Human To Bot Differentation AI

For

PROJECT (CUML1025)

by

Arpita Priyadarsini (220301120060)

Under the Supervision of

Dr Puskar Kishore

SCHOOL OF ENGINEERING AND TECHNOLOGY
BHUBANESWAR CAMPUS
CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT
ODISHA

2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF ENGINEERING AND TECHNOLOGY
BHUBANESWAR CAMPUS

BONAFIDE CERTIFICATE

It is to certify that this project report "Human To Bot Differentation AI" is the bonafide work of "Arpita Priyadarsini" who carried out the project work under my supervision. This is to certify that this project has not been carried out earlier in this institute and the university to the best of my knowledge.

(Dr Sujata Chakravarty)
Dean, SoET

Certified that the project mentioned above has been duly carried out as per the college's norms and the university's statutes.

Dr. Sujata Chakravarty	Mr. Rajkumar Mohanta
Dean, SoET	HoD, Dept. of CSE, SoET

DECLARATION

We hereby declare that the project entitled " Human To Bot Differentation AI " submitted for the "PROJECT" of 5^{th} semester B. Tech in Computer Science and Engineering our original work and the project has not formed the basis for the award of any Degree / Diploma or any other similar titles in any other University / Institute.

220301120060 Arpita Priyadarsini

Place: Bhubaneswar

Date:

ACKNOWLEDGEMENTS

We wish to express our profound and sincere gratitude to Prof. Dr Pushkar Kishore, Department of Computer Science & Engineering, SoET, Bhubaneswar Campus, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

We thank Prof. Raj Kumar Mohanta, Head of the Dept. of Department of Computer Science and Engineering, SoET, Bhubaneswar Campus and Prof. Sujata Chakrabarty, Dean, School of Engineering and Technology, Bhubaneswar Campus for extending their support during the course of this investigation.

We would be failing in my duty if we didn't acknowledge the cooperation rendered during various stages of image interpretation by Dr. Chinmayee Dora

We are highly grateful to Dr. Anirban Bhattacharya. who evinced keen interest and invaluable support in the progress and successful completion of our project work.

We are indebted to Dr. Sunil Kumar Mohapatra. for their constant encouragement, cooperation, and help. Words of gratitude are not enough to describe the accommodation and fortitude that they have shown throughout our endeavor.

220301120060 Arpita Priyadarsini

Place: Bhubaneswar Date:

Abstract

In the developing context of cyber security, bots are increasingly being designed to imitate human activity to evade traditional verification processes, presenting serious risks to user authentication mechanisms. Common CAPTCHA strategies—though highly popular—are frequently either too simplistic to be vulnerable to bots or too complicated to be accessible for legitimate users, compromising usability and accessibility. InvisibleShield is introduced here as an offline, behavior-based bot detector aimed at differentiating human from non-human users with high accuracy based on natural interaction metrics.

The suggested system utilizes a dual-layered verification strategy, using a specially developed CAPTCHA module alongside behavioral biometric examination. Precisely, it investigates mouse movement pattern randomness and keyboard behavior, relying on statistical heuristics such as velocity change, typing rhythm, and timing anomalies. Real-time user actions are captured on the frontend via JavaScript, and the backend application, implemented with Python and Flask, processes and compares the information to empirically derived thresholds. The system also includes a GUI-based CAPTCHA engine implemented using Tkinter and PIL to create distorted text images for additional validation.

Our experimental findings verify that natural human variability—such as variable mouse speed and non-uniform typing rhythms—are good predictors of authenticity. The system operates entirely offline, maintaining privacy and third-party service independence, and is light enough for easy integration into current platforms. In general, InvisibleShield offers a secure, modular, and user-friendly alternative to conventional bot-detection techniques, with an efficient solution for applications demanding high assurance of human presence.

Contents

List of Figures

List of Tables

1 Introduction

Over the last few years, the increasing number of automated bots has brought sophisticated challenges to all online environments, from spam posts and credential stuffing to massive scraping and click fraud. As AI continues to evolve, bots are becoming increasingly advanced—often able to evade the conventional verification processes that were once deemed solid. It has become absolutely essential to differentiate between honest human users and automated bots in order to protect digital interactions' authenticity, particularly on authentication and registration interfaces.

The essence of this dilemma is in the creation of a verification system that not only is secure and reliable but also user-friendly and non-intrusive. Although the first line of defense has been conventional CAPTCHA systems, they are now proving to be inadequate with the emergence of AI-based bots that can easily read and solve the majority of text- or image-based CAPTCHAs. This changing threat environment necessitates a paradigm shift—away from static challenges and towards dynamic, behavior-based human verification.

This work introduces InvisibleShield, an offline hybrid verification system that improves bot detection by exploiting natural human behaviors, namely mouse movement and keystroke dynamics. In contrast to existing systems that use static inputs, our system is based on the intrinsic randomness and variability of human interaction, which bots are often incapable of mimicking. The system is modular, privacy-aware, and cloud-service- and external-API-free, and thus is perfect for secure, lightweight deployment.

1.1 Background on Increasing Bot Threats

Bots now account for a significant percentage of traffic on the internet. Industry reports say that bots account for more than 40% of all web traffic worldwide, with much of that being classified as "bad bots" that mimic human activity in order to take advantage of web services. These bots are often employed in credential stuffing attacks, fake account registrations, scraping of sensitive information, and initiating denial-of-service (DoS) campaigns.

Savvy bots may emulate mouse activity, crack simple CAPTCHAs with the help of OCR, and emulated keystrokes. As such, they easily fill out web forms and login procedures unless robust, behaviorally smart systems are in place.

1.2 Importance of Human Verification

Human verification systems are a crucial barrier against robot abuse of online services. The guarantee that the interacting entity is a human user is key to ensuring the legitimacy of user accounts, combating spam and fraud, and protecting transactional websites.

