Continuous Keystroke-Based User Authentication
Modeling Typing Dynamics with Hidden Markov Models

Ryan Lindberg
Halıcıoğlu Data Science Institute
University of California, San Diego
rylindberg@ucsd.edu

Tanner Berman
Department of Computer Science and Engineering
University of California, San Diego
tberman@ucsd.edu

Anthony Velikov
Department of Computer Science and Engineering
University of California, San Diego
avelikov@ucsd.edu

Audrey Meredith
Halıcıoğlu Data Science Institute
University of California, San Diego
ameredith@ucsd.edu

Preity Singh
Department of Computer Science and Engineering
University of California, San Diego
prs009@ucsd.edu

Abstract

Traditional authentication methods typically verify a user only once at login by checking whether the submitted password string matches the stored credential. In this project we ask whether we can add a *behavioral* layer of security at that same moment: given that the correct password text was entered, do the *keystroke timings* for that password look like they were produced by the claimed user or someone else? Using a public keystroke dynamics benchmark dataset, we construct sequences of timing features for a fixed strong password (`.tie5Roanl`) and model each user with a discrete Hidden Markov Model (HMM). The HMM is trained with Expectation Maximization to capture user specific typing modes and temporal structure in their keystrokes. We treat keystroke based identification as a classification problem: for each password typing sequence we compute its log likelihood under every user's HMM and assign it to the most likely user. Our results show that this method works but remains limited compared to standard classification models.

1 Problem Description

1.1 Motivation and Problem Statement

The traditional authentication approach verifies a user by checking if they entered the correct password but this has a critical problem: if someone steals or guesses the password, they gain full access. For our project, we address this problem by asking: Can we add an extra layer of verification when someone is entering their password by analyzing their keystroke dynamics? We monitor how a user types (timing patterns, inter-key latencies) and determine whether these observed keystrokes are consistent with the typing signature of the authorized user. The input is a sequence of keystroke events, and the output is a decision about whether those keystrokes are more consistent with the claimed user or with someone else. Our HMM could be used to identify if the user suddenly switches, for example, if the user switches from User A to User B the system can lock the screen. This is a good way of passive monitoring and supports applications like automatic profile switching. Although we focus on

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

a single fixed password in this project, we treat it as a proof of concept for a more general question: can we reliably identify users from their broader typing patterns, not just from one memorized string? If keystroke dynamics for a single password already carry enough signal to distinguish users, that suggests richer, free text typing behavior could support even stronger behavioral biometrics.

1.2 Relevance to Probabilistic Reasoning and Learning

Keystroke dynamics are inherently sequential: when a user types, we observe a sequence of key events over time that exhibit patterns. Certain key pairs tend to have characteristic timing, users shift between different modes of typing (steady flow, fast bursts, hesitations, or error corrections), and even for the same user and phrase, timing is not deterministic. A probabilistic model is natural because it represents uncertainty in timing via random variables, models latent structures such as unobserved typing modes, and provides principled ways to compute likelihoods and decision rules. Hidden Markov Models are particularly well suited because they model sequences $O_{1:T}$ of observations with an underlying sequence of latent states $S_{1:T}$, the Markov structure captures how users transition between different internal states over time, and the EM algorithm provides the learning and inference.

1.3 Project Scope and Goals

Goal 1: implement a data-processing pipeline that converts raw keystroke logs into discrete observation sequences suitable for HMMs.

Goal 2: train one discrete HMM per user and evaluate per repetition identification accuracy, i.e, how often we can correctly identify the user who typed a given password repetition.

Goal 3: explore how modeling choices (e.g., number of hidden states, number of discretization bins, number of users) effect performance.

We hypothesize that a properly tuned user-specific HMM will achieve substantially higher per-repetition identification accuracy than random guessing, and that its performance will be sensitive to model capacity (number of hidden states and timing bins).

