

Lightweight, Autonomous Security Agents for Operator Workstations: An Explainable XGBoost-Driven Authentication Policy

Erik Pál

University of Debrecen

Róbert Lakatos

`lakatos.robert@inf.unideb.hu`

University of Debrecen

András Hajdu

University of Debrecen

Research Article

Keywords: continuous authentication, keystroke dynamics, autonomous security agent, explainable AI, XGBoost, SHAP edge computing, human-in-the-loop, energy efficiency, green AI

Posted Date: October 20th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11430757/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Lightweight, Autonomous Security Agents for Operator Workstations: An Explainable XGBoost-Driven Authentication Policy

Erik Pál¹, Róbert Lakatos^{1*}, András Hajdu¹

¹Department of Data Science and Visualization, Faculty of Informatics,
University of Debrecen, Debrecen, Hungary.

*Corresponding author(s). E-mail(s): lakatos.robert@inf.unideb.hu;
Contributing authors: pal.erik@inf.unideb.hu;
hajdu.andras@inf.unideb.hu;

Abstract

Background: Protecting edge workstations requires lightweight alternatives to energy-intensive deep learning. We introduce an autonomous security agent using an XGBoost keystroke-dynamics cognitive module. It monitors typing streams, assesses identity, and executes graded security policies with SHAP-explained, human-in-the-loop resolution.

Result: On a 13,251-keystroke benchmark (5 users), the module achieves 99.8% accuracy and a 0.0017% Equal Error Rate (reinterpreted as False Intervention Rate). Reaction time (63 ms) and peak memory (20.52 MB) substantially outperform the LSTM-based TypeNet baseline (180 ms, 145.90 MB); training requires 1.02 s versus 87.85 s. Under an equal-power assumption, this means about 65% less compute energy per decision and about 99% less per on-workstation adaptation. The agent organization adds further savings: it runs locally and event-driven, so explanations and incident reports are produced only on intervention, graded responses avoid needless lockouts, and retraining stays on the workstation.

Conclusion: This Green AI-aligned design enables mass edge deployment with precise, explainable, and provable (PEP-AI) human-in-the-loop security.

Keywords: continuous authentication, keystroke dynamics, autonomous security agent, explainable AI, XGBoost, SHAP edge computing, human-in-the-loop, energy efficiency, green AI

1 Background

Data has become a critical resource, and its protection is a primary concern for organizations and individuals alike [1]. Breach statistics consistently show that weak or reused passwords and the absence of multi-factor authentication remain dominant attack vectors [2], and the consequences of a breach are particularly severe in industrial and governmental environments, where they range from financial losses to operational disruption [3]. Electrical power grids are an extreme case of this problem. Their supervisory control and data acquisition (SCADA) and other operational technology (OT) systems are distributed, long-lived, and increasingly interconnected with information technology networks [4, 5]. The cyber attack on the Ukrainian power grid demonstrated that an adversary who controls an operator workstation can issue legitimate-looking commands to the physical process [6]. Authentication that occurs once at login cannot detect such a takeover of an already-open session. What is required instead is a verification that continues after login, and this is the role of the agent proposed in this article. A keystroke-based agent covers the typed part of such an interaction, for example commands, credentials or configuration entries typed by the intruder. The problem is, however, not specific to the energy sector: in any organization whose operator workstations can act on valuable systems or data, a hijacked or shared open session is equally dangerous. The mechanism presented in this article is therefore a general-purpose protection system; smart grids are the setting in which its constraints (distributed, long-lived, resource-limited, safety-critical) are most demanding, and in which its energy and explainability benefits matter most.

Continuous authentication addresses this gap by verifying the identity of the user throughout the session without explicit user action [7]. Keystroke dynamics, the individual rhythm and timing with which a person types, is an attractive modality for this purpose, since it requires no additional hardware and is collected passively from the normal work of the operator [8–10]. In our previous conference work [11], we showed that a gradient-boosted decision-tree model (XGBoost [12]) trained on engineered keystroke features identifies the enrolled user with 99.8% accuracy, while being considerably faster and smaller than the LSTM-based TypeNet model [13]. In that work, however, the model was a passive mathematical function that maps an input window to an output label; it did not decide what should happen when the typist is not the enrolled user.

The present article moves the focus from prediction to autonomous action. Following the agent view of artificial intelligence [14, 15], we treat the XGBoost model not as the end result, but as the *cognitive*, belief-forming module of an autonomous software agent. Because critical infrastructures such as energy systems, and organizations in general, consist of many distributed workstations, their protection can be distributed as well: instead of streaming raw behavioral data to a central, cloud-hosted deep learning service, each operator workstation hosts its own lightweight edge agent [16]. The agents act independently of each other and report to a common human supervisor, so together they form a distributed security layer. Each agent runs a *sensory–cognitive–action* cycle. It senses the environment through a custom keylogger, performs cognitive processing with the XGBoost tree ensemble, and acts by intervening in access management through a graded policy of challenges, alerts and lockouts. The balance between

sustainability and security is provided by this autonomy: a small tree ensemble that decides locally needs neither a GPU nor a permanent connection, and it scores each window in a single inference step. A fast decision does not, however, mean that the agent acts on every single score. Before it intervenes, the policy deliberately waits for two to three consecutive suspicious windows; this confirmation buffer accumulates evidence over time and trades a short delay for a more reliable decision. The agent approach can further improve both energy efficiency and explainability. The agent works event-driven and locally, so the costly work (explanation, incident reporting, interruption of the operator) is performed only when the policy decides to intervene, and raw behavioral data never has to be transferred; only decisions and the timing statistics needed for supervision are reported. The graded policy avoids needless escalation, and because the agent acts on an explicit belief and an explicit threshold, every intervention can be justified to the human with the evidence behind it.

