystrokes between pauses, Pause before/after punctuation marks, words between pauses, Characters/Words between sentence beginning and first pause
Linguistic	Number of characters/words/sentences per answer, Average word/sentence lengths, Nouns/Verbs/Modifiers/Modals/Named Entity per sentence
Count	Number of special/printable/non-printable characters, Backspace/Shift/Alt/Ctrl count, characters per burst, word wise speeds

Table 5.2: Feature set across 4 different types

frequency of key presses. The LGBM classifier, a powerful gradient-boosting model, uses these features to identify patterns associated with cheating.

Combining these two approaches allows us to utilize the strengths of both deep learning and traditional machine learning to improve cheating detection accuracy.

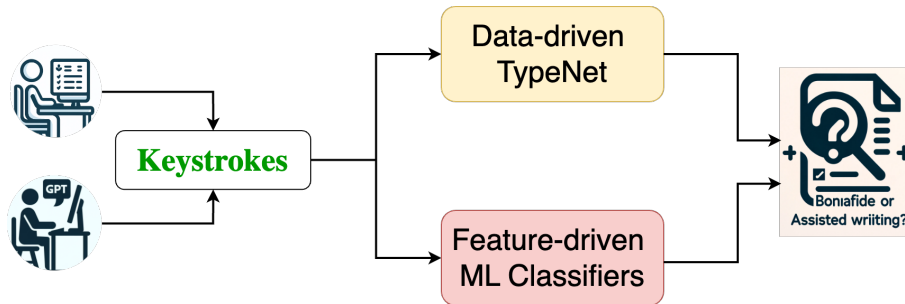


Figure 5.4: **Proposed Detection Methods:** data-driven deep learning model called TypeNet and feature-driven ML classifiers

We used a sliding window-based approach to extract features, as previous research indicates that shorter windows often perform better in scan-based methods. To determine the window size and slide, we calculated based on the average typing speed of 40 words per minute and an average of 4.5 characters per word, resulting in 180 characters per minute. This equates to approximately 350 keystrokes per minute.

Based on these calculations, we have used the following window sizes across various scenarios:

- **500 keystroke window size (500WS), 100 keystroke slide (100 Slide):** This corresponds to a window of 1-2 minutes with a slide of 30 seconds.

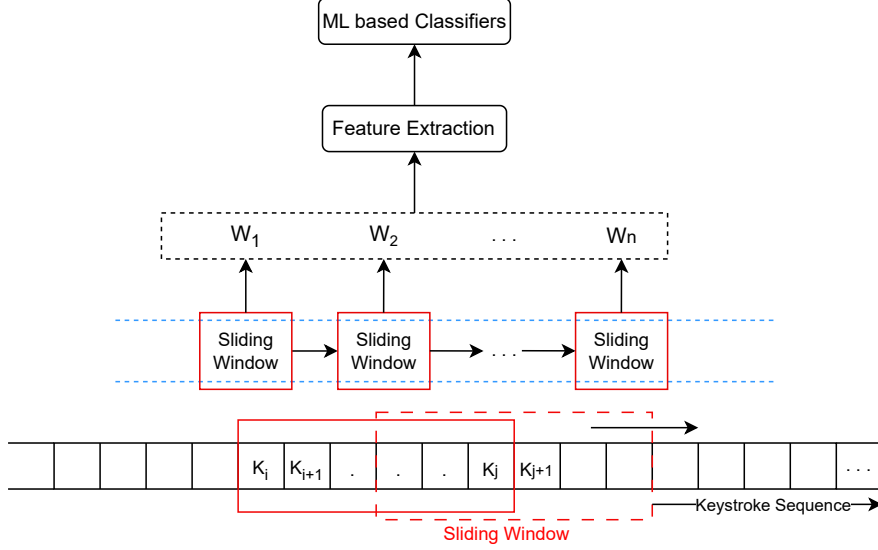


Figure 5.5: **Sliding Window**: across users with varying window size and slide, helps in extracting features from each windows

- **1000 keystroke window size (1000WS), 300 keystroke slide (300 Slide)**: This corresponds to a window of 2-3 minutes with a slide of 1 minute.
- **2000 keystroke window size (2000WS), 500 keystroke slide (500 Slide)**: This corresponds to a window of 4-5 minutes with a slide of 2 minutes.

As a middle ground, we have used the 1000WS, 300 Slide (2-3 minutes window size + 1 minute slide) for dataset and user-agnostic comparisons.

5.5 Detection Methods

We propose four distinct approaches tailored to the context and scale for distinguishing between bonafide and assisted typing:

- **ML Classifiers**: Widely recognized machine learning models offer a feature-driven approach to classification, with a proven track record in user authentication via keystroke dynamics[53, 58]. These models extract 20 distinct, manually designed features at the word level, as detailed in Section 5.3, and use these features for classification. The **LightGBM** (LGBM) model is particularly well-suited for analyzing keystroke data due to its efficiency in handling large datasets and robustness with diverse feature types. LGBM’s gradient boosting framework captures intricate patterns and interactions within keystroke features, such as atomic, pausality, linguistic, and count features. Its ability to manage missing values and perform well with high-dimensional data makes

it ideal for tasks like user authentication, behavior analysis, and anomaly detection in keystroke dynamics. Additionally, LGBM’s fast training speed and low memory usage enable scalable and real-time applications, enhancing the overall effectiveness of keystroke data analysis.

- **TypeNet**: In open-ended scenarios lacking a defined data collection process, LSTM-based models like TypeNet [59] demonstrate effectiveness. This data-driven approach leverages the sequential nature of keystroke data, outperforming other methods designed for handling time-series data.