2 Data Sourcing and Processing

2.1 Data Sources

We use a public keystroke dynamics benchmark dataset released by Carnegie Mellon University CyLab, originally introduced by Killourhy and Maxion [3] and distributed as the *Keystroke Dynamics Benchmark Data Set* [2]. We work with the `DSL-StrongPasswordData.csv` subset.

Each row in this dataset corresponds to one repetition of the same strong password (commonly reported as `.tie5Roanl`) typed by a human subject. The file contains around 50 subjects, up to 8 sessions per subject, and up to 50 repetitions of the password per session (400 repetitions per subject in total).

For each repetition, the dataset records **metadata** (`subject`, a unique subject ID such as `s002`; `sessionIndex`, an integer session number; and `rep`, an integer repetition index within the session), **per-key hold times** (`H.*` columns: key down to key up durations for each key in the password and the final Return key), **keydown–keydown latencies** (`DD.*.*` columns: elapsed time from keydown of key1 to keydown of key2), and **keyup–keydown latencies** (`UD.*.*` columns: elapsed time from keyup of key1 to keydown of key2; these values can be negative if key2 is pressed before key1 is released).

This dataset is well-suited to our problem because it provides many repetitions per user across multiple sessions, allowing us to study how stable keystroke patterns are over time. However, it also has important limitations: it focuses on a single memorized password typed in controlled conditions, so it may overestimate performance relative to real-world, noisy environments with free-text input and heterogeneous devices.

2.2 Preprocessing and Feature Engineering

To **encode** the data we discretized the keystroke timings into categorical bins, followed by a one-hot encoding of these bins. We chose to use all 31 **features** present besides session and repetition indices and chose a **80/10/10** ratio for the train/validation/test split. No data cleaning was necessary,

These processing choices are driven by the requirements of a discrete HMM. The raw keystroke data are continuous floating point timing values, but the HMM we use assumes that each observation O_t is a discrete symbol drawn from a categorical distribution. To make the data compatible with this emission model, we map each continuous timing value x_t into a bin index $o_t \in \{0, \dots, m-1\}$ with global quantile-based binning. The one-hot encoding used in our implementation is purely meant to make the `MultinomialHMM` API happy, because it expects count vectors rather than single digits. This representation preserves all the information in the discretized timings and exactly matches the discrete HMM formulation we use in our analysis.

3 Modeling and Inference

3.1 Probabilistic Model Specification

We model keystroke dynamics for each user with a discrete Hidden Markov Model using the `MultinomialHMM` class from the `hmmlearn` python library [1]. For a single password repetition of length T, we observe a sequence of discretized timing features

$$O_{1:T} = (O_1, \dots, O_T), \quad O_t \in \{0, \dots, m-1\},$$

where m is the number of quantile timing bins we split the data into.

Then we assume a corresponding sequence of latent "typing modes", represented by hidden typing states $S_{1:T}$ (e.g., "fast", "slow", "hesitating").

$$S_{1:T} = (S_1, \dots, S_T), \quad S_t \in \{1, \dots, n\}.$$

The HMM is parameterized by:

- The initial state distribution

$$\boldsymbol{\pi} = (\pi_1, \dots, \pi_n), \quad \pi_i = P(S_1 = i).$$

- The transition matrix

$$A = [a_{ij}]_{i,j=1}^{n}, \quad a_{ij} = P(S_{t+1} = j \mid S_t = i),$$

  which encodes how the typing mode evolves over time.

- The emission matrix

$$B = [b_{ik}]_{\substack{i=1,\dots,n \\ k=0,\dots,m-1}}, \quad b_{ik} = P(O_t = k \mid S_t = i),$$

  where each row $B_{i,:}$ is a categorical distribution over the m timing bins given hidden state i.

(see Appendix A.1, A.2 for the full EM derivation).