These design choices also determine how the agent has to be evaluated. The protection of a system and the energy consumed by that protection are two sides of the same sustainability question, which is most visible in energy systems. On the one hand, the take-over of an operator workstation can endanger, in the case of a power grid, the security of supply and resilience [6]. On the other hand, the security function is itself a computing workload that runs on every operator workstation, so its compute effort per decision, memory, model storage, retraining effort and data traffic add up across the fleet, which is why the Green AI agenda asks for efficiency to be reported next to accuracy [17]. We therefore explain every result of this article along four energy-related dimensions: (i) the compute effort of one decision (response time, CPU utilization); (ii) the footprint of holding, distributing and adapting the model (peak memory, model size, training time); (iii) the amount of costly work (explanation, incident reporting, interruption of the operator) that is triggered by the error characteristics of the model; and (iv) the data that has to leave the workstation. Electrical energy was not measured directly. The indicators above are used as proxies, the assumption behind this is stated in Section 3.10, and its consequences are listed among the limitations.

Autonomy in a security-critical environment, such as a power-system control room, is, however, only acceptable if the agent’s decisions can be trusted, understood and verified. We therefore design the agent around the PEP-AI requirements: its decisions are *precise* (grounded in the measured error characteristics of the model), *explainable* (every intervention carries a SHAP-based justification [18, 19] addressed to the human operator) and *provable* (every intervention, its explanation, the thresholds in force and the human response are logged for audit). High-risk decisions remain with a human, in a human-in-the-loop (HITL) arrangement [20, 21].

The contributions of this article are the following:

1. A formal agent architecture for keystroke-based continuous authentication, in which the engineered keystroke features define the agent’s state space and a decision policy $a_t = \pi(S_t, f_{XGB}(X_t))$ maps the agent’s security state and the XGBoost belief to an action (Section 3).

2. A graded, explainable intervention policy (allow, challenge, alert, critical) whose thresholds are derived from the ROC curve and the Equal Error Rate of the cognitive module, and whose escalation depends on the human response (Section 3.5).
3. A SHAP-based explanation channel that presents the agent’s reasoning to the operator and to a supervisor, together with a privacy-preserving reporting design in which keystrokes never leave the edge device (Sections 3.7 and 3.8).
4. An energy-oriented framework (Section 3.10) and a reinterpretation of the accuracy, latency, memory, model-size and training-time results of the cognitive module as agent-level decision and energy indicators, compared with the TypeNet LSTM baseline (Section 4).

The remainder of the article is organized as follows. Section 2 reviews keystroke dynamics, agent-based edge security and explainable AI. Section 3 describes the agent architecture, its sensory, cognitive and action modules, the decision policy and the energy-oriented framework. Section 4 reports the results and explains each of them in relation to energy use, Section 5 discusses human-machine collaboration and the limitations of autonomy, and Section 6 concludes the article.

2 Related work

This section reviews the theoretical and technological foundations that inform the design of our autonomous security agent. First, we examine the application of keystroke dynamics and machine learning to continuous authentication, establishing why gradient-boosted trees offer a highly accurate yet resource-efficient alternative to deep sequence models. Second, we discuss autonomous agents and edge computing to contextualize the strict operational and environmental constraints of critical infrastructures. This background demonstrates why effective security mechanisms must operate locally and align with Green AI principles. Finally, we explore explainable AI (XAI) and human-in-the-loop (HITL) automation. Integrating these domains is essential to ensure that an autonomous system can justify its interventions to human operators, fulfilling the core requirements for precise, explainable, and provable (PEP-AI) security.

2.1 Keystroke dynamics for continuous authentication

Keystroke dynamics has been studied as a biometric identifier since the observation that inter-key latencies are characteristic of the individual [8, 9]. Classical statistical detectors struggle with the high intra-user variability of typing [22], which motivated the use of machine learning and deep learning [10]. TypeNet [13] is a representative deep model that uses LSTM units trained with a triplet loss and reports an EER of 2.2% on public free-text datasets. Deep sequence models are, however, not the only competitive option. Gradient-boosted trees have repeatedly been found competitive on tabular keystroke features, both for fixed-text [23] and free-text [24] authentication, and distance-based feature sets such as DEFT [25] further improve free-text performance. XGBoost [12, 26] is well-suited to this setting because of its robustness on heterogeneous tabular data and its low computational cost at inference time.

2.2 Autonomous agents and edge security

A keystroke classifier alone, however, only predicts; acting on its predictions requires an agent. An agent is an entity that perceives its environment through sensors and acts upon it through actuators in pursuit of its goals [14]. Multi-agent systems (MAS) decompose a task into cooperating autonomous agents, which matches the physically distributed structure of systems with many workstations [15]. In the present design, the agents do not communicate with each other and are coordinated only through a common supervisor. The setting in which such agents operate imposes its own requirements. In critical infrastructures, as an example of the most demanding setting, smart-grid security surveys emphasize that the availability and integrity of control functions must be protected under strict real-time and resource constraints [4], and OT security guidance calls for defense in depth, least privilege, and monitoring of operator access in industrial control systems [5]. Edge computing moves computation to the data source, reducing latency, bandwidth and exposure of sensitive data [16]. The environmental cost of large models has additionally motivated the Green AI agenda, which treats efficiency as a primary evaluation criterion next to accuracy [17]. An authentication agent that runs on each operator workstation, therefore, has to be small, fast and self-sufficient.

