CLASSIFICATION OF TYPISTS WITH TYPING TELEMETRY

Hussein Tarek A Moneim Elguindi (s252091), Michael Xu (s251986)

ABSTRACT

We consider the classification problem of identifying a user typing on a keyboard using their keystroke dynamics and typing behavior. We evaluate and discuss our approach of using a Bidirectional Long Short Term Memory (Bi-LSTM) model to capture the time series dependencies of user keystroke behavior. Additionally, we introduce a new web-based system for measuring keystroke dynamics which we used for our data collection and inference.

Our source code is available at `https://github.com/HumaniseChisel/stylometry`.

1. INTRODUCTION

2. BACKGROUND & RELATED WORK

In this section, we introduce the use cases for behaviometrics and keystroke dynamics in authentication and security. We also discuss related works and their various implementations.

2.1. Behaviometrics & Keystroke Dynamics

Behaviometrics is a large field dealing with the measurement and analysis of patterns in all aspects of human behavior [1]. Narrowing down the field to human behavior in typing (i.e. key press and release timing, key hold timing, pauses, etc.) and analysis of typing rhythms is a subfield known as *keystroke dynamics* [2].

Keystroke dynamics has long been proposed as distinguishing markers between humans, as early as 1977 [3]. Since then, a number of papers have experimented with keystroke dynamics for the use of anomaly detection and authorization systems.

A reason for the popularity of keystroke dynamic based identification systems is due to the low cost of implementation and lack of special hardware required for measurement. These days, nearly all digital devices have keyboard input, whether digital or physical, and programming interfaces often expose methods for capturing high-granularity keyboard events. JavaScript exposes the KeyboardEvent Event [4], which we use in our implementation.

2.2. Related Work

The most recent review of keystroke dynamics in early 2025 analyzes different datasets, methods of implementation, and contrasts them. Early implementations of using keystroke dynamics for identification focus on statistical methods. However, [5] mentions that these approaches "necessitate manual feature extraction."

A popular company that commercializes this technology is TypingDNA [6], which raised $7 million in funding led by Google's Gradient Ventures in 2020 [7].

3. METHODS

3.1. Input Data Format

Raw keystroke dynamics were captured as a temporal sequence of discrete events. Each typing session consists of a series of key press and key release events, which were processed into a sequence of feature vectors X. For every keystroke k_i, we extracted a three-dimensional feature vector $v_i = [H_i, F_i, C_i]^T$:

Hold Time (H_i): The duration (in milliseconds) between the keydown and keyup events for a specific key. This metric captures the physical dwell time characteristic of a user's finger strength and typing style.

Flight Time (F_i): The latency (in milliseconds) between the keyup event of the previous key k_{i-1} and the keydown event of the current key k_i. This metric represents the speed of finger movement and hand transition patterns. In cases where k_i was pressed before k_{i-1} was released, we assigned a flight time of 0. That is, we do not allow for negative flight times.

Key Code (C_i): The ASCII integer representation of the character.

3.2. Network Architecture

To capture the temporal dependencies and rhythmic patterns inherent in user typing behavior, we employed a deep Recurrent Neural Network (RNN) architecture based on Bidirectional Long Short-Term Memory (Bi-LSTM) units. The model accepts an input sequence $X = \{x_1, x_2, ..., x_{43}\}$, where each timestep $x_t \in R^3$ represents a feature vector containing Key Hold Time, Flight Time (latency between keys), and the ASCII Key Code.

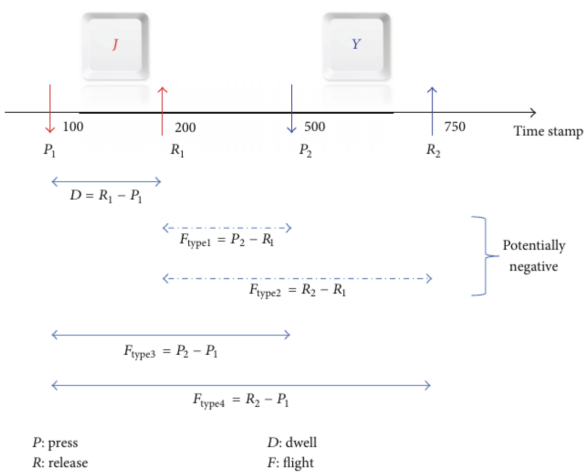

Fig. 1. Relevant keystroke events and features [5].

The core of the network consists of two stacked Bi-LSTM layers with a hidden state size of 64 units. The bidirectional configuration allows the network to process the input sequence in both forward and backward directions, effectively capturing context from both preceding and succeeding keystrokes within the window. The outputs of the forward and backward passes are concatenated, resulting in a 128-dimensional feature vector at each timestep.

From the sequence of outputs, we extract the hidden state corresponding to the final timestep (Many-to-One architecture) to serve as the global representation of the typing session. To prevent overfitting, a Dropout layer ($p = 0.3$) is applied to this representation. Finally, a fully connected (linear) layer maps the 128-dimensional vector to the output logits corresponding to the N unique user classes.