3.2 Model Assumptions and Dependencies

Our Hidden Markov Model for keystroke authentication relies on the following assumptions:

- **Assumption 1 (Markov Property)**: The hidden state at time t depends only on the hidden state at time $t-1$:

$$P(S_t \mid S_{1:t-1}) = P(S_t \mid S_{t-1})$$

  Justification for keystroke data: A user's current typing mode depends mostly on their most recent typing mode, not the entire history.

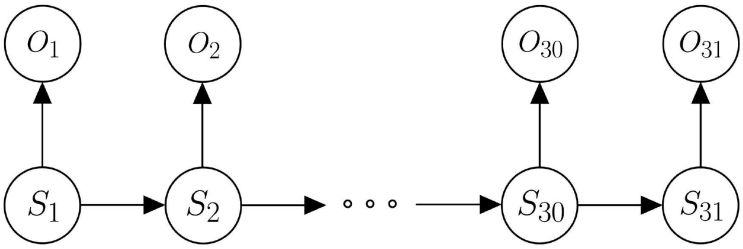

Figure 1: Graphical representation of our user-specific Hidden Markov Model. Hidden typing modes S_t form a first-order Markov chain, and each observed timing feature O_t is conditionally independent given S_t.

- **Assumption 2 (Conditional Independence of Observations)**: The observation at time t depends only on the current hidden state:

$$P(O_t \mid S_{1:T}, O_{1:t-1}) = P(O_t \mid S_t)$$

Justification for keystroke data: Given the user's current typing mode (the hidden state), the timing features for the current keystroke are independent of past observations. The hidden state captures all relevant information about the user's typing behavior at that moment.

- **Assumption 3 (Stationarity)**: The transition probabilities A and emission probabilities B remain constant throughout the sequence:

$$P(S_t = j \mid S_{t-1} = i) = a_{ij} \text{ for all } t$$
$$P(O_t = k \mid S_t = i) = b_{ik} \text{ for all } t$$

Justification for keystroke data: For a short password sequence, we assume the user's typing behavior is relatively stable. For longer sequences, this assumption may need relaxation.

- **Conditional Independence Structure**: Under these assumptions, the joint distribution factorizes as:

$$P(\mathbf{s}, \mathbf{o}) = P(S_1) \prod_{t=1}^{T-1} P(S_{t+1} \mid S_t) \prod_{t=1}^{T} P(O_t \mid S_t)$$
$$= \pi_{s_1} \prod_{t=1}^{T-1} a_{s_t, s_{t+1}} \prod_{t=1}^{T} b_{s_t, o_t}$$

This factorization allows for efficient inference via the forward-backward algorithm and learning via EM.

These assumptions relate to our dataset because keystroke dynamics naturally exhibit sequential structure with short-term temporal dependencies. The Markov assumption is reasonable for password-length sequences where typing modes change gradually. Conditional independence holds when the hidden state adequately captures the user's current typing behavior. The stationarity assumption may fail for very long typing sessions where fatigue or distraction could systematically alter transition and emission patterns, but is well-suited to our short password sequences.

3.3 Learning / Parameter Estimation

We train each user's HMM with the standard Baum Welch EM algorithm, maximizing the total log-likelihood of that user's discretized keystroke sequences. In the E-step we run the forward-backward algorithm to obtain posterior state and transition probabilities for each time step, and in the M-step we renormalize these expected counts to update the initial distribution $\boldsymbol{\pi}$, transition matrix A, and emission matrix B. Full derivations and explicit update equations are given in Appendix A.2.

3.4 Inference Algorithms

After training, each user u has an HMM

$$\theta^{(u)} = (\boldsymbol{\pi}^{(u)}, A^{(u)}, B^{(u)}).$$

Given a new keystroke sequence $\mathbf{o}$ from a single password repetition, we perform inference by computing its log-likelihood under each user model using the forward algorithm:

$$\ell^{(u)} = \log P(\mathbf{o} \mid \theta^{(u)}).$$

Assuming a uniform prior over users, the MAP estimate of the identity is

$$\hat{u} = \arg\max_{u \in \mathcal{U}} \ell^{(u)}.$$

Thus inference consists of evaluating the forward likelihood under each user's HMM and selecting the user with the highest log-likelihood. This is exactly the procedure we implement in our evaluation code.