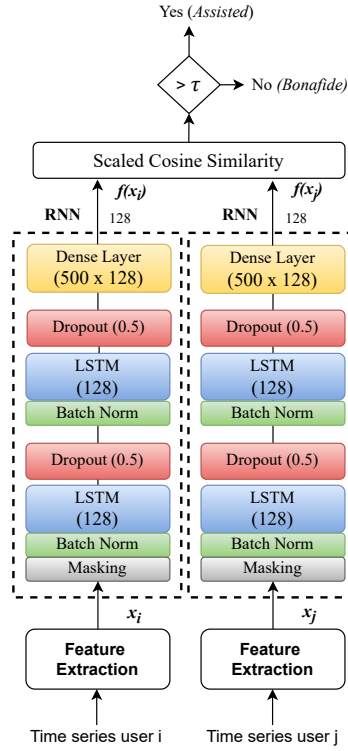


Figure 5.6: **System architecture**:, we modified the TypeNet architecture as two LSTM layers, each with a 128-dimensional output, interspersed with batch normalization layers and tanh activation functions. (2) A fully connected layer follows each LSTM to project the embeddings into a consistent 128-dimensional space. (3) Dropout layers are integrated after each LSTM layer to prevent overfitting, with a dropout rate of 0.5 and a recurrent dropout of 0.2. (4) Variable sequence length M and batch size during experiments, exploring ranges from 25 to 500 for M and 32 to 512 for batch size. We experimented with various training durations, noting that performance typically plateaued between 50 to 100 epochs. The Adam optimizer was utilized with a learning rate tailored from 0.0001 to 0.005, and L_2 regularization was included to control overfitting.

5.6 Evaluation Scenarios

- **User-specific and User-agnostic:** Our model’s performance is evaluated through user-specific and user-agnostic evaluations across individual and consolidated datasets, maintaining an 80-20 training-testing split. In user-agnostic scenarios, we explore two strategies: (1) training and testing across different datasets to assess performance variability and (2) merging datasets and splitting them with distinct user sets for training, validation, and testing, using varied split ratios like 50-25-25 and 80-10-10 to evaluate generalizability. For user-specific scenarios, we split each user’s sequences into an 80-20 ratio between training and testing sets, ensuring no overlap in sequences between them.
- **Keyboard-specific and Keyboard-agnostic:** Model performance is assessed in keyboard-specific and keyboard-agnostic scenarios using the Buffalo dataset, which includes interactions with four keyboard types. It follows an 80-20 split for training and testing in keyboard-specific cases, training on three keyboard types while testing on the fourth for keyboard-agnostic evaluations. For keyboard-specific evaluations, individual datasets maintain an 80-20 ratio of keyboard-specific sequences in train and test sets, ensuring no overlapping sequences.
- **Content-specific and Content-agnostic:** Distinct models are developed for each dataset with training-testing sets based on specific content. Performance evaluation involves testing the models on the two remaining datasets. The content-specific strategy maintains topic consistency in training and testing sets, while the content-agnostic approach evaluates the model’s adaptability by training on two topics and testing on a third, previously unseen topic, assessing proficiency in linguistic analysis within both seen and unseen contexts.
- **Dataset-specific and Dataset-agnostic** In the *dataset-specific* approach, we train and test the model using the same dataset, ensuring no sequence overlap between the training and testing sets. This method evaluates the model’s performance in a consistent typing environment. Conversely, the *dataset-agnostic* approach involves training the model on a mix of datasets and testing it on different datasets or combinations of datasets. This strategy assesses the model’s ability to generalize and remain effective across various environments, which may vary greatly in their curation. By employing both evaluation methods, we aim to thoroughly understand the model’s strengths and weaknesses in different data collection environments, ensuring its suitability and generalizability for real-world applications.

5.7 Evaluation Metrics

In all evaluation scenarios, the class ratio remains below **1.15**, indicating a nearly balanced class distribution. Therefore, it is crucial to assess *accuracy*, *false acceptance rate (FAR)*,

false rejection rate (FRR), and *F1 score* for a comprehensive evaluation of these binary classification models. Accuracy provides a quick measure of overall model performance, while FAR and FRR offer insights into the likelihood of false positives and negatives, respectively. The F1 score, balancing precision and recall, ensures the model accurately identifies classes without significant bias towards over- or under-prediction. Together, these metrics effectively reveal the strengths and weaknesses of the model in situations with balanced classes.

- **Accuracy**↑: Measures the overall effectiveness of the system in identifying both cheaters and honest individuals.
- **F1 Score**↑: Indicates how well the system balances catching cheaters without falsely accusing innocent people.
- **FAR**↓: Represents the percentage of cheaters mistakenly identified as honest.
- **FRR**↓: Indicates the percentage of honest individuals incorrectly flagged as cheaters.

Chapter 6

Results

The results of the LGBM classifier across the four evaluation scenarios are presented in Table 6.1. For comparison, I have included results from TypeNet. The performance of each model under all evaluation scenarios is detailed in Table 6.1 and explained in the following paragraphs.

6.0.1 Keyboard-Specific and Keyboard-Agnostic Scenarios

Typenet: In keyboard-specific scenarios, where the keyboard type is consistent and known, the models achieve accuracy in the range of 74.98% to 84.64% and F_1 scores between 73.34% to 83.45%. In keyboard-agnostic scenarios, the models attain an accuracy between 78.11% to 80.54% and F_1 scores between 76.89% to 78.77%. In both cases, the FAR and FRR remain below 30%. The insignificant differences between accuracy, F_1 score, and error rates suggest that the proposed plagiarism detector is robust to the type of keyboard used.