2.3 Explainability and human-in-the-loop autonomy

An agent that acts autonomously must also be able to justify its actions to the humans it affects. Explainable AI (XAI) aims to make the reasoning of learned models understandable to their users [27]. SHAP [18] attributes a prediction to its input features based on Shapley values, and TreeSHAP computes these attributions exactly and efficiently for tree ensembles [19], which makes it a natural fit for an XGBoost-based agent. Levels-of-automation models describe how decision authority can be shared between an automated system and a human [20], and interactive machine learning studies how human feedback can safely steer a learning system [21]. At the same time, a learning system that adapts from user feedback is exposed to poisoning, where an adversary manipulates the training signal [28]. These considerations determine how our agent explains itself, where it defers to a human, and how it learns from its own mistakes.

3 Methods

This section details the design and operational framework of the proposed lightweight, autonomous security agent. To achieve real-time, resource-efficient protection at the edge of cyber-physical systems, the agent is structured around a continuous sensory–cognitive–action cycle. We first present an overview of this overarching architecture, followed by a detailed formalization of its core components: the acquisition of raw keystroke dynamics, their transformation into a representative state space, and the generation of belief states via a gradient-boosted cognitive module. Building on this foundation, we define a state-dependent decision policy and its execution through

an adaptable action module. To ensure the system adheres to the principles of precise, explainable, and provable AI (PEP-AI), we subsequently detail the SHAP-driven explanation channel and the human-in-the-loop audit and adaptation mechanisms. Finally, we introduce an energy-oriented framework to contextualize the agent’s computational efficiency, formally bridging our methodology with the paradigms of Green AI.

3.1 Agent architecture overview

The proposed agent runs on the operator’s workstation and implements a closed sensory–cognitive–action loop (Figure 1). The *sensory module* captures keyboard events and converts them into feature vectors. The *cognitive module* (XGBoost) turns each feature vector into a belief about the identity of the typist. The *policy* combines this belief with the agent’s security state to select an action, and the *action module* carries it out, ranging from silent acceptance to a lockout that can only be lifted by operating-system re-authentication. An *explanation channel* attaches the reasons for each intervention, and a *reporting channel* sends the decision and summary timing statistics (but not the raw keystroke data) to a supervisor dashboard. All decisions are taken locally; the agent works identically when the dashboard is unreachable.

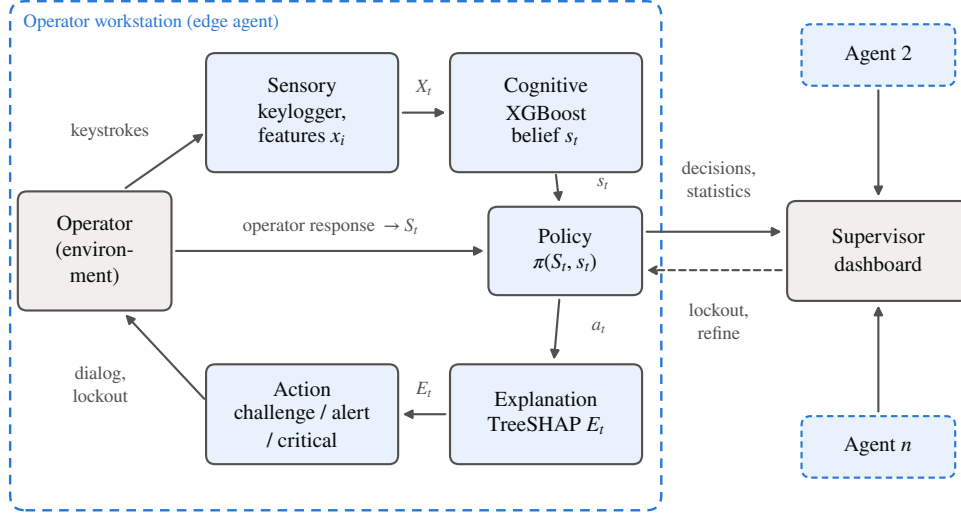


Fig. 1 The sensory–cognitive–action cycle of the edge security agent and its interaction with the operator and the supervisor.

3.2 Sensory module: keystroke acquisition

The sensory module is the custom keylogger, implemented in Python using the `pynput` library [29], which is used solely as a low-level interface to register `on_press` and

on_release events. All capture, timestamping, feature computation, and segmentation logic is our own implementation. For every keystroke, the module records the key identifier, relative and absolute press and release times, hold time, seek time, cumulative typing speed (keys per second), simultaneous key combinations, an error flag back-propagated from a subsequent *Backspace*, and the running error rate and accuracy (Table A1 in Appendix A). A queue-based structure keeps press and release events synchronized, and inactivity of 10 s resets the running statistics to prevent distortion. Absolute Unix timestamps are used only for bookkeeping and are discarded before training and inference.

These per-keystroke records are the input from which the agent forms its observations. In the agent, the event stream is segmented into non-overlapping windows of w keystrokes. The window length is a configurable parameter chosen at start-up; the reference implementation uses $w = 15$, the same window length as in the evaluation of the conference study [11]. For a window $W_t = (e_{(t-1)w+1}, \dots, e_{tw})$ of keyboard events e_i , the sensory module produces the agent’s observation, one feature vector per keystroke:

$$X_t = (x_i)_{i \in W_t}, \quad x_i = \phi(e_i \mid \mathcal{H}_i) \in \mathbb{R}^d, \quad (1)$$

where ϕ is the feature-engineering map of Section 3.3 and $\mathcal{H}_i$ is the short history of preceding events on which the n-gram, moving-window, and burst statistics depend. With one-hot encoded key identifiers and key combinations, the feature space of the reference model has $d = 609$ dimensions. The window mean $\bar{x}_t = \frac{1}{w} \sum_{i \in W_t} x_i$ serves as the representative feature vector of the window for the explanation channel (Section 3.7).