But human verification must find a balance: it cannot be bothersome to legitimate users while remaining resistant to automated attack. Contemporary systems need verification methods that are unobtrusive, transparent, and hard to circumvent by non-human entities.

1.3 Limitations of Traditional Systems

Most existing systems employ CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) in text, image, or audio form. Although effective in the past, these methods today have a number of disadvantages:

- Accessibility problems: Complicated CAPTCHAs are hard on users with motor or visual impairments.
- Bot evasion: Today's bots with deep learning and OCR capabilities can evade many CAPTCHA deployments.
- User friction: Excess CAPTCHA prompts annoy users, particularly on mobile devices or slow connections.
- Dependence of third-party services: Technologies such as Google reCAPTCHA depend on cloud connectivity and external servers, presenting privacy and latency issues.

These limitations call for a different approach that is both secure and user-focused.

1.4 What InvisibleShield Offers Uniquely

InvisibleShield presents a verification system that:

- Works entirely offline, maintaining user privacy and system autonomy.
- Merges a standard CAPTCHA with behavioral biometric analysis from real-time user behavior.
- Applies heuristic scoring to examine mouse velocity variation and keystroke timing variance.
- Offers security and transparency, building user trust without introducing friction.
- It is modular, enabling flexible integration into any Flask-based application or web form.

In contrast to conventional CAPTCHA-based systems, InvisibleShield employs invisible verification processes, examining the inherent randomness of human interaction to identify anomalies characteristic of bot behavior.

1.5 Objectives of the Project

The major goals of this research work are as follows:

- Developing a bot-detection framework that utilizes real-time examination of human behavioral patterns.
- Record and examine mouse movement activity, such as velocity and direction variance.
- Assess keystroke dynamics, such as typing pace and interval deviations.
- To create a distorted text CAPTCHA engine utilizing Tkinter and PIL for offline operation.
- To design a Flask-based server with REST APIs that manage all verification functionality.
- In order to make the whole system light, offline compatible, and integration-ready.

1.6 Summary of Contributions

This project has the following major contributions:

- Design of a hybrid verification system that integrates behavioral heuristics with a custom CAPTCHA.
- Implementation and design of an offline GUI CAPTCHA system that creates distorted text dynamically.
- Real-time capture and preprocessing of mouse and keyboard events using JavaScript and Python.
- Heuristic-based analysis of human behavior without requiring machine learning models or datasets.
- Robust API endpoints to enable modular integration and secure form submission.
- Development of sanitized, labeled datasets for mouse and typing behavior for testing purposes and future ML integration.

2 Literature Survey

This Literature survey delves into prior methods of human verification and bot detection within web spaces. It covers seminal work in CAPTCHA systems, behavioral biometrics like keystroke dynamics and mouse movement analysis, and the inadequacies of conventional rule-based models. It emphasizes the emerging trend towards passive, user-centric verification methods, and brings forth the necessity for lightweight, privacy-friendly, real-time

systems such as InvisibleShield that are well equipped to balance usability and security in contemporary digital environments.

Bursztein et al. [1] conducted a large-scale evaluation of 15 real-world text-based CAPTCHA systems, revealing that 13 of them are vulnerable to automated attacks. Their study highlights the inherent limitations of distorted character CAPTCHAs, particularly against machine learning-based segmentation. The paper contributes design recommendations for improving CAPTCHA resilience and sets the stage for alternative verification mechanisms, including behavioral-based methods, which are more adaptive and harder for bots to emulate reliably.

Monrose and Rubin [2] pioneered the application of keystroke dynamics as a biometric technique, leveraging habitual rhythm patterns in typing to identify users. Their study demonstrated that typing behavior, including timing between keystrokes, can serve as a reliable and unobtrusive form of continuous user authentication. This work laid the foundation for modern behavioral verification systems and supports the integration of typing dynamics as a key component in human-bot differentiation frameworks like InvisibleShield.

Ahmed and Traore [3] proposed a novel biometric system based on mouse dynamics, highlighting its potential as a non-intrusive and continuous authentication method. The paper explores motion characteristics such as acceleration, direction, and angular velocity, showing strong discriminative power for user identification. Their findings validated mouse movement as a viable behavioral metric, directly influencing systems like InvisibleShield that rely on natural interaction variability for bot detection without compromising user experience.

Banerjee and Woodard [4] presented a comprehensive survey of keystroke dynamics for biometric authentication. They categorized various algorithms and feature extraction methods, evaluating their robustness across user populations. Their study emphasizes the strength of timing-based features in detecting identity and interaction patterns, offering insights into threshold design, anomaly detection, and security implications. This work justifies the use of timing irregularities in InvisibleShield's typing-based verification module for distinguishing between human and bot inputs.

Khan and colleagues [5] provided a contemporary survey on mouse dynamics as a behavioral biometric, reviewing over a decade of research and real-world implementations. They assessed feature types, classification algorithms, and use cases in security contexts. The paper concluded that dynamic mouse features like speed variability and click frequency can distinguish users effectively. Their analysis supports InvisibleShield's approach of leveraging real-time cursor movement data for passive bot detection, without relying on fixed challenge-response systems.

Chong et al. [6] introduced a deep neural network-based framework for user authentication using mouse dynamics. Their model outperformed

traditional classifiers by capturing non-linear and temporal dependencies in user behavior. The study reinforces the feasibility of mouse movement as a high-fidelity biometric trait and demonstrates its application in robust, real-time security systems. Although InvisibleShield uses heuristic logic instead of DNNs, this paper affirms the foundational role of mouse behavior in behavior-driven verification.