4 Results and Discussion

4.1 Overall Identification Performance

We first tuned the HMM on the validation set by sweeping over the number of hidden states $K \in \{2, 3, 4, 5\}$ and the number of timing bins $m \in \{10, 15, 20\}$. Very small models ($K = 2$ or $m = 10$) underfit and achieved substantially worse validation accuracy, while larger models ($K = 5$ or $m = 20$ with more states) became numerically unstable and showed highly variable performance across random restarts. The best trade-off on validation was obtained with $K = 4$ hidden states and $m = 20$ timing bins, which we use for all reported test results.

Table 1 summarizes the closed-set identification accuracy on the 51-user test set. A random guess baseline, which picks a user uniformly at random, achieves about $1/51 \approx 1.96\%$ accuracy. Our discrete HMM achieves **57.6%** per-repetition identification accuracy, meaning that for a little more than half of the password repetitions the correct user's model assigns the highest likelihood among all 51 users.

Table 1: Identification accuracy on the 51-user test set.

Task Type	Metric	Result
Identification (Random Baseline)	Accuracy	$\approx 1.96\%$
Identification (Discrete HMM)	Accuracy	**57.6%**

To understand how this scales with the number of users, we also ran the same pipeline on a subset of 5 users. In that setting the HMM achieves around **75%** accuracy. The drop from 75% (5 users) to 57.6% (51 users) is consistent with the intuition that as we add more users, their typing patterns overlap more and the identification problem becomes harder.

4.2 Decision Confidence via Log-Likelihood Margins

Accuracy alone does not reveal how confidently the HMM separates genuine users from impostors. For each test repetition, we therefore computed the *log-likelihood margin* between the true user's model and the most competitive impostor model,

$$\text{margin} = \log p(\mathbf{o} \mid \theta^{(\text{true})}) - \max_{u \neq \text{true}} \log p(\mathbf{o} \mid \theta^{(u)}).$$

Positive margins correspond to correct identifications; negative margins correspond to misclassifications.

Figure 2 shows the distribution of these margins across the 51-user test set. Most mass lies close to zero, with relatively few repetitions in the strongly positive or strongly negative tails. This indicates that in many cases the true user's HMM is only slightly more likely than the best impostor model, which explains why overall accuracy is moderate rather than high. In other words, the model often finds multiple users whose typing patterns explain a given repetition almost equally well, so small fluctuations in the learned parameters can flip the decision. For deployment as a security mechanism, this suggests that the HMM would need to be combined with other signals or used as a soft score rather than a hard one-of-N classifier.

4.3 Qualitative Behavior of the Learned HMMs

Qualitatively, the learned HMMs capture broad aspects of a user's typing rhythm: transitions between hidden states tend to align with faster and slower segments of the password, and certain states are associated with consistently longer or shorter inter-key timings. This matches the intuition that people exhibit stable micro-patterns in how they speed up, slow down, or hesitate while typing.

At the same time, the model struggles to represent finer-grained structure. Each password repetition contains only 31 events, and timing values vary substantially across repetitions even for the same user. After discretization into 20 bins, many subtle differences collapse into the same bin, causing the emission distributions to blur together timing patterns that might otherwise be discriminative. As a result, the latent states do not map cleanly onto interpretable behaviors such as "hesitation at the dot" or "fast typing through the middle of the password"; instead, they behave more like coarse clusters over noisy timing features.

4.4 Convergence, Scalability, and Data Requirements

From an optimization standpoint, Baum Welch EM converged reliably for all users. Sequences are short and we work in log space, so numerical issues were minimal. Both training and inference scale roughly as $\mathcal{O}(K^2T)$ per sequence, and with $K = 4$ and $T = 31$ this cost is negligible, suggesting that the approach is computationally compatible with real-time authentication.