3.3 State space: feature engineering

The raw stream is processed by the pipeline of Figure 2:

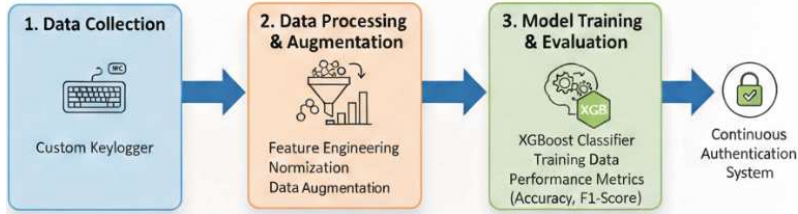


Fig. 2 Processing pipeline of the sensory and cognitive modules, from raw keystroke collection through feature engineering to model training and evaluation (from [11]).

- missing hold times caused by interrupted captures are imputed from the next key press;
- bursts of continuous typing are identified to compute cumulative keys per second and words per minute;
- the mean and standard deviation of hold and seek times over bigram and trigram sequences capture rhythmic regularities;

- moving averages and standard deviations over a window of 5 keystrokes capture short-term trends;
- micro-bursts with inter-key pauses shorter than 100 ms model fluid typing.

These steps expand the 12 raw features into 20 numerical features and 5 categorical variables, whose pairwise correlations remain moderate (Figure 3). Categorical variables are one-hot encoded, and binary variables are converted to numerical values. Together, these features define the agent’s state space: they are what the agent can perceive about the typist.

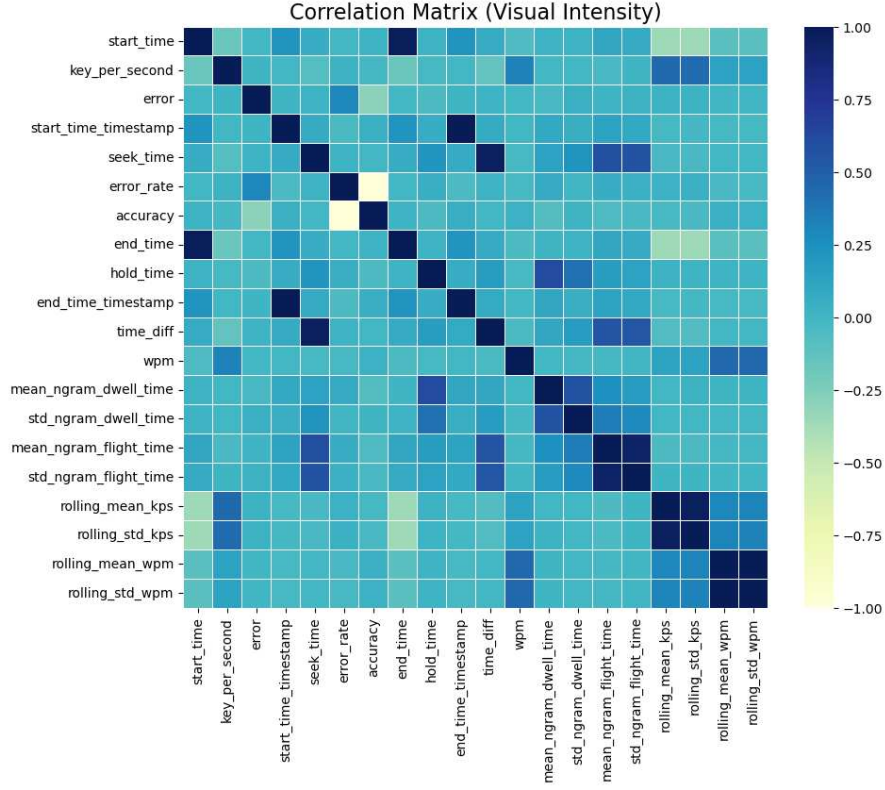


Fig. 3 Pearson correlation matrix of the core keystroke-dynamics features after feature engineering (from [11]).

3.4 Cognitive module: belief-state generation with XGBoost

The cognitive module is a binary XGBoost classifier [12] that distinguishes the enrolled operator (**Main**) from all other typists (**Other**). The benchmark dataset, collected with the sensory module, contains 13,251 keystroke samples from 5 users typing on the same keyboard: 4,337 samples of the Main user and 8,914 samples of the 4 other users, an imbalance of approximately 1:2 that was not resampled. Per-user session counts

range from 11 to 74. The data is partitioned session-wise with `GroupShuffleSplit` from `scikit-learn` [30] (random state 42) into training, validation and test sets in a 60/20/20% ratio (7,950, 2,650 and 2,651 samples), so that no recording session is shared between subsets. The model is trained with a logistic objective, 100 trees and the hyperparameters of Table 1, which were tuned on the validation set [11].

Table 1 Hyperparameters of the XGBoost cognitive module, tuned on the validation set [11].

Hyperparameter	Value
Number of trees	100
Learning rate	0.01
Maximum depth	7
γ (minimum split loss)	0.2
Subsample / column subsample per tree	0.7 / 0.7
L1 (α) / L2 (λ) regularization	0.05 / 0.1
Objective / evaluation metric	logistic / mlogloss

Principal component analysis was used only for exploratory visualization (Figure 4) and is not part of the classifier.