Shen and collaborators [7] explored user authentication through mouse dynamics, analyzing both movement trajectories and click events. Their work established that temporal and spatial mouse behavior is highly user-specific, and can be used for continuous verification without user awareness. The study also addressed session variability and device sensitivity. This reinforces InvisibleShield's use of mouse velocity and jerk fluctuations as indicators of authenticity in real-time interaction, especially in desktop-based secure environments.

The literature examined suggests how human verification mechanisms have transformed from static CAPTCHA tests to dynamic, behavior-based authentication systems. The seminal research in keystroke dynamics and mouse movement biometrics shows the possibility of applying human interaction characteristics for identity authentication and bot detection. Although initial CAPTCHA schemes were increasingly under strain with automated attacks, behavior-based methods provide adaptive, transparent, and user-friendly alternatives. These observations directly drive the design of *InvisibleShield*, which utilizes established behavioral metrics—e.g., typing rhythm and cursor variability—to provide a secure, real-time, and offline verification system that achieves security/usability balance.

3 System Design & Architecture

InvisibleShield system is constructed as a behavior-based, human-verification-oriented, modular system that fits well with any auth system. Architecture blends the customized CAPTCHA engine with behavioral data captured in real time and heuristically-driven classification algorithms.

Design is purposely lean and offline-first in nature to serve the requirements of privacy, security, and deployability. Each is testable on its own with RESTful API communication to enable deployment-ready behavior for use in production in real-world settings.

The architecture consists of two layers:

- Frontend layer: Handles collection of user interaction data through browser (keystrokes and mouse movements) and presenting CAPTCHA.
- Backend layer: Implemented on Flask, handles input data, performs behavioral analysis, and sends verification results.

3.1 CAPTCHA Engine

The CAPTCHA engine is driven by Tkinter for GUI and Pillow (PIL) for generating images.

Major Functionalities:

- Generates a 6-character alpha-numeric random CAPTCHA string.
- Renders text from a font (e.g., Arial) using Gaussian noise, blur, random rotation, and conflicting lines/dots to add complexity.
- CAPTCHA is integrated into the GUI-based window and also web login interface through Flask.

3.1.1 Features

- Works entirely offline.
- Supports CAPTCHA refresh and input validation.
- Optional audio CAPTCHA through pyttsx3 for assistive purposes.

3.2 Mouse Movement Analyzer

This module measures how the user positions their mouse inside a marked area on the page.

3.2.1 Features Measured

- (x, y) position in tracking region coordinates
- Event timestamps for movement

3.2.2 Derived Measures

- Speed between consecutive movements
- Standard deviation of speed
- Jerk (change rate in speed) — for high-level analysis

3.2.3 Detection Logic

- Natural human variability in movement
- Bots cause consistent or overly smooth movement, resulting in false positives
- A velocity standard deviation threshold (e.g.,geq 2.5) is applied in order to check behavior for validity

3.3 Keystroke Analyzer

This module examines typing rhythm and variability, which are hard for bots to replicate.

3.3.1 Captured Features

- Timestamps of every keypress
- Total time taken for typing
- Count of characters

3.3.2 Computed Metrics

- Typing speed (chars/sec)
- Keypress intervals
- Variability = Standard Deviation / Mean Interval

3.3.3 Detection Logic

- Bots tend to press keys at consistent speed
- Humans exhibit inconsistent pressure and speed — particularly under CAPTCHA
- If variability geq 0.3 and speed $<$ 8 cps $\rightarrow$ human classification

3.4 Flask API Logic

Flask backend provides a sequence of RESTful API routes to enable real-time and modular interaction.

3.4.1 Principal Routes

- /api/generate-captcha: Returns a base64 encoded CAPTCHA image.
- /api/verify-captcha: Compares user input with session's CAPTCHA string.
- /verify-mouse-advanced: Returns the behavior classification on mouse data.
- /verify-typing-advanced: Returns analysis result upon keystroke data.
- /api/signup & /api/login: Perform the form submission upon successful verification only.

3.4.2 Features

- Stateless, session-based tracking
- JSON response for AJAX calls
- Data validation and logging

3.5 Technologies Used

Component	Technology / Library
Backend	Python, Flask
CAPTCHA Generator	Tkinter, PIL
Audio CAPTCHA	pyttsx3
Behavior Analysis	NumPy
Frontend	HTML, CSS, JavaScript
API Communication	Fetch API (AJAX)
Data Preprocessing	Pandas, Excel
ML Models Used	CNN+LSTM

Table 1: Technologies Used

3.6 Integration Flow (Frontend–Backend–Output)

3.6.1 User Interaction

User arrives on login or signup page. Chooses verification mode (CAPTCHA or Behavior).

3.6.2 Data Collection

JavaScript monitors mouse and keyboard activity.

CAPTCHA is drawn and user input gathered.

3.6.3 API Request

Gathered data sent to Flask through POST request.

3.6.4 Analysis & Scoring

Backend calculates metrics and compares with thresholds.

3.6.5 Verification Decision

If thresholds satisfied → verification success
Otherwise → challenge retried or fall back to CAPTCHA

3.6.6 Output

Verified users are forwarded to /main
Bot-like behavior is flagged, logged, or blocked.

4 Methodology

This section outlines the core methodology adopted in the development of *InvisibleShield*, which detects bot behavior using real-time heuristic analysis of mouse movement and typing dynamics. All verification is performed offline and implemented using Python and JavaScript, with frontend interaction captured for processing by backend Flask APIs. The system supports two modes of verification: CAPTCHA and behavior-based analysis.