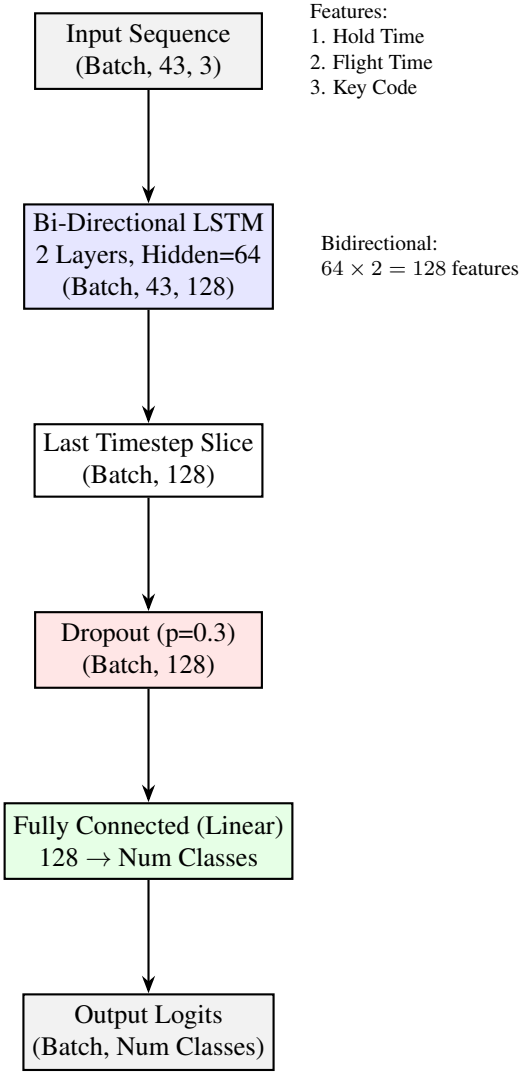

Fig. 2. Bi-directional LSTM neural network architecture.

3.3. Training Configuration

The dataset was partitioned using a stratified sampling strategy to ensure equal representation of all N user classes across subsets. The data was split into Training (70%), Validation (15%), and Test (15%) sets.

The model was trained using the Cross-Entropy Loss function, which is suitable for multi-class classification problems. We minimized this objective using the Adam optimizer, selected for its adaptive learning rate capabilities, with an initial learning rate of $\eta = 0.001$.

Training proceeded for 20 epochs with a batch size of 32 sequences. To monitor generalization performance and prevent overfitting, we evaluated the model on the held-out validation set at the end of each epoch.

The proposed system was implemented using the PyTorch deep learning framework. Data preprocessing pipelines, including vectorization and standardization, were built using NumPy and Scikit-learn. The training and inference routines utilized a DataLoader abstraction to handle batched input and shuffling. All experiments were conducted on a standard workstation environment, with tensor computations offloaded to a CUDA-enabled GPU where available to accelerate training throughput.

4. RESULTS

4.1. Performance Metrics

To evaluate the efficacy of the proposed Bi-LSTM model for user identification, we conducted experiments on a dataset containing keystroke dynamics from two distinct subjects (”Hussein” and ”Michael”). The model’s performance was assessed using a held-out test set comprising 30 sequences (15 samples per user), ensuring the evaluation reflected the model’s generalization capability on unseen data.

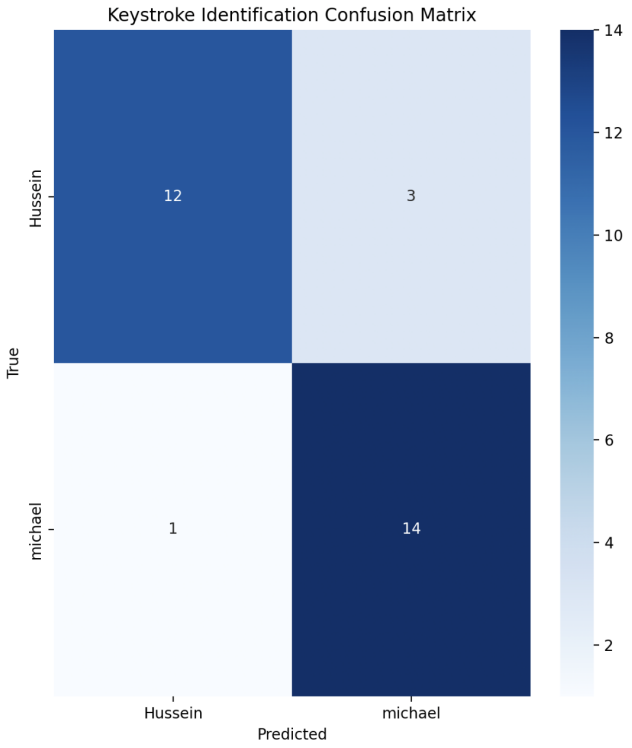

Fig. 3. Confusion matrix for two subjects.