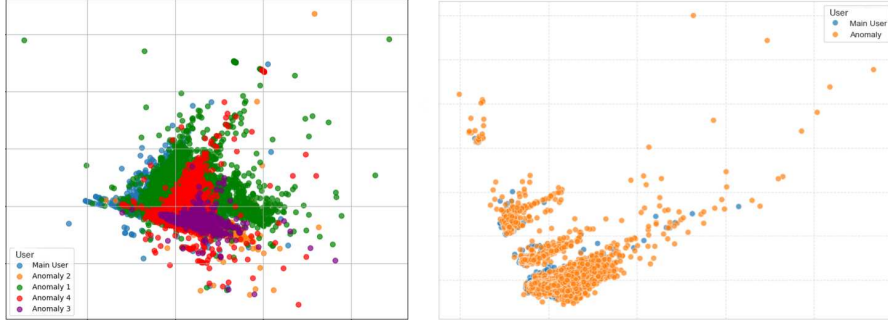


Fig. 4 PCA visualization of the dataset before (left) and after (right) feature engineering, colored by user identity; the engineered features improve the separability of the Main user (from [11]).

Once trained, the classifier is applied to the agent’s observations as follows. For an observation X_t , the classifier outputs class probabilities for every keystroke vector x_i of the window. The predicted label $\hat{y}_t$ is the majority vote over the rows, and the confidence c_t is the mean of the maximal class probabilities. The agent’s belief that the typist is *not* the enrolled operator, the *impostor score*, is

$$s_t = f_{XGB}(X_t) = \begin{cases} c_t, & \hat{y}_t = \text{Other}, \\ 1 - c_t, & \hat{y}_t = \text{Main}. \end{cases} \quad (2)$$

In the agent view, s_t is not a final output but the belief state on which the policy acts.

3.5 Decision policy

The belief s_t alone does not determine what the agent does; the decision also depends on the security state. The agent selects its action with a policy

$$a_t = \pi(S_t, f_{XGB}(X_t)), \quad a_t \in \mathcal{A} = \{\text{ALLOW}, \text{CHALLENGE}, \text{ALERT}, \text{CRITICAL}\}, \quad (3)$$

where a_t is the action of the agent (e.g., permitting the session or blocking it), S_t is the security state of the system, and $f_{XGB}(X_t) = s_t$ is the prediction of the XGBoost cognitive module.

Security state.

The state is $S_t = (\rho_t^w, \rho_t^c, \kappa_t, o_t)$. The run lengths ρ_t^w and ρ_t^c count the consecutive windows, up to and including t , whose impostor score reached the warning threshold τ_w and the critical threshold τ_c , respectively:

$$\rho_t^\tau = (\rho_{t-1}^\tau + 1) \mathbf{1}[s_t \geq \tau], \quad \tau \in \{\tau_w, \tau_c\}. \quad (4)$$

The binary flag κ_t indicates that the agent is *armed*: an alert was dismissed without re-authentication. Finally, o_t is the level of the intervention currently displayed to the operator, if any.

Policy.

The policy escalates in four levels:

$$\pi(S_t, s_t) = \begin{cases} \text{CRITICAL}, & \kappa_t = 1 \wedge \hat{y}_t = \text{Other} \wedge s_t \geq \tau_w, \\ \text{ALERT}, & \rho_t^c \geq 3, \\ \text{CHALLENGE}, & \rho_t^w \geq 2, \\ \text{ALLOW}, & \text{otherwise,} \end{cases} \quad (5)$$

evaluated from top to bottom. Requiring runs of consecutive suspicious windows prevents a single noisy window from triggering an intervention. This confirmation buffer delays a challenge by one window and an alert by two windows (w and $2w$ keystrokes) compared with acting on a single window, in exchange for decisions that rest on accumulated rather than momentary evidence. While an intervention is displayed, a new decision of the same or a lower level is suppressed, and a higher level supersedes the displayed one, so the operator is never flooded with stacked interventions.

Response-driven transitions.

The human response is part of the environment and changes the state. A challenge that is not answered within 30 s, or closed without re-authentication, escalates to an alert. An alert that is dismissed without re-authentication arms the agent ($\kappa = 1$), so that the next suspicious window leads directly to the critical level. A successful operating-system re-authentication at any level disarms the agent and clears the run lengths

($\kappa = 0, \rho^w = \rho^c = 0$), so detection restarts from ALLOW. Algorithm 1 summarizes one decision step, and Figure 5 the escalation levels with their transitions.

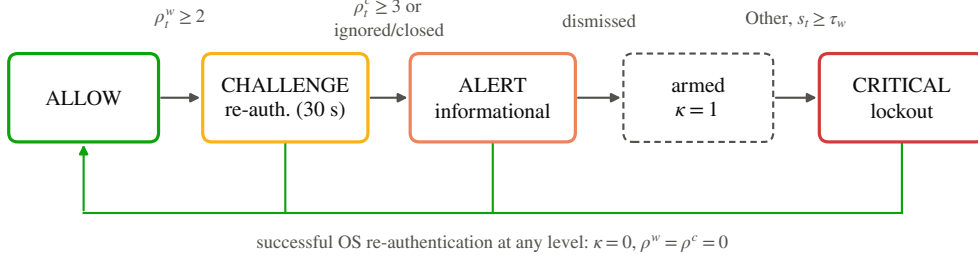


Fig. 5 Escalation levels of the decision policy (Eq. (5)) and the response-driven transitions. A higher level supersedes a displayed lower one; a successful operating-system re-authentication at any level disarms the agent and clears the run lengths.

