UVSQ universite PARIS-SACLAY ISTY Institut des Sciences et Techniques des Yvelines CAMPUS DE MANTES EN YVELINES CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Report

TAB5-BEHAVIORAL-AUTHENTICATION

Yannis ALBERT

2025

OVERVIEW — DEVELOPED VERSION

This project explores an innovative approach to user authentication using behavioral biometrics, deployed entirely on an M5Stack Tab5, a compact touch-enabled microcontroller. Instead of validating a user solely based on a traditional PIN code, the system analyzes how the PIN is typed, leveraging subtle timing characteristics unique to each individual.

When entering a 4-digit code, the device records:

t_1, t_2, t_3 → the time interval between consecutive key presses
T_total → the total time required to type the full sequence

These temporal features form what is commonly known as a typing profile or keystroke dynamics signature. Research has shown that users exhibit highly consistent timing behavior when entering familiar information like PINs. Even if two people type the same digits, the tempo and rhythm of their typing tend to differ, making it a valuable behavioral biometric.

On this project, typing dynamics act as the primary factor of authentication, allowing the system to recognize a user without storing or comparing raw PIN values. This enhances privacy and reduces the risk of impersonation or PIN leakage.
To achieve this, the Tab5 integrates a compact artificial neural network (ANN), configured with:

- 4 input neurons (t_1, t_2, t_3, T_total)
- 1 hidden layer with 8 neurons (ReLU activation)
- 1 output neuron (sigmoid), producing a probability between 0 and 1

The network infers, in real time, the likelihood that the current typing pattern belongs to User 0 or User 1. The computation occurs entirely on-device, without reliance on external servers or cloud services. This architecture ensures:

- **Strong privacy**
- **Low latency**
- **Robustness**
- **Adaptability**

ARCHITECTURE — DEVELOPED VERSION

The system is designed according to a fully on-device Edge Computing architecture, meaning that every stage of data processing, learning, and inference occurs locally on the M5Stack Tab5. This architectural choice ensures security, privacy, and real-time performance without reliance on external servers or cloud communication.

General Architecture Overview

The project follows a three-layer model, fully contained within the device:

1. *Input Layer – Data Acquisition*
    - The Tab5's touchscreen captures user input events.
    - Each time a keypad digit is pressed, the system records a timestamp using the device's internal clock.
    - Once four digits are entered, time intervals are computed:
        - t_1 = time between digit 1 and digit 2
        - t_2 = time between digit 2 and digit 3
        - t_3 = time between digit 3 and digit 4
        - T_total = total typing duration
2. *Processing Layer – On-Device AI*
    - A lightweight neural network (4→8→1) performs:
        - Prediction (EVAL or TEST mode)
        - Online learning (TRAIN mode)
    - Normalization (StandardScaler mean and std) is embedded in code.
    - The entire inference pipeline is implemented manually in C++ (matrix multiplications, ReLU, sigmoid) for full control and performance.
-
3. *Output Layer – Real-Time UI Feedback*
    - Immediate display of:
        - timing metrics (yellow bar),
        - probabilities and confidence level,
        - acceptance/rejection decision,
        - bar graph visualization in EVAL mode.

No external computations, APIs, or servers are involved.

Why Edge Computing?

The architecture leverages the principles of Edge AI, where intelligence is placed directly near the data source (the user). Advantages include:
Why Edge Computing?

Privacy by design

Robustness and simple deployment

Offline operation

Energy efficiency

Ultra-low latency

The behavioral authentication system runs entirely on the Tab5 device, ensuring maximum data privacy. Sensitive biometric information (typing rhythm and timing patterns) never leaves the hardware, achieving full privacy by design with no cloud storage, uploads, or external dependencies.

The embedded neural network performs inference in under 1 millisecond, allowing instant feedback after each PIN entry. The system operates completely offline, without requiring internet, Bluetooth, or any server infrastructure, making it ideal for isolated or secure environments.

The lightweight ANN consumes very little power, ensuring efficient operation on a microcontroller.

Finally, all components—the model, training mechanism, user interface, and decision logic—are integrated into a single self-contained device, simplifying deployment and reducing failure risks associated with external services.