LGBM Classifier: The LGBM classifier is robust against changes in keyboard types. When tasked with a specific goal, the behavior modeling remains unaffected by the type of keyboard used, demonstrating the model’s versatility across different typing setups. In both keyboard-specific and keyboard-agnostic scenarios, the classifier achieves accuracy rates above 97% and F1 scores exceeding 98%, while maintaining a False Acceptance Rate (FAR) below 3% and a False Rejection Rate (FRR) below 7%.

6.0.2 Context-Specific and Context-Agnostic Scenarios

Typenet: In context-specific scenarios, TypeNet achieves accuracy in the range of 76.52% to 80.24% and F_1 scores between 78.01% to 81.52%, with FAR between 30.81% to 34.62%, and FRR between 21.27% to 31.13%. These metrics demonstrate the model’s effectiveness in environments that match its training conditions. However, performance slightly drops in context-agnostic scenarios, where the conditions vary from the training settings. Accuracy ranges from 70.21% to 78.67%, F_1 scores from 70.23% to 78.24%, FAR increases to between 32.3% to 39.65% from 30.81% to 34.62%, and FRR extends from 24.1% to 34.73%

from 24.1% to 34.73%. Despite the drops, performance remains encouraging, indicating only modest reductions from the context-specific results. This highlights TypeNet’s adaptability across varying contexts.

LBGM Classifier: Modest reductions in metrics are observed across different contexts, which is expected since genre and writing tasks influence typing rhythms and patterns. In context-specific scenarios, accuracy ranges between 86-8% with F1 scores around 85%, while both the False Acceptance Rate (FAR) and False Rejection Rate (FRR) remain below 15%.

6.0.3 User-Specific and User-Agnostic Models

Typenet: In the user-specific models, where training and testing sets contain keystroke sequences from the same user with no overlap, the achieved accuracy and F_1 scores were 81.86% and 81.85%, respectively. FAR and FRR were recorded at 23.71% and 26.24%. On the other hand, user-agnostic models achieved accuracies between 63.56% and 66.54%. FAR and FRR ranged between 38.82% - 42.51% and 39.57% - 39.86%, respectively. These findings suggest that the model performs better when trained and tested on keystroke sequences from the same user, indicating limited generalizability across different users’ typing patterns. This is because there are significant differences in users’ typing behaviors, even when typing the same content, which underscores the effectiveness of continuous authentication via keystroke dynamics.

LBGM Classifier: The initial user-specific results, when trained on the first 70% of windows and tested on the remaining 30%, showed an 8% drop from the expected 89%. This was due to differences in typing patterns between the initial phases (Proposer, Translator, Transcriber) and the final revision phase (Evaluator). To address this, I shuffled all windows for each user and retrained with a 70-30 split. As anticipated, performance improved to 89%. Consequently, in both user-specific and user-agnostic cases, the accuracy is around 88%, with an F1 score of 88%, and the False Acceptance Rate (FAR) and False Rejection Rate (FRR) below 12% and 10%, respectively.

6.0.4 Dataset-Specific and Dataset-Agnostic Scenarios

Typenet: In dataset-specific scenarios, the TypeNet model demonstrates significant improvement, achieving accuracy and F_1 scores above 80%, with the highest accuracy recorded at 85.72%. FAR and FRR were both below 23%, with the lowest being 17.64% and 23.91%, respectively (see Table ??). This indicates excellent performance when the model is trained and tested on a dataset specifically curated in a controlled environment. In dataset-agnostic scenarios, where the model is trained across multiple datasets, two cases show higher accuracy and lower FAR/FRR when the training includes a mix of keystroke sequences from the SBU dataset (S) and either the Buffalo (B) or Proposed (P) datasets. This maximizes the training data pool, resulting in a better model and enhanced performance. The model achieves its best results when trained on a combination of the SBU and Proposed datasets

and tested on the Buffalo dataset, with an accuracy, F1 score, FAR, and FRR of 73.23%, 72.65%, 27.95%, and 40.22%, respectively. However, in other cases where the training data is less diverse, accuracy and F_1 score fall below 60%, with FAR and FRR exceeding 40% and 45%, respectively, as shown in Table ???. These results suggest that TypeNet requires more extensive data to generalize across different environments. However, with a larger dataset for training, the model’s performance improves, indicating that increased data volume could enhance its robustness.

LBGM Classifier: In dataset-specific scenarios, the accuracies are above 90% for all three datasets. However, in dataset-agnostic scenarios, the performance has decreased. Data diversity in these scenarios slightly reduces performance, and increased data variance further deteriorates performance as more data from different datasets is included in the training set. Models trained on a single dataset perform well on a combination of the other two datasets, achieving the highest accuracy of 75%. Conversely, models trained on the combination of two datasets underperform on the third dataset, with the highest accuracy reaching only 71%. It is important to note that I have kept the window size consistent across all three datasets (1000 window size with a 300 slide) for comparison purposes. However, this uniform window size may contribute to poor performance in combined training scenarios. Each dataset might have been collected under different conditions, necessitating different window sizes. Therefore, the choice of window size is crucial when comparing across multiple datasets or data collection environments.

6.1 Ablation study

6.1.1 LGBM Classifier

The user-specific performances in the LGBM classifier were actually worse than the user-agnostic performances. This is counter-intuitive since writing behavior changes across users but remains consistent within the same user. To investigate this, I conducted ablation studies and analyzed the writing process further. I found that the end revisions (Evaluator) are different from the Proposer, Translator, and Transcriber stages. Therefore, I shuffled all the windows and retrained with the same 70-30 split. As expected, the user-specific performances improved from 81.87% to 88.89%, confirming our hypothesis that writing behavior is distributed across four distinct phases.