Algorithm 1 One decision step of the security agent

Require: window W_t , state $S_t = (\rho^w, \rho^c, \kappa, o)$, thresholds $\tau_w < \tau_c$

- 1: $X_t \leftarrow$ features of W_t ▷ sensory module, Eq. (1)
 - 2: $(\hat{y}_t, s_t) \leftarrow f_{XGB}(X_t)$ ▷ cognitive module, Eq. (2)
 - 3: update ρ^w, ρ^c ▷ Eq. (4)
 - 4: $a_t \leftarrow \pi(S_t, s_t)$ ▷ Eq. (5)
 - 5: **if** $a_t = \text{ALLOW}$ **or** ($o \neq \emptyset$ **and** $\text{rank}(a_t) \leq \text{rank}(o)$) **then**
 - 6: **return** ▷ no new intervention
 - 7: **end if**
 - 8: $E_t \leftarrow$ top-5 SHAP reasons for $\bar{x}_t$ ▷ explanation channel, Sect. 3.7
 - 9: log and report $(a_t, s_t, E_t, \tau_w, \tau_c)$ ▷ provable audit trail
 - 10: execute the action hook of a_t ; $o \leftarrow a_t$ ▷ action module
 - 11: on the operator's response: re-authentication $\Rightarrow \kappa, \rho^w, \rho^c \leftarrow 0$; dismissed alert $\Rightarrow \kappa \leftarrow 1$; ignored challenge $\Rightarrow$ escalate to ALERT
-

Thresholds from the cognitive module's error characteristics.

The policy and Algorithm 1 rely on the two thresholds τ_w and τ_c , which are defined next. The thresholds are not set by hand; they are computed after training from the ROC curve [31] of the cognitive module on its own held-out test set, with the impostor (Other) class as the positive class and the keystroke-level impostor probabilities as scores. Let $\text{FPR}(\theta)$ be the fraction of the Main user's test keystrokes with a score of at least θ (legitimate activity that would be flagged) and $\text{FNR}(\theta)$ the fraction of impostor keystrokes with a score below θ (missed impostor activity). The critical threshold is

the operating point of the Equal Error Rate,

$$\tau_c = \arg \min_{\theta} |\text{FPR}(\theta) - \text{FNR}(\theta)|, \quad (6)$$

and the warning threshold is the most conservative lower threshold at which the Main user would be flagged more than twice as often as at τ_c :

$$\tau_w = \max \{ \theta : \text{FPR}(\theta) > 2 \text{FPR}(\tau_c) \}. \quad (7)$$

The band $[\tau_w, \tau_c)$ is thus a “suspicious but not conclusive” region that can only trigger a challenge, while an alert requires sustained evidence above the EER operating point. If no point of the ROC curve satisfies Eq. (7), the implementation falls back to $\tau_w = 0.85 \tau_c$. The thresholds are stored with the model and are recomputed whenever the model is retrained.

3.6 Action module

Once the policy has selected a level, the action module carries it out. The actions are implemented as editable hooks (`on_challenge`, `on_alert`, `on_critical`; Appendix B), so that a deployment can adapt the response without changing the policy. The responses described below are only the defaults: the action taken at each level is fully customizable, since each hook is an ordinary function that receives the incident and the agent and can be replaced by any deployment-specific behavior, while the logging and reporting of the decision take place regardless of the hooks. The default responses are: an always-on-top *challenge* dialog that asks for re-authentication and escalates after 30 s; an informational *alert* dialog; and at the *critical* level, a lock-out dialog that can only be closed by a successful operating-system re-authentication, together with a call to an enforcement hook. The enforcement hook, which could suspend network access or terminate the session, is deliberately implemented as a logged dry run in the current version; connecting it to a real actuator is left to a reviewed, deployment-specific decision (Section 5). Re-authentication is delegated to the operating system’s own prompt, so the agent never handles a password. A supervisor can additionally trigger the logout of a workstation remotely from the dashboard.

3.7 Explanation channel: SHAP as operator communication

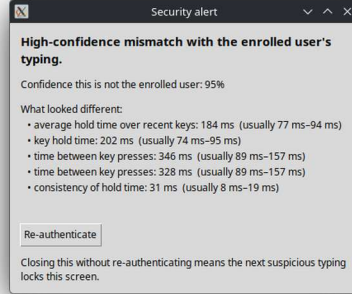
Because an intervention interrupts the operator’s work, the agent also has to state why it intervened. For every intervention, the agent computes TreeSHAP attributions [19] for the representative window vector $\bar{x}_t$ locally. Let $\varphi_j(\bar{x}_t)$ be the SHAP value of feature j with respect to the model’s positive class. The *impact* of feature j toward the impostor hypothesis is

$$\psi_j(\bar{x}_t) = \sigma \varphi_j(\bar{x}_t), \quad \sigma = \begin{cases} +1, & \text{if the positive class is Other,} \\ -1, & \text{if the positive class is Main,} \end{cases} \quad (8)$$

so that $\psi_j > 0$ always means that feature j made the window look less like the enrolled operator.

The explanation is presented in plain language, both in the dialog on the workstation and to the supervisor (see Figure 6).