The limiting factor is not computation but data. Each user contributes only a few hundred repetitions of a single password, which is barely enough to estimate transition and emission probabilities for even a small HMM. The log-likelihood margins in Figure 2 suggest that many errors stem from genuine ambiguity rather than clear model failures: with more diverse data (e.g., free-text typing and more sessions per user), higher-capacity models or richer features might separate users more cleanly.

4.5 Limitations and Alternative Explanations

However, our approach is limited in several important ways. We train one independent HMM per user on a single fixed password, which simplifies the problem and does not capture free-text typing. As a result, the model only sees a very narrow slice of each user's behavior and cannot exploit richer language or context dependent patterns. With more diverse data and longer sequences, an HMM could potentially be used for user authentication on generalized free text typing rather than a fixed prompt. In addition, we rely on a discretized/binned timing values which discards fine grained timing information and may underfit the true complexity of human typing dynamics. Finally, although our model far outperformed a random guess, its real world application is questionable with an accuracy of 57.6% in the 51 user setting. Other classification models that make use of logistic regression or deep learning would likely perform better at this task, especially if trained on richer features.

There are also alternative explanations for our results. One possibility is that our discretization scheme and choice of K and m are suboptimal, and that a different representation (e.g., per-feature binning) would yield more separable likelihoods across users. Another is that the Baum Welch EM algorithm is getting stuck in local optima for some users due to limited data, so part of the observed error may be an optimization issue rather than a fundamental limit of keystroke dynamics. More broadly, it would be valuable to move from single password experiments such as this one to free text typing and study how user specific structure varies across content, devices, and sessions. In spite of these limitations, we believe that integrating keystroke based HMMs with other behavioral or contextual signals (e.g., login history, device fingerprints, or mouse dynamics) could yield a more reliable and continuous authentication system suitable for real world security.

5 Conclusion

In this project, we investigated whether keystroke dynamics for a single fixed password contain enough behavioral structure to support user identification. By training one discrete Hidden Markov Model per user and evaluating the likelihood of each keystroke sequence under every user's model, we showed that even a relatively simple probabilistic model can extract meaningful aspects of an individual's typing rhythm. The learned latent states correspond to "typing modes" and recur

across repetitions for the same user, indicating that users exhibit reproducible micro-patterns in their keystroke timing.

Our results, however, also reveal limitations. Identification accuracy drops as the number of users increases, and the log-likelihood margins show that many sequences remain ambiguous (explained nearly equally well by multiple users' models). We also find that performance is highly sensitive to design choices such as the number of hidden states and discretization bins: small to moderate settings achieve the best balance between expressiveness and overfitting, while larger models tend to become numerically unstable and fail to improve accuracy. Combined with the information loss introduced by discretizing continuous timing values, these factors constrain the reliability of the HMM in the 51 user setting.

Overall, our findings suggest that while HMM-based keystroke modeling is a promising component of behavioral authentication, it is insufficient on its own for robust multi-user identification. Extending this framework to richer data, such as free text typing, longer sequences, or additional behavioral aspects and exploring more expressive sequence models could lead to more reliable identification among users. Our results illustrate that typing behaviors do contain distinctive structure, but unlocking their full potential for security will require both richer datasets and more flexible models.

6 Reflections & Contributions

6.1 Advice for Future Students

Much better results could be achieved with traditional classification models such as linear regression, or perhaps with a deep learning model. Overall, our main takeaway is that sequence models like HMMs are interesting and conceptually aligned with keystroke dynamics, but they are not automatically the best-performing choice on this dataset. Future students should be prepared to iterate on both the model class and the feature representation, and to honestly report when a simpler approach outperforms the more sophisticated one.

6.2 Team Contributions and Reflections

Ryan Lindberg I contributed the initial code for the HMM model, the data preprocessing code, figures 1 & 2, as well as many sections in the report write up.