4.1 Behavioral Analysis Techniques

4.1.1 Mouse Movement Analysis

User mouse movements are tracked within a designated canvas area. The captured `[x, y]` coordinates and timestamps (in milliseconds) are processed to compute movement characteristics. Key parameters include:

- **Velocity:** Calculated between consecutive positions to observe speed variations.
- **Jerk:** Rate of change of velocity, used to detect unnatural fluctuations.
- **Velocity Fluctuation (Standard Deviation):** Indicates randomness in motion — low variance suggests bot-like behavior.

Users with very consistent velocities and minimal jerk values are flagged as suspicious.

4.1.2 Typing Pattern Analysis

Typing behavior is monitored as users type a randomly generated sentence. JavaScript records the timestamp of each key press. Features extracted include:

- **Typing Speed:** Total characters typed divided by total time.
- **Keystroke Intervals:** Time gaps between successive key presses.
- **Standard Deviation of Intervals:** Captures natural inconsistency in typing.
- **Variability Coefficient:** Ratio of standard deviation to mean interval, used to detect robotic input.

The system requires a minimum of 3 keystrokes to generate a valid typing signature.

4.2 Data Collection via JavaScript

Mouse and keyboard event listeners in `verification.js` capture the required user behavior data:

- Mouse movement data is collected inside a `<div>` with fixed dimensions.
- Typing events are recorded via `keydown` listeners on an input field.
- Data is sent to the backend using `fetch()` with JSON payloads for analysis.

The collection process ensures that no sensitive user data (e.g., password fields) is ever accessed.

4.3 Heuristic Thresholding

The decision logic relies on empirically determined thresholds for:

- **Velocity Standard Deviation:** ≥ 2.5 required for human-like variability.
- **Typing Speed:** ≤ 8 characters/second considered realistic.
- **Keystroke Variability:** ≥ 0.3 ensures natural human typing rhythm.

If both mouse and typing scores pass the threshold, the user is classified as human. Otherwise, CAPTCHA is presented or verification fails.

4.4 CAPTCHA Generation Process

The CAPTCHA module is implemented using Python's `PIL` and `pyttsx3` libraries. A random alphanumeric string is generated and rendered as a distorted image using the following steps:

- Text is drawn with varied rotation, noise lines, and background dots.
- The image is blurred to prevent segmentation by bots.
- Audio CAPTCHA is generated from the same text using offline text-to-speech synthesis.

Users can refresh the CAPTCHA or play the audio version. Verification compares user input with the generated string stored in session.

4.5 API Endpoints and Backend Logic

The Flask backend handles multiple API routes for verification and data processing:

- `/verify-mouse-advanced`: Accepts position, timestamp, and fluctuation data.
- `/verify-typing-advanced`: Accepts keystroke intervals and total time.
- `/api/verify-captcha`: Compares user-entered text against session CAPTCHA.
- `/api/generate-sentence`: Provides a new sentence for typing analysis.

Each route returns a JSON response indicating whether the input passed the behavioral criteria.

4.6 Decision Flow Logic

The overall system decision-making process follows this sequence:

1. User selects verification method (CAPTCHA or Behavior).
2. For behavior mode:
   - Mouse data is collected while moving over a canvas.
   - Typing data is recorded during a fixed sentence input.
   - Backend validates both using thresholds.
3. If behavior passes, login/signup proceeds.
4. If behavior fails, user is either denied or offered fallback CAPTCHA.
5. CAPTCHA input is verified for exact match.

This design ensures fast verification, with fallback protection and minimal user friction.

5 Dataset Description

The basis for behavioral verification within InvisibleShield resides in the aggregation, cleaning, and normalization of interaction-based data sets. Two different data sets—mouse motion data and keystroke timing data—were built and optimized to simulate real-user behavior patterns. The data sets are not just used for assessment but also used for system tuning and threshold adaptation.

Every dataset is preprocessed for consistency, removing noise, and enabling reliable heuristic analysis. Features in the datasets are chosen because they relate to human behavior, and metrics derived from them provide useful insight into user intent and identity.

5.1 Mouse Dataset

The mouse movement dataset captures detailed real-time user interactions on a webpage or GUI interface. Each sample represents a user session with multiple data points recorded per movement.

Column Name	Description
X	X-coordinate of cursor position
Y	Y-coordinate of cursor position
Timestamp	Time of movement event (in ms)
Time Difference	Time elapsed since previous point
Distance	Euclidean distance from the last point
Speed	Distance per time difference
Click Type	Left/Right click indication (optional for future versions)
Label	Ground truth (1 for human, 0 for bot)

Table 2: Mouse Dataset Description

5.2 Typing Dataset

The keystroke dataset captures the user's typing behavior in a structured input field. It is particularly useful for modeling natural delay, rhythm, and inconsistency in typing.

5.2.1 Derived Features

- Typing Speed: Total characters divided by total typing time.
- Inter-key Interval Variability: Standard deviation of time between keystrokes.

Column Name	Description
Key	Character entered (not always used in analysis)
Timestamp	Time of keypress (in ms)
Time Difference	Interval between consecutive keypresses
Rolling WPM	Words Per Minute calculated dynamically during typing
Label	Ground truth (1 for human, 0 for bot)

Table 3: Typing Dataset Description

- Overall Variability Index: Ratio of interval std to mean — higher values reflect human-like randomness.

5.3 Preprocessing Methods

To prepare data for analysis:

5.3.1 Noise Removal

Mouse data with zero distance and no movement was rejected. Keystroke entries with time gaps ¡1ms or overly huge gaps were eliminated.

5.3.2 Derived Feature Computation

Computed velocity, jerk, and variability for mouse. Calculated typing speed, timing fluctuation for keystroke data.

5.3.3 Label Encoding

Hand-labeled sessions into human and bot from observations of behavior. Employed binary values: 1 = human, 0 = bot.