6.1.2 TypeNet

The performance of deep learning architectures improved with increased data variance, enabling better generalization with more data. Models trained on a single dataset achieved accuracies ranging from 53% to 59%. However, models trained on a combination of two datasets showed improved performance, reaching approximately 73% accuracy. This confirms our expectation that models learn better with more data. It is well-known that typical

Table 6.1: all results

Dataset		TypeNet				LGBM			
Train	Test	Acc. [↑]	F1 [↑]	FAR [↓]	FRR [↓]	Acc. [↑]	F1 [↑]	FAR [↓]	FRR [↓]
<i>Keyboard Specific</i>									
K_0	K_0	84.64	83.45	25.38	19.83	98.74	99.10	0.36	3.55
K_1	K_1	80.02	78.91	29.11	24.19	97.91	98.51	1.48	3.47
K_2	K_2	77.77	76.58	32.14	25.12	96.17	97.24	2.75	6.25
K_3	K_3	74.98	73.34	32.60	30.16	97.14	97.99	1.10	7.07
<i>Keyboard Agnostic</i>									
(K_0, K_1, K_2)	K_3	78.11	75.88	28.04	29.01	97.19	98.03	1.21	6.64
(K_0, K_1, K_3)	K_2	79.71	78.06	27.50	28.37	97.94	98.51	1.89	2.41
(K_0, K_2, K_3)	K_1	80.54	78.77	24.01	28.37	97.57	98.26	2.12	3.12
(K_1, K_2, K_3)	K_0	78.96	76.89	29.15	24.95	97.24	98.07	1.30	6.30
<i>Context Specific</i>									
GM	GM	80.24	81.52	30.81	21.27	88.68	85.71	13.63	9.81
GC	GC	79.39	80.67	34.62	22.33	87.58	84.61	15.67	10.18
RF	RF	76.52	78.01	34.02	31.13	86.71	82.86	11.83	15.64
<i>Context Agnostic</i>									
(GM, RF)	GC	72.96	72.40	38.30	28.56	85.50	82.24	17.05	12.75
(GC, RF)	GM	78.67	78.24	32.30	24.10	85.89	82.59	14.84	13.61
(GM, GC)	RF	70.21	70.23	39.65	34.73	86.63	81.79	21.14	8.58
<i>User Specific</i>									
Combined (56-14-30)		81.86	81.85	23.71	26.24	81.87	80.66	26.73	8.95
Combined-Shuffled (56-14-30)		-	-	-	-	88.89	89.33	9.89	12.39
<i>User Agnostic</i>									
Combined (80-10-10)		63.56	62.98	42.51	39.57	89.10	88.45	11.51	10.33
Combined (50-25-25)		66.54	66.22	38.82	39.86	88.84	88.14	12.13	10.27
<i>Dataset Specific</i>									
SBU	SBU	80.85	83.31	18.83	32.93	88.98	86.10	13.21	9.59
Proposed	Proposed	81.04	82.16	22.60	28.20	92.42	90.93	12.20	4.04
Buffalo	Buffalo	85.72	84.72	17.64	23.91	98.39	98.86	1.10	2.80
<i>Dataset Agnostic</i>									
(SBU, Proposed)	Buffalo	68.72	66.21	33.13	39.66	71.98	75.74	35.46	12.34
(SBU, Buffalo)	Proposed	73.23	72.65	27.95	40.22	58.14	13.39	3.96	92.44
(Proposed, Buffalo)	SBU	52.24	61.86	47.53	48.51	67.25	64.84	29.88	34.90
SBU	(Proposed, Buffalo)	59.73	57.03	41.36	44.29	75.26	78.53	31.76	10.89
Proposed	(Buffalo, SBU)	56.17	54.13	44.95	45.72	46.97	30.30	78.29	24.42
Buffalo	(Proposed, SBU)	53.57	53.16	46.32	48.01	68.37	59.78	45.39	21.22

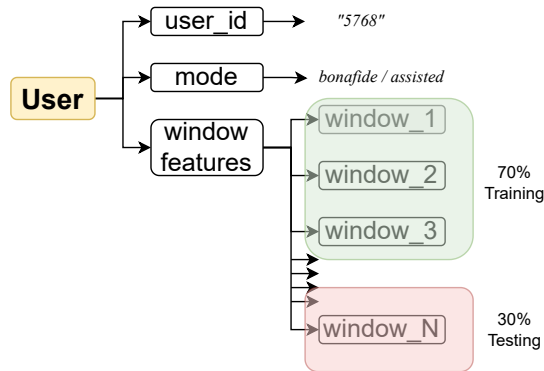


Figure 6.1: **LGBM Classifier**: A 70-30 train-test split is used for **User-Specific** training. However, according to Hayes’s cognitive writing model, the final revisions differ from the initial drafts. Therefore, shuffling these segments and retraining for user-specific evaluations helps improve accuracy.

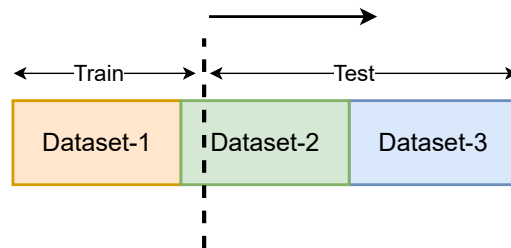


Figure 6.2: **TypeNet**: In **data-agnostic** scenarios, increased data variance in a deep learning model improves performance as the boundary separation between training and testing sets shifts. Models trained on a single dataset do not generalize well to other datasets. However, training on a combination of two datasets enhances accuracy, demonstrating that greater data variance helps the model learn more effectively.

deep learning architectures tend to perform better in such diverse scenarios.