Tanner Berman I helped test and tune hyperparameters and co-wrote sections of the report including Modeling and Inference.

Anthony Velikov I expanded evaluation methods to include verification(now scraped for space) and helped write parts of the writeup, including conclusion and results and discussion.

Audrey Meredith I helped build our final HMM model by experimenting with different hyperparameters to maximize the accuracy of our HMM model as well as helped write sections of the writeup.

Preity Singh I co-wrote sections of the writeup, including the abstraction, motivation and problem statement, project scope and goals, model assumptions and dependencies, and results and discussion.

6.3 Use of Generative AI

We used generative AI tools for code debugging suggestions and LaTeX formatting help. All modeling decisions, experiments, and final text were critically reviewed and finalized by the authors.

References

[1] hmmlearn: Hidden markov models in python. `https://hmmlearn.readthedocs.io/`. Version 0.3.x, Accessed 2025-11-12.

[2] Kevin S. Killourhy and Roy A. Maxion. Keystroke dynamics benchmark data set. `https://www.cs.cmu.edu/~keystroke/` and Kaggle mirror: `https://www.kaggle.com/datasets/`

carnegiecylab/keystroke-dynamics-benchmark-data-set, 2009. Accessed 2025-11-10.

[3] Kevin S. Killourhy and Roy A. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–134, Estoril, Portugal, 2009. IEEE Computer Society.

A Additional Modeling Details

A.1 HMM Likelihood and Inference

Under the first-order Markov and conditional-independence assumptions, the joint distribution over hidden states and observations factorizes as

$$p(o_{1:T}, s_{1:T}) = p(S_1 = s_1)\, p(O_1 = o_1 \mid S_1 = s_1) \prod_{t=1}^{T-1} p(S_{t+1} = s_{t+1} \mid S_t = s_t)\, p(O_{t+1} = o_{t+1} \mid S_{t+1} = s_{t+1})$$
$$= \pi_{s_1}\, b_{s_1,o_1} \prod_{t=1}^{T-1} a_{s_t,s_{t+1}}\, b_{s_{t+1},o_{t+1}}. \tag{1}$$

The sequence likelihood is obtained by marginalizing out the latent states:

$$p(o_{1:T}) = \sum_{s_1,\ldots,s_T} p(o_{1:T}, s_{1:T}). \tag{2}$$

In practice, we compute this likelihood using the forward algorithm. Define the forward variables

$$\alpha_{it} = P(o_{1:t}, S_t = i), \quad i = 1, \ldots, n,\ t = 1, \ldots, T. \tag{3}$$

Satisfy the recursion

$$\alpha_{i1} = \pi_i\, b_{i,o_1}, \tag{4}$$

$$\alpha_{j,t+1} = \left(\sum_{i=1}^{n} \alpha_{it}\, a_{ij} \right) b_{j,o_{t+1}}, \quad t = 1, \ldots, T-1. \tag{5}$$

The likelihood of the observed sequence is then

$$p(o_{1:T}) = \sum_{i=1}^{n} \alpha_{iT}. \tag{6}$$

For posterior inference over the hidden states, we also define backward variables

$$\beta_{it} = P(o_{t+1:T} \mid S_t = i), \quad i = 1, \ldots, n,\ t = 1, \ldots, T, \tag{7}$$

which satisfy the standard backward recursion. Combining α_{it} and β_{it} yields smoothed state posteriors $P(S_t = i \mid o_{1:T}) \propto \alpha_{it}\beta_{it}$

A.2 EM Learning / Parameter Estimation

For a single user we observe R sequences of discretized keystroke timings:

$$\{\mathbf{o}^{(r)}\}_{r=1}^{R}, \qquad \mathbf{o}^{(r)} = (o_1^{(r)}, \ldots, o_{T_r}^{(r)}).$$

We seek HMM parameters

$$\theta = (\boldsymbol{\pi}, A, B)$$

that maximize the total log-likelihood:

$$\theta^* = \arg\max_{\theta} \sum_{r=1}^{R} \log P(\mathbf{o}^{(r)} \mid \theta).$$

Direct optimization is intractable, so we use the EM algorithm for HMMs (the Baum Welch algorithm).