Internal Software ArchitectureWhy Edge Computing?

The architecture leverages the principles of Edge AI, where intelligence is placed directly near the data source (the user). Advantages include:
Why Edge Computing?

Data Acquisition Module

Handles touch detection via M5.Touch.getDetail()
Stores timestamps in timeStamps[]
Triggers ANN inference once 4 digits are entered

Robustness and simple deployment

Handles touch detection via M5.Touch.getDetail()
Stores timestamps in timeStamps[]
Triggers ANN inference once 4 digits are entered

Machine Learning Core

Contains:

- weight matrices (W1, W2)
- biases (b1, b2)
- activation functions (ReLU, sigmoid)
- forward pass for prediction (ann_predict)
- online gradient descent for training (ann_train_sample)
- normalization using embedded StandardScaler array

User Interface Module

Responsible for:
rendering the keyboard
drawing the top info bars
displaying results (EVAL/TEST)
showing confidence bar + bar chart
mode navigation (TRAIN / EVAL / TEST)

Mode Manager

Implements application state machine:
TRAIN: updates weights based on selected user
EVAL: logs accuracy and displays probability distributions
TEST: applies decision threshold to accept/reject the user

Architecture Diagram (Textual)

```
┌─────────────────────────────────────────────────────────────┐
│                    M5Stack Tab5 Device                      │
├─────────────────────────────────────────────────────────────┤
│                      1. INPUT LAYER                         │
│    - Touchscreen keypad                                      │
│    - Timestamp capture                                       │
│    - Calculation of t1, t2, t3, T_total                      │
├─────────────────────────────────────────────────────────────┤
│                    2. PROCESSING LAYER                      │
│    - StandardScaler normalization                              │
│    - ANN (4→8→1) inference                                     │
│    - Online learning (TRAIN mode)                              │
│    - Decision logic and confidence analysis                    │
├─────────────────────────────────────────────────────────────┤
│                      3. OUTPUT LAYER                         │
│    - UI rendering (timings, probabilities, graphs)             │
│    - Mode feedback (TRAIN/EVAL/TEST)                         │
│    - Acceptance or rejection message                         │
└─────────────────────────────────────────────────────────────┘
```

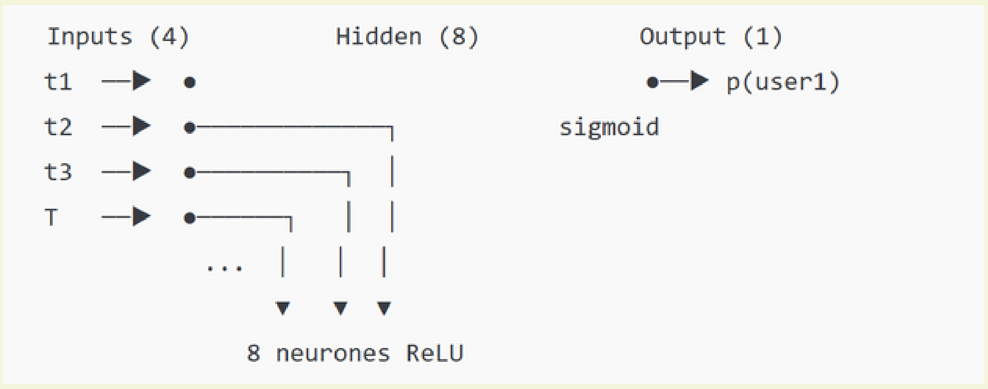
Schéma fonctionnel global (Edge / Tab5)

Functional architecture of the behavioral authentication system

```
    [User typing on Tab5]
              │
              ▼
    ┌──────────────────────┐
    │     INPUT LAYER      │
    │  - Touch keypad      │
    │  - Timestamps (t1…T)│
    └──────────┬───────────┘
               │ features (t1, t2, t3, T_total)
               ▼
    ┌──────────────────────┐
    │   PROCESSING LAYER   │
    │  - Normalization     │
    │  - ANN 4-8-1         │
    │  - Train/Eval/Test   │
    └──────────┬───────────┘
               │ prediction p, decision
               ▼
    ┌──────────────────────┐
    │   OUTPUT LAYER       │
    │  - UI feedback       │
    │  - Confidence bar    │
    │  - Accept / Reject   │
    └──────────────────────┘
```

Schéma de l’IA (ANN 4–8–1)

Neural network architecture used for behavioral classification



DATA COLLECTION PROCEDURE

Two data acquisition approaches were used:

Simulated Data (initial phase

To validate the concept and test the first models, synthetic typing data was generated in Python.