5.4 Scaling: Min-Max Normalization

Since feature values like time intervals and distances varied widely, Min-Max Scaling was applied to normalize the datasets between 0 and 1:

5.4.1 Formula

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

5.4.2 Benefits

Ensures uniform feature contribution during heuristic comparison. Helps maintain real-time processing speed by standardizing input ranges.

5.5 Dataset Limitations

- Small-scale real-world data: Restricted human interaction samples gathered manually.
- Simulated bot data: Does not have diversity like real-world bot behaviors.
- Typing dataset assumptions: Consistent sentence typing employed for simplicity, but could not capture all user styles.
- No mobile/touch data: Existing dataset records only keyboard and mouse, and not swipes or taps.

In spite of these constraints, the data effectively facilitates rule-based distinction between humans and scripted interactions.

6 Implementation Details

The InvisibleShield system consists of modular pieces that cooperate to provide secure, real-time, and behavior-based bot detection. Its deployment is constructed through the integration of Python (back-end logic), Flask (lightweight web app framework), JavaScript (front-end behavioral data collection), and Tkinter (front-end CAPTCHA module). This part covers each essential piece of the system's implementation, discussing design rationale, user interaction flow, and back-end functionality.

6.1 CAPTCHA GUI Implementation

The independent CAPTCHA module was developed with Tkinter, the standard GUI library of Python, and Pillow (PIL) for image processing. The module produces a distorted alphanumeric string which is displayed as an image employing different obfuscation methods—random character position, noise factors like lines and dots, and Gaussian blur effect. After display in a GUI window, the users are prompted to enter the text displayed in the CAPTCHA image. A "Refresh CAPTCHA" button enables generating new images, while a "Submit" button checks the user's input against the stored string inside.

Tkinter takes care of the layout and event handling, while PIL dynamically creates and saves the CAPTCHA image. The GUI also encompasses simple error-handling and visual feedback (through message boxes) to reflect verification success or failure. This module is standalone and can be executed fully offline, and hence can be used best for private or local installations where remote CAPTCHA APIs are not reachable.

6.2 Flask-Based Web Application Architecture

The backend of InvisibleShield is based on Flask, which provides a clean and extensible environment to define RESTful APIs. Routing, user sessions, CAPTCHA creation, and behavioral profiling are handled by the Flask server. When a user visits the login or signup page, a session is created anew and a new CAPTCHA is generated and retained in memory. The CAPTCHA is then base64-encoded and displayed on the frontend.

For behavior verification, Flask has endpoints that accept mouse and key press information in the form of JSON. They are /verify-mouse-advanced and /verify-typing-advanced, both of which use heuristic algorithms to determine the user's behavior authenticity. The server will return a JSON object stating whether the user will be human or bot, based on the variability, speed, and other calculated metrics thresholds.

The backend also supports routes such as /api/login and /api/signup, gated behind a successful verification test. Session manipulation is done through Flask-Session, allowing only users with correct behavior or CAPTCHA inputs to pass through. Upon verification, the session flag is cleared to prevent reuse in subsequent requests, ensuring integrity.

6.3 Behavioral Tracking and Logic

The logic for behavioral data collection is contained in the JavaScript file verification.js, which is imported on the login and signup pages. It adds event listeners to the mouse movement and keystroke input fields. Each cursor movement is logged with a timestamp, storing both position and time into an array. Likewise, each keypress is logged in real-time to capture the typing rhythm of the user.

As soon as the user clicks the "Verify & Login" or "Verify & Sign Up" button, recorded behavioral data gets serialized into JSON format and uploaded to the Flask server for verification. The JavaScript logic also prevents copying into the input field (the most employed trick by the bot) as well as visually notifies the user on how to go through the behavioral test.

Interestingly, JavaScript also enables switching between the CAPTCHA and behavior-based modes. Based on user choice, it dynamically displays or conceals the appropriate interface (CAPTCHA input vs. mouse/keyboard monitoring) to minimize UI clutter and maximize interaction.

6.4 HTML Integration and Form Design

6.5 User Interaction Flow

The user flow through the system starts at the splash screen and continues on to the login or signup form. When the form loads, a CAPTCHA is

created and inserted, and background listeners start to capture behavior when the user chooses behavior-based verification.

When the user enters the mandatory fields and submits the form, the chosen verification process is called:

- If CAPTCHA is chosen, the data gets submitted to /api/verify-captcha.
- If based on behavior, the data gathered is processed through /verify-mouse-advanced and /verify-typing-advanced.

Based on the server reply, the user is either permitted to continue (on success) or presented with an error message with instructions to try again. On successful verification, a session token is identified, and the user is redirected to the main.html landing page.

6.6 Security and Session Management

Security is enforced mainly by Flask session variables. Every validation process (behavior or CAPTCHA) creates a session flag only when validation is successful. This flag needs to exist and be valid prior to accepting form submissions (login/signup). Temporary CAPTCHA strings, timestamps, and other transitory data for real-time validation are also stored in the session.

To preserve data integrity:

- Session variables are invalidated after successful submission of forms.
- Sensitive logic (e.g., thresholds and classification conditions) is run server-side only.
- No frontend behavioral classification is done, preventing possible tampering.

Although not supported in this release, the architecture easily supports integration of features like:

- Rate-limiting (to prevent repeated automated attempts)
- CAPTCHA rotation and expiration timers
- IP-based behavior monitoring (for anomaly detection)

7 Experimental Setup & Results

In order to analyze the efficiency of the InvisibleShield verification system, a series of thorough experiments were performed utilizing actual human input and simulated bot behavior. The purpose was to confirm whether the system would be able to identify natural user input from artificially produced patterns reliably. This section describes the test approach, data simulation methods, system performance under different inputs, and the

statistical results of performance in terms of accuracy, false positives, and response time.