6.2 Discussions

Upon reviewing the effectiveness of the two detection methods in various evaluation settings, it is clear that no single model excels in every scenario. Latency and word feature-based classifiers perform well in consistent typing environments but struggle to generalize when data collection varies across datasets. LSTM-based models, such as TypeNet, show strong performance in user-specific scenarios, achieving over 80% accuracy on the same dataset, and around 70% accuracy in user-agnostic situations on the same dataset. However, their accuracy declines to 60% when faced with new datasets and varying typing setups.

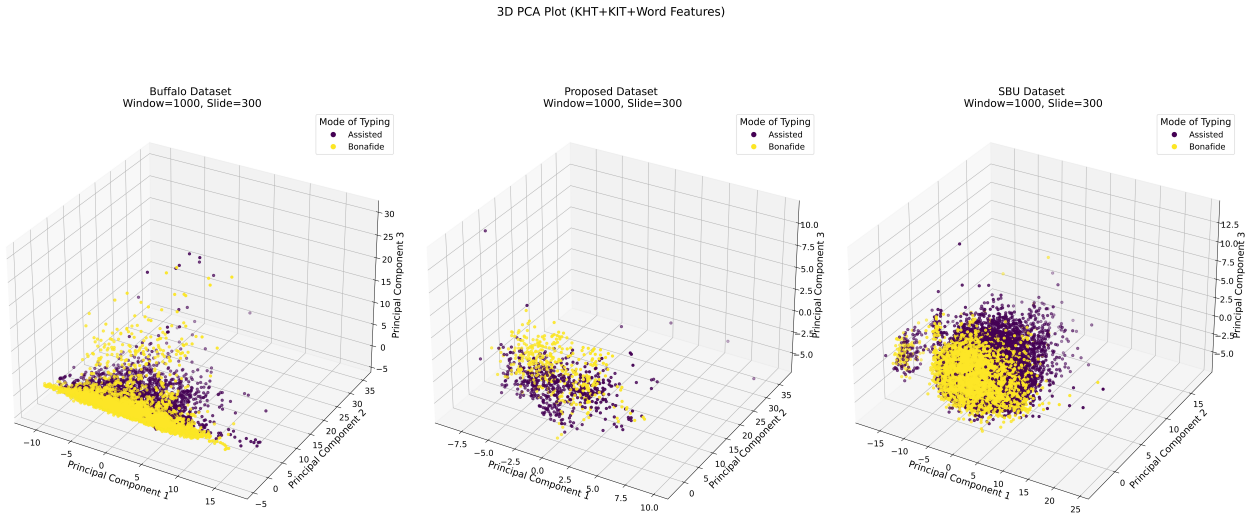


Figure 6.3: **Same window size:** PCA plots of the feature set indicate limited separability in the proposed dataset when using 1000 keystrokes with a 300-slide window size.

We further applied dimensionality reduction techniques such as t-SNE and PCA to investigate data separability between bonafide and assisted writing. The compiled plots are presented in Figure 6.3 and 6.4. By adjusting hyperparameters for optimal perplexity, sample size, and scaling methods, we found that while these two writing styles are separable within the same dataset, they tend to overlap when comparing across different datasets. This overlap suggests a loss of model generalization, meaning models trained on one dataset do not perform as well on others, as confirmed by our performance metrics.

Additionally, the choice of window size affects separability. Figure 6.3 shows that with a window size of 1000 keystrokes and a 300-slide (i.e., 2-3 minutes window + 1 minute slide), there is limited separability across all datasets, particularly in the proposed dataset.

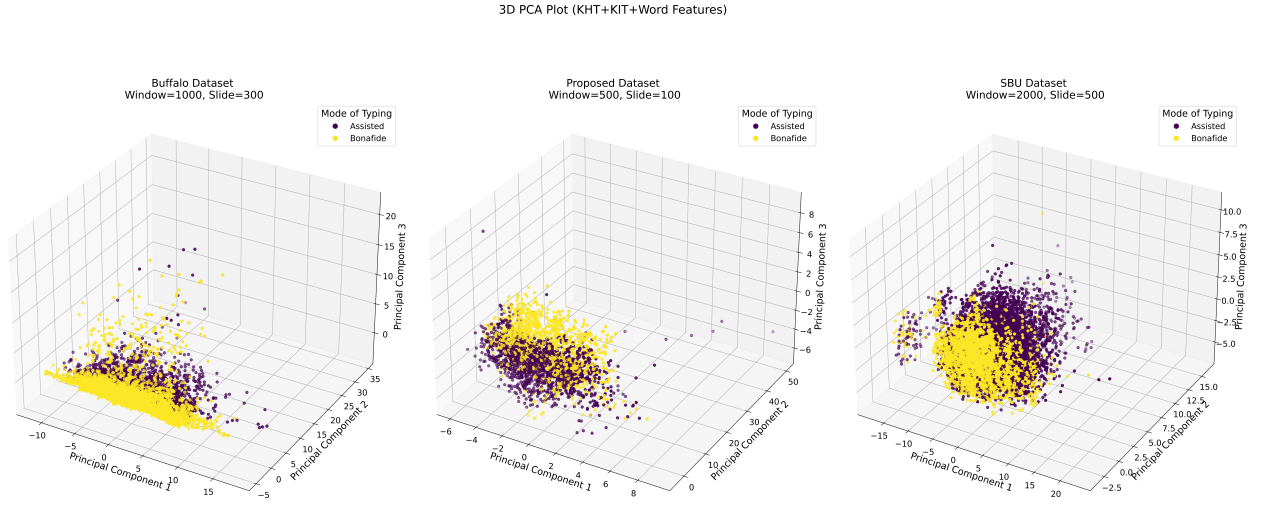


Figure 6.4: **Varying window size:** PCA plots with different window sizes and slides—1000 keystrokes with a 300-slide for Buffalo, 500 keystrokes with a 100-slide for the proposed dataset, and 2000 keystrokes with a 500-slide for the SBU dataset—show good separability. This implies that using different window sizes for various tasks and data collection environments is beneficial.