E-step. For each sequence $\mathbf{o}^{(r)}$, we run the forward–backward algorithm to compute:

$$\alpha_{it}^{(r)} = P(o_{1:t}^{(r)},\, S_t = i), \qquad \beta_{it}^{(r)} = P(o_{t+1:T_r}^{(r)} \mid S_t = i).$$

From these we obtain:

- the posterior probability that the model is in state i at time t:

$$P(S_t = i \mid \mathbf{o}^{(r)}) = \frac{\alpha_{it}^{(r)}\, \beta_{it}^{(r)}}{\sum_j \alpha_{jt}^{(r)}\, \beta_{jt}^{(r)}}.$$

- and the posterior probability of making a transition $i \to j$ at time t:

$$P(S_t = i, S_{t+1} = j \mid \mathbf{o}^{(r)}) \propto \alpha_{it}^{(r)}\, a_{ij}\, b_j(o_{t+1}^{(r)})\, \beta_{j,t+1}^{(r)}.$$

These posteriors serve as the "expected counts" of state occupancies and transitions under the current parameters.

M-step. We update the parameters by normalizing these expected counts:

$$\pi_i \leftarrow \frac{1}{R} \sum_{r=1}^{R} P(S_1^{(r)} = i \mid \mathbf{o}^{(r)}),$$

$$a_{ij} \leftarrow \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r - 1} P(S_t = i, S_{t+1} = j \mid \mathbf{o}^{(r)})}{\sum_{r=1}^{R} \sum_{t=1}^{T_r - 1} P(S_t = i \mid \mathbf{o}^{(r)})},$$

$$b_{ik} \leftarrow \frac{\sum_{r=1}^{R} \sum_{t=1}^{T_r} \mathbf{1}\{o_t^{(r)} = k\}\, P(S_t = i \mid \mathbf{o}^{(r)})}{\sum_{r=1}^{R} \sum_{t=1}^{T_r} P(S_t = i \mid \mathbf{o}^{(r)})}.$$

These E–M steps repeat until convergence. In our implementation, we rely on the `MultinomialHMM` class from the `hmmlearn` [1] library to carry out these Baum Welch updates numerically for each user-specific model.

A.3 Evaluation Results Figure

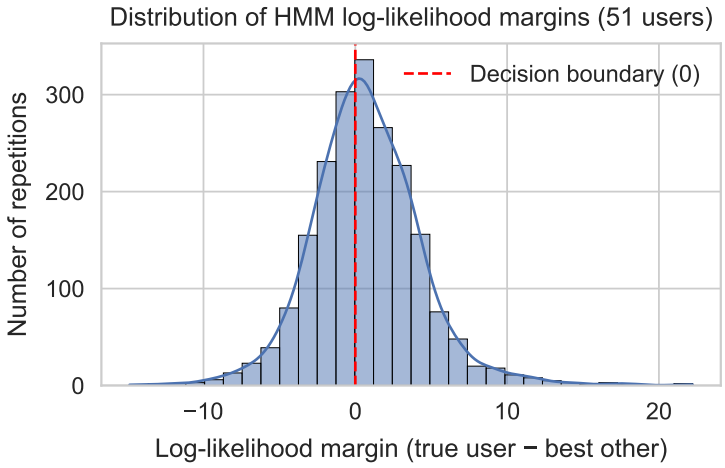

Figure 2: Distribution of per-repetition log-likelihood margins on the 51-user test set, where each margin is the difference between the true user's HMM log-likelihood and the highest impostor log-likelihood for a given password repetition. The bulk of the mass is concentrated near zero, indicating that for many repetitions the true and impostor models assign very similar probabilities, which aligns with the moderate overall identification accuracy.