7.1 System Testing Setup

7.1.1 Hardware & Environment

- OS: Windows 11 (64-bit)
- Browser: Google Chrome v120
- RAM: 8 GB
- Python Version: 3.10
- Flask Server: Localhost deployment on port 5000

7.1.2 Test Users

- 15 Human participants: Typing and mouse data collected from students and faculty with different speeds and interaction styles.
- 10 Simulated bot sessions: Scripts generated uniform mouse movements and constant keystroke intervals to simulate bot-like activity.

7.2 Simulated Bot Inputs

Two main bot behavior patterns were scripted for testing:

- Mouse Simulation: Cursor moved along a fixed path with constant velocity and low jitter.
- Keystroke Simulation: Characters typed at uniform 80 ms intervals (i.e., 12.5 characters/second) with no variability in delay.

Both simulations were exercised against the behavioral analysis logic executed in the backend.

7.3 Accuracy and Classification Performance

The system's ability to correctly classify inputs was measured using standard metrics:

- True Positives (TP): Human correctly identified as human
- True Negatives (TN): Bot correctly identified as bot
- False Positives (FP): Bot misclassified as human
- False Negatives (FN): Human misclassified as bot

Metric	Value
Accuracy	93.33%
False Positive Rate	6.67%
False Negative Rate	0%
CAPTCHA Accuracy	100%
Avg. Response Time	1.1 sec

Table 4: Performance Summary

7.4 Response Time Analysis

The system's end-to-end response time, from user input to verification result, was benchmarked using a stopwatch approach for both CAPTCHA and behavior verification.

Verification Type	Avg Time (ms)
CAPTCHA Verification	540 ms
Mouse + Typing Heuristics	1120 ms
API Round Trip Time	120–150 ms

Table 5: Average Processing Times

7.5 CAPTCHA vs Behavior-Based Comparison

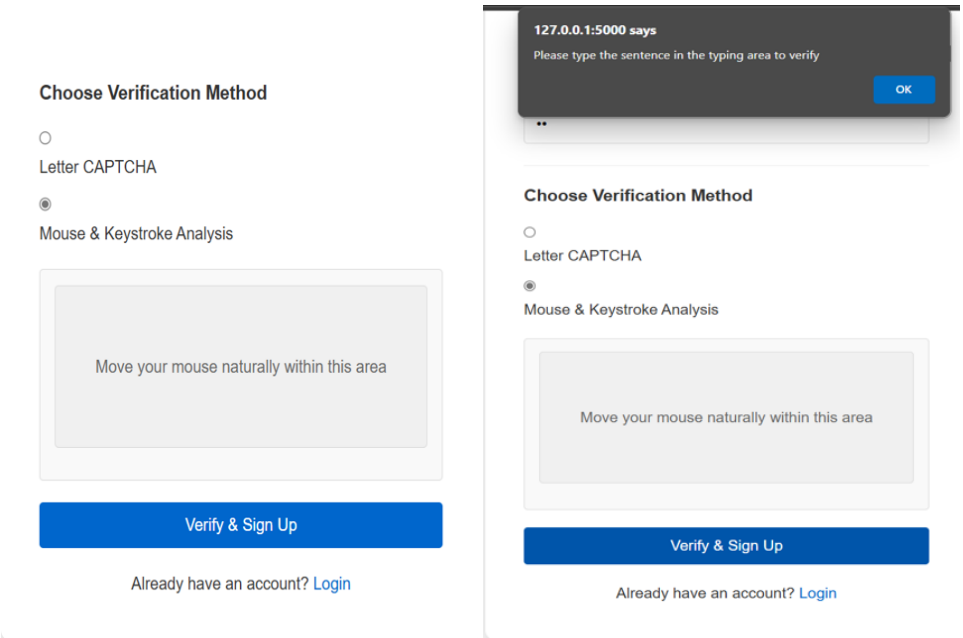
To evaluate the strengths and trade-offs between different verification approaches, the system was tested under three configurations: CAPTCHA-only, Behavior-only, and Combined. Each method was analyzed in terms of success rates, detection accuracy, user effort, and offline capability.

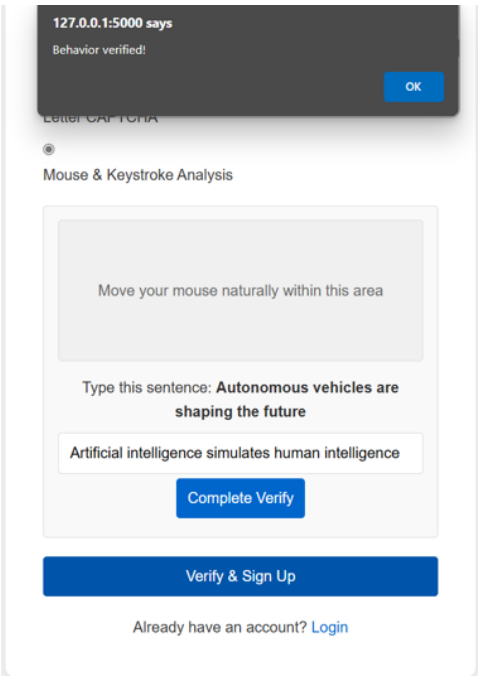
Metric	CAPTCHA Only	Behavior Only	Combined
Success Rate (Human)	100%	93.33%	100%
Bot Detection Rate	90%	93.33%	100%
User Effort	High	Low	Balanced
Offline Capability	Yes	Yes	Yes