However, as we adjust the window sizes and slides, separability improves. Figure 6.4 shows PCA plots with different window sizes and slides—1000 keystrokes with a 300-slide for Buffalo (i.e., 2-3 minutes window + 1 minute slide), 500 keystrokes with a 100-slide for the proposed dataset (2-3 minutes window + 30 seconds slide), and 2000 keystrokes with a 500-slide (4-5 minutes window + 2 minutes slide) for the SBU dataset—demonstrating good separability. This indicates that using different window sizes for various tasks and data collection environments is beneficial.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion and Future Scope

In our study, we introduce a novel dataset aimed at detecting plagiarism using keystroke dynamics. This dataset establishes baseline results and encourages further advancements in this research area. We perform a comprehensive analysis of the dataset, utilizing data visualization techniques to gain a deeper understanding of keystroke patterns. Our methodology features a multi-layered pipeline, starting with the extraction of character-level features. We then progress to word-level, sentence-level, paragraph-level, and finally, session-level analysis.

Keystroke dynamics, which capture detailed timing information during typing, can effectively differentiate between genuine (bonafide) and assisted writing by modeling unique writing behaviors.

Feature-driven approaches, when using the correct window size, can outperform deep learning models in analyzing keystroke data. These methods better capture the nuanced patterns and variations in keystroke dynamics. Despite showing promising results in many scenarios, feature-driven approaches face challenges in generalizing across different conditions. To address this, we propose an **adaptive sliding window** technique that adjusts the window size dynamically based on the input data, enhancing the model’s ability to generalize.

To ensure low false positive and false negative rates, incorporating a **confidence score** in the model is crucial. This confidence score indicates the model’s certainty in its predictions. If the confidence score for a sample falls below a predefined threshold, the sample can be classified into a third category, rather than forcing a potentially incorrect classification. This approach helps reduce ethical concerns by preventing misclassification of sensitive data.

Additionally, it is essential to consider inductive bias, which refers to the inherent assumptions made by the model. One significant inductive bias in keystroke dynamics is the difference in writing behavior when composing on familiar versus unfamiliar topics. People tend to type more consistently and confidently on familiar topics, whereas unfamiliar topics may result in more varied and hesitant keystroke patterns. Understanding and accounting for this bias can further enhance the accuracy and reliability of keystroke dynamics models.

Our deep learning approach enhances the model’s ability to adapt to individual users’ keystroke dynamics, resulting in a 10-15% improvement in accuracy when customized for specific users. However, despite these promising results in controlled environments, the model’s performance decreases significantly when applied to datasets from varied contexts. This highlights a critical area for future research, emphasizing the need to maintain generalizability and practical applicability.

Our study underscores the importance of cautious interpretation when analyzing individual keystroke features, as they can be sensitive to minor variations in writing tasks [48]. Given a profile of fundamental typing behavior, it becomes possible to explore the differences between authentic and assisted writing profiles. However, the usefulness, adaptability, and efficiency—particularly in eliminating false allegations—of such applications remain subjects for future research.

Moving forward, our objective is to investigate the integration of writing analysis, eye tracking, and mouse behavior with typing rhythm to develop a comprehensive multi-modal system for identifying cheating. This approach will significantly benefit from advancements in *human-computer interaction* research and interface design.

Bibliography

- [1] S. M. Alzahrani, N. Salim, and A. Abraham, “Understanding plagiarism linguistic patterns, textual features, and detection methods,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 133–149, 2011.
- [2] K. Rao, “Plagiarism, a scourge,” *Current Science*, pp. 581–586, 2008.
- [3] M. Zhang, H. Guo, and X. Liu, “Using keystroke analytics to understand cognitive processes during writing,” *International Educational Data Mining Society*, 2021.
- [4] A. A. Wahab, D. Hou, S. Schuckers, and A. Barbir, “Utilizing keystroke dynamics as additional security measure to protect account recovery mechanism.” in *ICISSP*, 2021, pp. 33–42.
- [5] A. Esmaili, H. Hekmatmehr, S. Atashrouz, S. A. Madani, M. Pourmahdi, D. Nedeljkovic, A. Hemmati-Sarapardeh, and A. Mohaddespour, “Insights into modeling refractive index of ionic liquids using chemical structure-based machine learning methods,” *Scientific Reports*, vol. 13, no. 1, p. 11966, 2023.
- [6] I. Anderson, “Avoiding plagiarism in academic writing,” *Nursing standard*, vol. 23, no. 18, pp. 35–38, 2009.
- [7] D. Pecorari, “Academic writing and plagiarism,” 2008.
- [8] M. Roig, “Avoiding plagiarism, self-plagiarism, and other questionable writing practices: A guide to ethical writing,” *The Office of Research Integrity (ORI)*, 2015.
- [9] H. Zhang and T. W. Chow, “A coarse-to-fine framework to efficiently thwart plagiarism,” *Pattern Recognition*, vol. 44, no. 2, pp. 471–487, 2011.
- [10] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price, and B. Akram, “Detecting chatgpt-generated code submissions in a cs1 course using machine learning models,” 2024.
- [11] P. S. Teh, A. B. J. Teoh, S. Yue *et al.*, “A survey of keystroke dynamics biometrics,” *The Scientific World Journal*, vol. 2013, 2013.