(a) Dialog on the operator workstation



(b) The same incident on the supervisor dashboard (top reason shown)

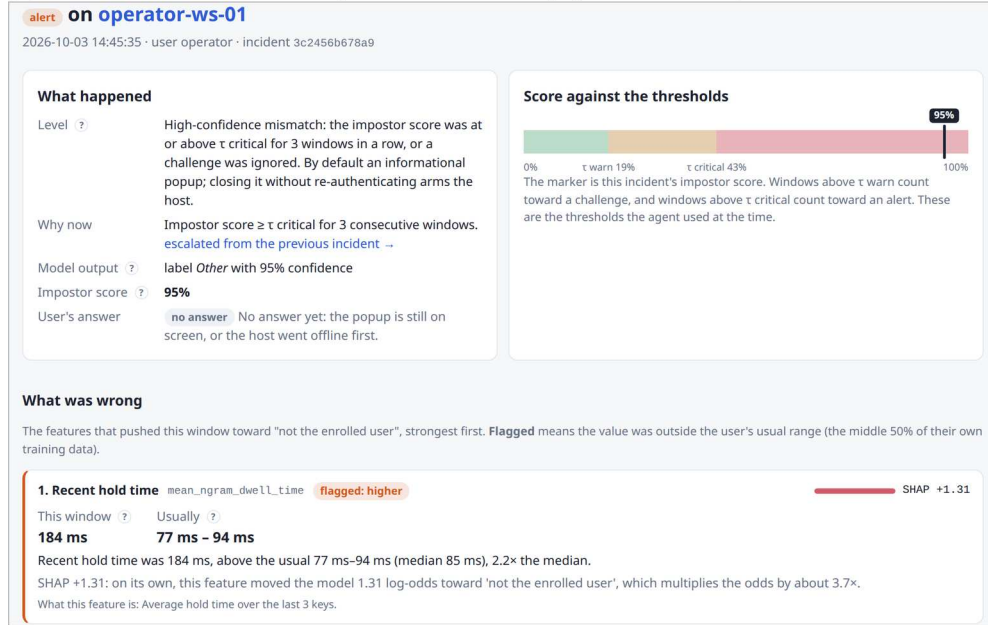


Fig. 6 The explanation channel for one real incident: an alert raised on a held-out impostor session (impostor score 95%). (a) The dialog shown on the workstation with the five reasons; (b) the same incident on the supervisor dashboard, with the score on the τ_w/τ_c scale and the top reason with its usual range and SHAP impact.

The explanation E_t consists of the (at most) five timing and statistical features with the largest positive impact; the one-hot key and key-combination indicators are excluded, so that the explanation never reveals which keys were typed. For each of them, the agent reports the value in the current window and the enrolled operator’s usual range, the interquartile range $[q_{25}, q_{75}]$ of that feature over the operator’s own training data, which is stored with the model, and states whether the value is above, below, or within that range.

For example, in the incident of Figure 6, recorded by replaying a held-out impostor session through the agent, the average hold time over the last three keys was 184 ms, above the operator’s usual 77–94 ms, with a SHAP impact of +1.31, which multiplies the odds of the impostor hypothesis by about $e^{1.31} \approx 3.7$. In terms of the PEP-AI requirements, SHAP is therefore not a post hoc analysis tool but the channel through which the agent justifies its decision to intervene.

3.8 Human-in-the-loop operation, audit and privacy

The explanation is addressed to people, whose responses in turn shape the agent’s behavior. The human operator is part of the control loop. Every intervention ends with an explicit outcome (re-authenticated, closed, timed out, or superseded by a higher level), which is fed back into the state S_t and recorded. Each agent reports its decisions to a supervisor dashboard that lists the connected workstations, the history of incidents with explanations, and live statistics for each operator’s model. Every incident record contains the thresholds that were in force at the time of the decision, so that each intervention can be reproduced and verified later, which is the *provable* component of PEP-AI. Because these reports leave the workstation, their content is deliberately restricted. The reporting channel is designed as a privacy boundary (Table 2): biometric raw data and the model stay on the edge device, and the dashboard receives only decisions and aggregate timing statistics.

Table 2 Information crossing the boundary between the edge agent and the supervisor dashboard.

Reported to the supervisor	Never leaves the workstation
Host and operator identity; model fingerprint, thresholds and test-set performance; the operator’s usual range of each timing feature	Raw keystrokes and key identities
Each intervention: level, impostor score, trigger, top SHAP reasons (timing features only), thresholds in force	Per-keystroke feature vectors x_i , and the key and key-combination components of $\bar{x}_t$
The operator’s response to each intervention	Key and key-combination frequencies
Latest impostor scores and the timing statistics of the last window (timing components of $\bar{x}_t$)	The trained model

3.9 Human-reviewed adaptation

The recorded outcomes also allow the agent to learn from its own false interventions. When the operator passes the re-authentication at a CHALLENGE or ALERT, the agent has made a mistake: the typist was the enrolled operator. The keystroke windows of the suspicious stretch that led to the intervention are then flagged for review on the workstation. They are not learned automatically. Only an explicit “refine model” action, triggered by the operator or by the supervisor, adds them to the training data as the Main user, re-trains the model, recomputes τ_w and τ_c , and loads the new model into the running agent. Re-authentication at the CRITICAL level is never used for learning, because by that point an alert has already been dismissed, and the password alone is weaker evidence of the operator’s identity. Keeping a human in the adaptation loop reduces the risk that an adversary who once obtained the password could gradually poison the model with fabricated sessions [28].

3.10 Energy-oriented interpretation of the methods

Each of the components described above contributes to the energy that the agent consumes. The energy that the agent consumes over an observation period can be decomposed according to the four dimensions introduced in Section 1:

$$E_{\text{agent}} \approx \underbrace{n_w \bar{P} t_{\text{inf}}}_{\text{cognition}} + \underbrace{n_a E_{\text{int}}}_{\text{interventions}} + \underbrace{n_r \bar{P} T_{\text{train}}}_{\text{adaptation}} + \underbrace{E_{\text{comm}}}_{\text{communication}}, \quad (9)$$

where n_w is the number of scored windows, $\bar{P}$ the mean power draw of the workstation processor while the agent computes, t_{inf} the response time of the cognitive module, n_a the number of interventions, E_{int} the energy of one intervention path (SHAP computation, dialog, logging and report), n_r the number of model refinements, T_{train} the training time and E_{comm} the energy of communicating with the supervisor. The energy of keystroke capture is identical for every cognitive module and is therefore omitted from the comparison. The feature preparation differs between the two modules (engineered features of 15-keystroke windows for XGBoost, scaled 5-keystroke sequences for TypeNet); it was not measured separately and is not part of the comparison.