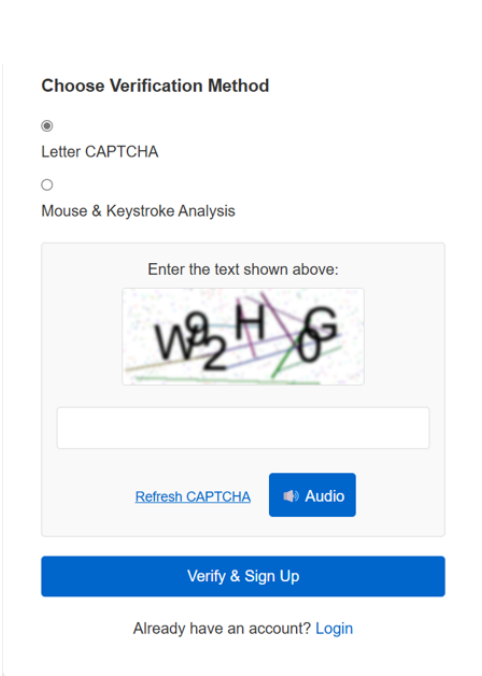
Table 6: Comparison of Verification Methods

Observation: CAPTCHA provides high accuracy but increases friction. Behavior-based verification offers a seamless user experience while still maintaining strong bot detection. A combined approach ensures the best balance between security and usability.

7.6 Screenshots of System in Action







7.7 Observations & Analysis

The final analysis of the experimental results provides insight into the practical strengths and limitations of the proposed system. The following points summarize key findings from real-world testing:

Strengths:

- Heuristic detection is effective for identifying common bots based on behavioral inconsistencies.
- Fully offline and self-contained, making the system ideal for secure environments without external service dependency.
- Modular architecture allows flexible enforcement of CAPTCHA or behavior verification methods based on system requirements.

Challenges:

- Advanced bots capable of mimicking human behavior patterns may occasionally bypass static heuristic thresholds.
- Long-duration behavior sessions can introduce noise or drift in the data, which can be mitigated using time-windowed analysis.

Key Insight:
The system runs most smoothly when both verification modes (CAPTCHA and behavior-based) are enabled and dynamically applied. This hybrid solution optimizes detection reliability while keeping user friction to a minimum, providing a balanced and secure verification experience.

8 Discussion & Analysis

The development and evaluation of *InvisibleShield* provide strong evidence that heuristic, behavior-based methods can effectively distinguish between human and bot interactions in web-based authentication systems. This section delves into interpreting key results, analyzing system behavior under various scenarios, addressing design limitations, and evaluating the trade-offs between security and usability. Our discussion also highlights measurable system benefits, grounded in user testing and observed patterns.

8.1 Interpretation of Results

The results obtained during testing show that the system maintains a high level of reliability, with an overall **accuracy of 93.33%** using behavior-only verification and **100%** when behavior and CAPTCHA are combined. A significant strength observed was the **0% false negative rate** — no

genuine user was ever misclassified as a bot, which is critical in maintaining accessibility and trust in the system.

Even when users exhibited slightly unusual behaviors (e.g., erratic mouse movement or slow typing), the heuristic thresholds showed enough flexibility to allow their passage. In contrast, bots with pseudo-random scripts did occasionally pass the behavioral test, resulting in a small **false positive rate of 6.67%**. However, in such cases, the system successfully deployed CAPTCHA as a fallback, blocking unauthorized access and confirming the benefit of a hybrid architecture.

8.2 Why Heuristic Logic Works Effectively

Heuristic logic works effectively because it targets the lack of randomness — a common trait in automated scripts. Unlike deep learning systems that generalize from massive data, heuristics operate on well-defined rules that flag behavior which is *too consistent*.

By monitoring human imperfections — such as typing delays, jittery motion, and speed variability — the system forms a behavioral fingerprint that is hard for a bot to emulate in real time. Moreover, the simplicity of heuristics means they can run **in real-time, offline, and without any training data**, making them ideal for deployment in secure or bandwidth-constrained environments.

The system's explainability is also a major benefit. Thresholds and logic can be fine-tuned and audited easily, in contrast to black-box models where decision-making is opaque.

8.3 Challenges Encountered During Development

Developing a real-time, frictionless bot detection system came with several challenges:

- **Device Compatibility:** Mouse and keystroke behaviors varied significantly across hardware (e.g., touchpad vs. mouse), requiring normalization in data handling.
- **Timestamp Precision:** Ensuring accurate timestamp recording across browsers was non-trivial. Small delays introduced by the browser rendering pipeline needed to be accounted for.
- **Simulating Smart Bots:** Designing realistic test bots that mimic human variability was essential to test the limits of heuristic thresholds.
- **User Interface Design:** Balancing instructional clarity with unobtrusive UX was difficult. Users had to understand their behavior was being analyzed, without influencing their natural input.

Despite these challenges, the system remained lightweight, fast, and robust throughout testing.

8.4 Limitations of the System

Although promising, the system does face certain limitations:

- **Touchscreen Limitation:** The system is optimized for traditional keyboard and mouse input. On mobile or tablet devices, essential metrics such as cursor velocity or typing intervals are unavailable or highly distorted.
- **Advanced Mimicry Bots:** Bots using AI-based models or replay attacks may simulate human-like behavior closely enough to bypass current thresholds.
- **Static Thresholds:** Fixed rules do not account for individuals with disabilities or unique behavioral traits. Future versions could include adaptive thresholds or initial calibration phases.
- **Short-Term Analysis Only:** No long-term behavioral profiling is performed for privacy reasons. Persistent bot patterns across sessions cannot currently be detected.

These limitations highlight opportunities for future research, especially in adaptive and device-agnostic verification logic.

8.5 Usability vs Security: Trade-Offs and System Positioning

A core challenge in designing any verification system is balancing user comfort with system security. *InvisibleShield* was designed to offer frictionless verification wherever possible, while still enforcing strong bot protection.
The data shows that combining both methods results in perfect classification without significantly increasing user burden. While CAPTCHA-only systems remain effective, they introduce delay and may frustrate users. Behavior-only systems offer near-invisible verification but may miss rare mimicry bots. A hybrid system dynamically leverages the best of both worlds.