- [12] P. Deane and M. Zhang, “Exploring the feasibility of using writing process features to assess text production skills,” *ETS Research Report Series*, vol. 2015, no. 2, pp. 1–16, 2015.
- [13] N. Agarwal, N. F. Danielsen, P. K. Gravdal, and P. Bours, “Contract cheat detection using biometric keystroke dynamics,” in *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2022, pp. 15–21.
- [14] E. A. Kochegurova and R. P. Zateev, “Hidden monitoring based on keystroke dynamics in online examination system,” *Programming and Computer Software*, vol. 48, no. 6, pp. 385–398, 2022.
- [15] P. K. Mungai and R. Huang, “Using keystroke dynamics in a multi-level architecture to protect online examinations from impersonation,” in *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. IEEE, 2017, pp. 622–627.
- [16] J. R. Hayes, “Modeling and remodeling writing,” *Written communication*, vol. 29, no. 3, pp. 369–388, 2012.
- [17] L. Flower and J. R. Hayes, “A cognitive process theory of writing,” *College composition and communication*, vol. 32, no. 4, pp. 365–387, 1981.
- [18] A. Patel, K. Bakhtiyari, and M. Taghavi, “Evaluation of cheating detection methods in academic writings,” *Library Hi Tech*, vol. 29, no. 4, pp. 623–640, 2011.
- [19] F. Kamalov, H. Sulieman, and D. Santandreu Calonge, “Machine learning based approach to exam cheating detection,” *Plos one*, vol. 16, no. 8, p. e0254340, 2021.
- [20] W. Alsabhan, “Student cheating detection in higher education by implementing machine learning and lstm techniques,” *Sensors*, vol. 23, no. 8, p. 4149, 2023.
- [21] Z. Li, Z. Zhu, and T. Yang, “A multi-index examination cheating detection method based on neural network,” in *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*. IEEE, 2019, pp. 575–581.
- [22] M. Roa’a, I. A. Aljazaery, and A. Alaidi, “Automated cheating detection based on video surveillance in the examination classes,” *International Journal of Interactive Mobile Technologies*, vol. 16, no. 08, p. 125, 2022.
- [23] L. C. O. Tiong and H. J. Lee, “E-cheating prevention measures: detection of cheating at online examinations using deep learning approach—a case study,” *arXiv preprint arXiv:2101.09841*, 2021.
- [24] B. Ayotte, M. Banavar, D. Hou, and S. Schuckers, “Fast free-text authentication via instance-based keystroke dynamics,” *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 2, no. 4, pp. 377–387, 2020.

- [25] Y. Muliono, H. Ham, and D. Darmawan, “Keystroke dynamic classification using machine learning for password authorization,” *Procedia Computer Science*, vol. 135, pp. 564–569, 2018.
- [26] J. Kim and P. Kang, “Freely typed keystroke dynamics-based user authentication for mobile devices based on heterogeneous features,” *Pattern Recognition*, vol. 108, p. 107556, 2020.
- [27] H. Lee, J. Y. Hwang, D. I. Kim, S. Lee, S.-H. Lee, J. S. Shin *et al.*, “Understanding keystroke dynamics for smartphone users authentication and keystroke dynamics on smartphones built-in motion sensors,” *Security and Communication Networks*, vol. 2018, 2018.
- [28] A. T. Kiyani, A. Lasebae, K. Ali, M. U. Rehman, and B. Haq, “Continuous user authentication featuring keystroke dynamics based on robust recurrent confidence model and ensemble learning approach,” *IEEE Access*, vol. 8, pp. 156 177–156 189, 2020.
- [29] F. Monrose, M. K. Reiter, and S. Wetzel, “Password hardening based on keystroke dynamics,” in *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999, pp. 73–82.
- [30] K. Mansour and K. Mahmoud, “A new approach for textual password hardening using keystroke latency times.” *Int. Arab J. Inf. Technol.*, vol. 18, no. 3, pp. 336–346, 2021.
- [31] A. Morales, A. Acien, J. Fierrez, J. V. Monaco, R. Tolosana, R. Vera, and J. Ortega-Garcia, “Keystroke biometrics in response to fake news propagation in a global pandemic,” in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 1604–1609.
- [32] R. Banerjee, S. Feng, J. S. Kang, and Y. Choi, “Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1469–1473. [Online]. Available: <https://aclanthology.org/D14-1155>
- [33] A. Kuruvilla, R. Daley, and R. Kumar, “Spotting fake profiles in social networks via keystroke dynamics,” in *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*. IEEE, 2024, pp. 525–533.
- [34] T. Bhattasali and K. Saeed, “Typing pattern analysis for fake profile detection in social media,” in *Computer Information Systems and Industrial Management: 20th International Conference, CISIM 2021, Elk, Poland, September 24–26, 2021, Proceedings 20*. Springer, 2021, pp. 17–27.