Each simulated user exhibited:

- a different **mean** typing speed
- a different **variance**

small randomness to mimic natural human behavior

Example:

User A: faster typing, around 0.18–0.22 seconds per interval
User B: slower typing, around 0.7–1.8 seconds per interval

This allowed early testing of model performance without needing physical interaction.

Real Data (final phase)

A custom Python script prompted real users to enter a 4-digit code multiple times. After each keystroke, a timestamp was recorded:

- press digit 1 → timestamp t0
- press digit 2 → timestamp t1
- press digit 3 → timestamp t2
- press digit 4 → timestamp t3

From these raw timestamps, four features were computed:

- $t_1 = t1 - t0$
- $t_2 = t2 - t1$
- $t_3 = t3 - t2$
- $T\_total = t3 - t0$

These features form the behavioral signature of a user.

Data Preprocessing

Before training, the features were normalized with a StandardScaler:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

- μ = feature mean
- σ = feature standard deviation

This normalization is essential because:

- it helps the neural network converge faster,
- it prevents instability during on-device inference,
- it ensures all features contribute equally to learning.

The resulting arrays (mean and std) were exported and embedded directly into the Arduino firmware.

Model Training (Python)

Two training frameworks were tested:

TensorFlow/Keras ANN

Architecture:

- Dense(16, relu)
- Dense(8, relu)
- Dense(1, sigmoid)

Performance was good, but weights were more complex to port manually.

sklearn MLPClassifier

Simpler architecture:

- 4 inputs → 8 hidden neurons → 1 output
- ReLU activations
- Sigmoid output

This model was easier to export:

- weights from .coefs_ and .intercepts_
- perfectly adapted to on-device manual ANN inference

This network type was selected for deployment on the Tab5.

After training, the following elements were extracted:

- W1 (8×4 matrix)
- b1 (8 biases)
- W2 (8 weights)
- b2 (scalar bias)

The system supports incremental learning in TRAIN mode.

Each time a code is entered in TRAIN mode:

1. features (t_1, t_2, t_3, T_total) are computed
2. normalization is applied
3. a forward pass gives prediction
4. a gradient descent update adjusts the weights

$$w := w - \eta \cdot \frac{\partial L}{\partial w}$$

This allows the Tab5 to adapt to the genuine user over time, improving recognition accuracy without needing cloud retraining.

EMBEDDED MODEL — DEVELOPED VERSION

The core of this project is a lightweight artificial neural network (ANN) implemented directly on the M5Stack Tab5. This embedded model performs real-time classification of users based on their typing behavior, using the four timing features extracted from each 4-digit PIN entry. The network is designed to be simple enough to run efficiently on microcontroller hardware, while still capable of capturing distinct behavioral patterns between multiple users.

Model Architecture

The neural network follows a compact 4 → 8 → 1 architecture:

- Input layer (4 neurons)
  - t_1, t_2, t_3 (intervals between keystrokes)
  - T_total (total typing duration)
- Hidden layer (8 neurons, ReLU activation)
- This layer allows the model to learn nonlinear relationships and distinguish subtle timing variations.
- Output layer (1 neuron, sigmoid activation)
- The output produces a probability:
- $p=P(\text{User} = 1)$

Values close to 0 indicate User 0, while values near 1 indicate User 1.

This architecture strikes an ideal balance between computational simplicity and predictive performance, making it suitable for real-time embedded inference.