4.2. Confusion Matrix

The classification performance is summarized in Figure 3. The matrix reveals high classification accuracy across both classes:

True Positives (Diagonal): The model correctly identified 12 instances of "Hussein" and 14 instances of "Michael".

Misclassifications (Off-Diagonal): 3 instances of "Hussein" were misclassified as "Michael". Only 1 instance of "Michael" was misclassified as "Hussein".

4.3. Overall Accuracy

After training, validation, and tuning our hyper-parameters, we tested the model on the test set we reserved. We observed a test set accuracy of 86.67% with F1 scores between 0.86 and 0.87. Given our limited dataset size (200 samples) and relatively non-computationally expensive training (we were able to train on a singular CPU machine), we are happy with our accuracy and inference results.

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

This study presented a deep learning framework for behavioral biometrics, specifically focusing on user identification through keystroke dynamics. By leveraging a Bidirectional Long Short-Term Memory (Bi-LSTM) architecture, we successfully modeled the complex temporal dependencies and rhythmic patterns inherent in typing behavior.

The proposed system processes raw keystroke events—specifically hold times, flight times, and key codes—transforming them into high-dimensional feature representations without the need for extensive manual feature engineering. Our experimental results on a controlled dataset demonstrated the model's robustness, achieving an overall classification accuracy of 86.67% on the test set. The distinction in performance between subjects (93.3% vs. 80.0%) highlights the model's sensitivity to individual typing variations, effectively capturing the unique "neural signature" of a user's motor control.

These findings validate the hypothesis that keystroke dynamics serve as a reliable, non-intrusive biometric identifier. The ability of the Bi-LSTM to aggregate context from both past and future keystrokes within a sequence proved critical in distinguishing between users with potentially similar typing speeds but distinct rhythmic cadences.

5.2. Future Work

While the current results are promising, several avenues for future research remain to enhance the system's scalability and real-world applicability:

- **Scalability to Larger Populations:** The current study focused on a binary classification task. Future work will evaluate the architecture's performance on datasets with a larger number of subjects to assess how well the feature space discriminates as the class count increases.
- **Continuous Authentication and Anomaly Detection:** Moving beyond closed-set identification, we aim to adapt the model for open-set authentication. This involves training the system to detect "impostors" (unseen users) by establishing a confidence threshold for acceptance, rather than simply classifying inputs into known categories.
- **Robustness to Environmental Factors:** Further investigation is needed to determine how the model generalizes across different keyboard types (e.g., mechanical vs. membrane) and user states (e.g., fatigue or stress), which are known to affect typing rhythm.

In summary, this work establishes a strong foundation for using deep recurrent neural networks in keystroke dynamics, offering a pathway toward secure, continuous, and passive authentication systems that require no specialized hardware.

6. REFERENCES

[1] Mordechai Nisenson, Ido Yariv, Ran El-Yaniv, and Ron Meir, "Towards behaviometric security systems: Learn-

ing to identify a typist," in *Knowledge Discovery in Databases: PKDD 2003*, Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, Eds., Berlin, Heidelberg, 2003, pp. 363–374, Springer Berlin Heidelberg.

[2] Kevin S. Killourhy and Roy A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, 2009, pp. 125–134.

[3] John Leggett, Glen Williams, Mark Usnick, and Mike Longnecker, "Dynamic identity verification via keystroke characteristics," *International Journal of Man-Machine Studies*, vol. 35, no. 6, pp. 859–870, 1991.

[4] MDN Web Docs, "Keyboardevent - web apis — mdn," 2025, Accessed: 2025-12-05.

[5] Soykat Amin and Cristian Di Iorio, "A review of several keystroke dynamics methods," 2025.

[6] TypingDNA, "TypingDNA - typing biometrics authentication api," `https://www.typingdna.com/`, 2025, Accessed: 2025-12-05.

[7] TechCrunch, "This startup is raising $7 million for a technology that can authenticate people based on their typing style," *TechCrunch*, Jan. 2020, Accessed: 2025-12-05.

DECLARATION OF USE OF GENERATIVE AI

This declaration **must** be filled out and included as the **final page** of the document. The questions apply to all parts of the work, including research, project writing, and coding.

- I/we have used generative AI tools: yes

If you answered *yes*, please complete the following sections. List the generative AI tools you have used:

- Claude Code
- Google Gemini

Describe how the tools were used:

What did you use the tool(s) for? The tools were used for writing an MVP of the web app. Claude Code and Google's Gemini helped us with the frontend Svelte code to collect training/validation/testing data and to perform inference. In addition, generative AI was used to generate to write parts of the API handler code to store training data to an S3 bucket.

At what stage(s) of the process did you use the tool(s)? The beginning stages for getting a quick MVP of the user interface. The maintenance stages for adding new features like the reset button or the UI for displaying keystroke logs.

How did you use or incorporate the generated output? In the case of Claude Code, the code is review manually by us, then prompted to write the changes. In the case of Google's Gemini, the changes were reviewed by us, then snippets were copied into our code editors.