- [35] I. Tsimperidis, P. D. Yoo, K. Taha, A. Mylonas, and V. Katos, “R 2 bn: An adaptive model for keystroke-dynamics-based educational level classification,” *IEEE transactions on cybernetics*, vol. 50, no. 2, pp. 525–535, 2018.
- [36] A. N. H. Nahin, J. M. Alam, H. Mahmud, and K. Hasan, “Identifying emotion by keystroke dynamics and text pattern analysis,” *Behaviour & Information Technology*, vol. 33, no. 9, pp. 987–996, 2014.
- [37] A. Milne, K. Farrahi, and M. A. Nicolaou, “Less is more: Univariate modelling to detect early parkinson’s disease from keystroke dynamics,” in *Discovery Science: 21st International Conference, DS 2018, Limassol, Cyprus, October 29–31, 2018, Proceedings 21*. Springer, 2018, pp. 435–446.
- [38] L. M. Vizer, L. Zhou, and A. Sears, “Automated stress detection using keystroke and linguistic features: An exploratory study,” *International Journal of Human-Computer Studies*, vol. 67, no. 10, pp. 870–886, 2009.
- [39] A. Acien, A. Morales, R. Vera-Rodriguez, J. Fierrez, I. Mondesire-Crump, T. Arroyo-Gallego *et al.*, “Detection of mental fatigue in the general population: Feasibility study of keystroke dynamics as a real-world biomarker,” *JMIR Biomedical Engineering*, vol. 7, no. 2, p. e41003, 2022.
- [40] A. Pentel, “Predicting age and gender by keystroke dynamics and mouse patterns,” in *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, 2017, pp. 381–385.
- [41] I. Tsimperidis, C. Yucel, and V. Katos, “Age and gender as cyber attribution features in keystroke dynamic-based user classification processes,” *Electronics*, vol. 10, no. 7, p. 835, 2021.
- [42] B. Plank, “Predicting authorship and author traits from keystroke dynamics,” in *Proceedings of the Second Workshop on Computational Modeling of People’s Opinions, Personality, and Emotions in Social Media*, 2018, pp. 98–104.
- [43] S. Z. S. Idrus, E. Cherrier, C. Rosenberger, and P. Bours, “Soft biometrics for keystroke dynamics,” in *Image Analysis and Recognition: 10th International Conference, ICIAR 2013, Póvoa do Varzim, Portugal, June 26–28, 2013. Proceedings 10*. Springer, 2013, pp. 11–18.
- [44] A. A. Buker, G. Roffo, and A. Vinciarelli, “Type like a man! inferring gender from keystroke dynamics in live-chats,” *IEEE Intelligent Systems*, vol. 34, no. 6, pp. 53–59, 2019.
- [45] M. Mohlala, A. R. Ikuesan, and H. S. Venter, “User attribution based on keystroke dynamics in digital forensic readiness process,” in *2017 IEEE Conference on Application, Information and Network Security (AINS)*. IEEE, 2017, pp. 124–129.

- [46] P. R. Borj and P. Bours, “Detecting liars in chats using keystroke dynamics,” in *Proceedings of the 2019 3rd International Conference on Biometric Engineering and Applications*, 2019, pp. 1–6.
- [47] Y. M. Lim, A. Ayesh, and M. Stacey, “Using mouse and keyboard dynamics to detect cognitive stress during mental arithmetic,” in *Intelligent Systems in Science and Information 2014: Extended and Selected Results from the Science and Information Conference 2014*. Springer, 2015, pp. 335–350.
- [48] R. Conijn, J. Roeser, and M. Van Zaanen, “Understanding the keystroke log: the effect of writing task on keystroke features,” *Reading and Writing*, vol. 32, no. 9, pp. 2353–2374, 2019.
- [49] D. G. Brizan, A. Goodkind, P. Koch, K. Balagani, V. V. Phoha, and A. Rosenberg, “Utilizing linguistically enhanced keystroke dynamics to predict typist cognition and demographics,” *International Journal of Human-Computer Studies*, vol. 82, pp. 57–68, 2015.
- [50] S. S. Zeid, R. A. ElKamar, and S. I. Hassan, “Fixed-text vs. free-text keystroke dynamics for user authentication,” *Engineering Research Journal-Faculty of Engineering (Shoubra)*, vol. 51, no. 1, pp. 95–104, 2022.
- [51] K. Trezise, T. Ryan, P. de Barba, and G. Kennedy, “Detecting academic misconduct using learning analytics,” *Journal of Learning Analytics*, vol. 6, no. 3, pp. 90–104, 2019.
- [52] H. G. Lettner-Rust, “Questions of transfer: Writers’ perspective on familiar/unfamiliar writing tasks in a capstone writing course,” 2010.
- [53] K. S. Balagani and N. Y. I. O. T. O. WESTBURY, “Investigating cognitive rhythms as a new modality for continuous authentication,” Tech. rep. DTIC Document, Tech. Rep., 2013.
- [54] R. Banerjee, S. Feng, J. S. Kang, and Y. Choi, “Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, pp. 1469–1473. [Online]. Available: <http://www.aclweb.org/anthology/D14-1155>
- [55] Y. Sun, H. Ceker, and S. Upadhyaya, “Shared keystroke dataset for continuous authentication,” in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2016, pp. 1–6.
- [56] R. Shadman, A. A. Wahab, M. Manno, M. Lukaszewski, D. Hou, and F. Hus-sain, “Keystroke dynamics: Concepts, techniques, and applications,” *arXiv preprint arXiv:2303.04605*, 2023.

- [57] S. Krishnamoorthy, L. Rueda, S. Saad, and H. Elmiligi, “Identification of user behavioral biometrics for authentication using keystroke dynamics and machine learning,” in *Proceedings of the 2018 2nd international conference on biometric engineering and applications*, 2018, pp. 50–57.
- [58] K. Revett, F. Gorunescu, M. Gorunescu, M. Ene, S. Magalhaes, and H. Santos, “A machine learning approach to keystroke dynamics based user authentication,” *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 1, pp. 55–70, 2007.
- [59] A. Acien, A. Morales, J. V. Monaco, R. Vera-Rodriguez, and J. Fierrez, “Typenet: Deep learning keystroke biometrics,” 2021.