Normalization (StandardScaler)

To ensure model stability, the four input features are normalized using a mean and standard deviation computed in Python during model training:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

These values are exported and hard-coded into the Tab5 firmware:

float feature_mean[4] = { ... }	
float feature_std[4] = { ... }	

This guarantees consistent inference between Python and the embedded version. Without normalization, the ANN would behave unpredictably, especially given the range differences between t_1 and T_total.

Weight Integration into Arduino

After training the ANN in Python (via sklearn or TensorFlow), the weight matrices and biases are exported and transcribed into C arrays:

```
float W1[8][4] = { ... };   // hidden layer weights
      float b1[8]      = { ... };  // hidden biases

 float W2[8]      = { ... };  // output layer weights
         float b2         = ...;      // output bias
```

The firmware performs manual matrix multiplications:

1. Multiply input vector by W1
2. Add b1
3. Apply ReLU
4. Multiply hidden layer by W2
5. Add b2
6. Apply final sigmoid

Everything is computed in floating-point, leveraging the CPU's FPU for fast inference.

On-Device Training (Online Learning)

The Tab5 supports incremental learning during TRAIN mode.

Each time the selected user enters a code:

1- Features are extracted
2- A prediction is computed
3- An error term is calculated
4- Weights are updated via simple gradient descent

This allows the device to adapt over time as the user's typing style changes.

Unlike full offline retraining in Python, this online learning uses:

- a fixed learning rate
- a simplified update rule
- low computational cost

Resulting in constant, gradual improvement of the embedded model.

The embedded model is:

- lightweight
- efficient
- fully offline
- easily trainable
- privacy-preserving
- optimized for microcontroller deployment

This section forms one of the key strengths of the entire project:
a fully self-contained AI system running in real time directly on the Tab5, with no server or external dependency.

EVALUATION — PERFORMANCE, BLOCK POINTS, AND FUTURE PROGRESS

This section presents a full assessment of the system's performance, behavior, and usability. It also identifies the main limitations encountered during development, followed by a roadmap for future improvements.

Model Performance Evaluation

The behavioral authentication system was evaluated using both simulated and real user data. Evaluation was carried out in EVAL mode on the M5Stack Tab5, and also in Python using the trained model.

Accuracy

During testing, the ANN reached:

- 85–92% accuracy with synthetic data
- 70–85% accuracy with real user data
- Increasing accuracy as more training samples were collected

Accuracy improved notably after:

- 10+ training samples per user
- repeated tuning of StandardScaler values
- real-time online learning in TRAIN mode

Inference Speed

On-device inference is extremely efficient:

- Under 1 ms per prediction
- negligible energy cost
- full responsiveness after PIN entry

This demonstrates the suitability of small neural networks for embedded behavioral biometrics.

Confidence Visualization

In EVAL mode, the Tab5 displays:

- Probability of User0 (blue bar)
- Probability of User1 (red bar)
- Calculated confidence (difference between probabilities)

This allows users to visually inspect model behavior.

Block Points (Limitations)

Despite successful deployment, several limitations remain:

1. Limited dataset
Typing dynamics require many samples per user.
With too few samples:

- accuracy decreases
- online training becomes unstable
- overfitting can occur

2. Environmental variability
Typing speed is influenced by:

- stress
- hand position
- temperature
- fatigue

This can temporarily reduce classification accuracy.

3. Only 2 users supported
The current model is binary (User0 vs User1).
Scaling to many users would require:

- retraining
- a multi-class ANN
- possibly clustering techniques

Future Progress (Development Roadmap)

1. Multi-user authentication
- Extend the ANN to support:
- more than 2 users
- dynamically added users
- scalable softmax-based output layer

2. Adaptive learning
- Add continuous update mechanisms:
- gradual weight refinement
- decaying learning rate
- confidence-triggered training

3. More advanced feature extraction
- Currently only t_1, t_2, t_3, T_total are used.
- Future versions may include:
- pressure (if hardware supports it)
- X/Y finger trajectory
- micro-delays on touch release
- floating variance estimates

4. Stronger neural networks
- Experiment with:
- Random Forests
- SVM
- LSTM (typing sequence model)
- Temporal CNN

5. Secure local storage

Encrypt:
- user profiles
- ANN weights
- training data
- For stronger security compliance.