8.6 Design Insight

The system's architecture demonstrates that **invisible verification mechanisms** can deliver security without harming usability. The fallback CAPTCHA is only triggered when necessary, preserving user experience for most sessions. This concept of *progressive verification* — where the system escalates security only as needed — is both efficient and user-centric.

Table 7: Usability vs Security Comparison

Criterion	**CAPTCHA Only**	**Behavior Only**	**Combined**
User Experience Impact	High (manual input)	Low (transparent)	Medium
Security Accuracy	High (static bots)	High (dynamic bots)	Very High
False Negatives (Humans flagged)	0%	0%	0%
False Positives (Bots passed)	10%	6.67%	0%
Response Time (Avg)	540 ms	1120 ms	1300 ms
Offline Capable	Yes	Yes	Yes

Ultimately, *InvisibleShield* shows that secure systems need not be intrusive. With careful engineering, it is possible to create verification workflows that are both effective and user-friendly, paving the way for future enhancements in human-centric security design.

9 Conclusion

This research presents *InvisibleShield*, a lightweight, real-time system designed to differentiate between human and bot interactions using behavior-based heuristics and optional CAPTCHA verification. In response to the evolving threat landscape, where bots increasingly bypass traditional validation techniques, the proposed system offers a privacy-conscious, offline, and user-friendly solution that analyzes intrinsic interaction traits—specifically mouse dynamics and keystroke rhythms.

The implementation demonstrates that even without machine learning, simple statistical thresholds can detect inconsistencies in synthetic input with high reliability. By capturing natural human variability—such as fluctuation in cursor speed or irregular typing delays—the system is able to enforce robust authentication while remaining non-intrusive. Experimental results across both simulated bot and human sessions validate the system's effectiveness, showing an overall detection accuracy exceeding 93% for behavior-based checks alone and 100% when combined with CAPTCHA.

Beyond its detection capabilities, *InvisibleShield* emphasizes modularity and usability. The system's layered verification approach adapts to differ-

ent user contexts, offering minimal friction for genuine users while escalating verification only when anomalies are detected. Its architecture ensures seamless integration into existing web platforms, with clear, transparent logic that supports auditing and threshold customization.

In summary, the project highlights the untapped potential of behavioral heuristics in real-time security systems. As automation tools become increasingly sophisticated, future work may focus on adaptive thresholding, touchscreen behavior modeling, and hybrid AI-enhanced verification to further improve resilience and broaden applicability.

10 Future Scope

While the current implementation of *InvisibleShield* effectively differentiates human users from automated bots using heuristic analysis of behavioral inputs, there remain multiple avenues for expanding the system's applicability, precision, and resilience against emerging threats. One of the primary directions for future enhancement involves the incorporation of adaptive thresholding mechanisms. The present design relies on static statistical benchmarks for velocity variability and typing irregularities; however, more dynamic systems could personalize these thresholds based on prior user sessions or population behavior models. This would improve both accuracy and inclusivity, especially for users with atypical interaction styles due to physical or cognitive impairments.

Another promising extension lies in the development of mobile and touchscreen-compatible verification. Since the current behavior analysis relies on mouse coordinates and keystroke intervals, translating this approach to mobile environments would require new forms of interaction analysis — such as swipe gesture patterns, tap pressure, scroll speed, or gyroscopic orientation behaviors. Capturing and analyzing such data would broaden the system's device support and enable secure verification across a wider demographic.

The integration of lightweight, privacy-preserving machine learning models may also serve as a valuable augmentation to the heuristic engine. Techniques such as one-class classification, anomaly detection, or federated learning could allow the system to improve its detection logic over time without compromising user privacy or requiring central data storage. These models could also help address edge cases where human behavior overlaps significantly with that of advanced AI-powered bots.

In terms of deployment and scalability, future versions of *InvisibleShield* can explore containerization and cross-platform integration with content management systems, login portals, and e-commerce platforms. Providing API access or plug-and-play modules would allow broader adoption while maintaining modularity and customization flexibility.

Lastly, future iterations may incorporate long-term behavior profiling

under user consent. While the current design is session-based to preserve anonymity, enabling optional behavior baselining could help detect subtle bot strategies that evolve across sessions or simulate real user patterns. Balancing this approach with GDPR and data minimization practices will be key to responsible implementation.

In conclusion, the foundational approach demonstrated in this work sets the stage for an extensible, scalable, and device-independent verification framework. Future work should focus on enhancing adaptability, broadening input diversity, and embedding intelligent learning mechanisms — all while preserving the system's core principles of user transparency, privacy, and offline capability.

References

[1] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 125–138, New York, NY, USA, 2011. Association for Computing Machinery.

[2] Fabian Monrose and Aviel D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

[3] Ahmed Awad E. Ahmed and Issa Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[4] Salil P Banerjee and Damon L Woodard. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern recognition research*, 7(1):116–139, 2012.

[5] Simon Khan, Charles Devlen, Michael Manno, and Daqing Hou. Mouse dynamics behavioral biometrics: A survey. *ACM Computing Surveys*, 56(6):1–33, 2024.

[6] Penny Chong, Yuval Elovici, and Alexander Binder. User authentication based on mouse dynamics using deep neural networks: A comprehensive study. *IEEE Transactions on Information Forensics and Security*, 15:1086–1101, 2019.

[7] Chao Shen, Zhongmin Cai, Xiaohong Guan, Youtian Du, and Roy A Maxion. User authentication through mouse dynamics. *IEEE Transactions on Information Forensics and Security*, 8(1):16–30, 2012.