Not every term of Eq. (9) was measured. The results of this article inform four of its factors: t_{inf} and T_{train} (Table 3), the number of interventions n_a , which is governed by the error characteristics and the operating point of the cognitive module (Sections 4.1 and 4.3), and the structure of E_{comm} , which is governed by the privacy boundary of Table 2. The power $\bar{P}$, the energy E_{int} of an intervention, and the absolute value of E_{comm} were not measured. Two cognitive modules that run on the same class of processor at a comparable CPU utilization can nevertheless be compared: if $\bar{P}_X \approx \bar{P}_T$ for XGBoost (X) and TypeNet (T), the ratio of the energy of the two modules for the same task equals the ratio of their compute times,

$$\frac{E_X}{E_T} = \frac{\bar{P}_X t_X}{\bar{P}_T t_T} \approx \frac{t_X}{t_T}, \quad (10)$$

independently of the number of workstations and of the amount of typing, since both enter Eq. (9) as common factors. Eq. (10) is an indicator of relative compute

effort under an explicit assumption, not a measurement of energy. It ignores the power drawn by memory and by idle components, and it does not hold if the two models load the processor differently. Section 4 uses it, together with peak memory and model size, to explain each result in relation to energy use.

3.11 Theoretical motivation

Finally, the choice of a tree ensemble as the cognitive module can also be motivated conceptually. As a conceptual motivation for tree-based modelling, keystroke dynamics can be viewed as a stochastic, rhythmic process $v_i = (h_i, f_i, k_i, c_i, e_i, s_i)$ of hold time, flight time, key identity, context, event type and statistical aggregates, whose short-term dependence suggests an autoregressive analogy $v_i = \sum_{j=1}^p \alpha_j v_{i-j} + \epsilon_i$ [11]. Classification requires a fixed-length template of statistical moments of hold and flight times, including digraph-specific terms, which is what the feature map ϕ provides; the distance between two operators’ templates then quantifies their difference in identity. This framing is an interpretive lens and was not computed or validated as part of the pipeline; none of the results in Section 4 depend on it. It motivates, however, why a gradient-boosted tree ensemble, which captures non-linear relations between statistical moments without recurrent computation, is an adequate cognitive module for a resource-constrained agent.

4 Results

This section evaluates the proposed autonomous security agent across multiple dimensions critical for mass deployment at the edge of cyber-physical systems: decision quality, computational footprint, and operational efficiency. We first benchmark the XGBoost-based cognitive module against an LSTM-based TypeNet baseline, demonstrating that superior biometric authentication performance can be achieved while concurrently reducing reaction time, memory consumption, and adaptation latency. Subsequently, we analyze the agent’s dynamic operating point, illustrating how ROC-derived thresholds and a state-dependent policy constrain the False Intervention Rate, thereby minimizing costly operator disruptions. These empirical findings are systematically mapped to our energy-oriented framework, substantiating the agent’s alignment with Green AI principles and indicating that robust, human-in-the-loop security (PEP-AI) does not necessitate a trade-off with computational overhead. Finally, we evaluate the complete agent end to end on held-out sessions, together with the quality of its explanations.

4.1 Decision quality of the cognitive module

Table 3 summarizes the decision quality of the two cognitive modules. The XGBoost module identifies the Main user with 99.8% accuracy, 1.2 percentage points above TypeNet (98.6%), and reduces the EER from 0.0167% to 0.0017%. In the agent setting, the EER is reinterpreted as the keystroke-level *False Intervention Rate* (FIR) at the EER operating point: at this threshold, the fraction of the legitimate operator’s keystrokes that would be treated as an intrusion equals the fraction of an intruder’s

keystrokes that would be accepted. A lower FIR means fewer unnecessary interruptions of the operator’s work and fewer missed intrusions. The Support values are not directly comparable across models, because TypeNet’s Support counts overlapping 5-keystroke sequences, whereas XGBoost’s counts individual keystrokes.

Table 3 Decision quality and resource use of the cognitive modules (from [11]), with the agent-level interpretation used in this article.

Metric	TypeNet (LSTM)	XGBoost	Agent-level interpretation
Accuracy	98.6%	99.8%	correct identity beliefs
EER	0.0167%	0.0017%	keystroke-level FIR
AUC	0.9995	0.9999	separability of belief states
Recall	0.99	1.00	
Precision	0.99	1.00	
F1-score	0.99	1.00	
Support	728	2651	(not directly comparable)
Training time (CPU)	87.85 s	1.02 s	cost of (re)adaptation
Average inference time	180 ms	63 ms	reaction time per decision
Model size	1.99 MB	0.18 MB	deployable footprint
Peak memory usage	145.90 MB	20.52 MB	per-agent memory footprint

The decision quality determines how much of the agent’s costly work is useful. Only an intervention triggers the explanation, dialog, logging and incident-reporting path ($n_a E_{\text{int}}$ in Eq. (9)); the short status updates sent to the dashboard after scored windows belong to E_{comm} , so every false intervention on the legitimate operator is a wasted execution of this path and an unnecessary interruption of the operator’s work, whereas every missed intrusion leaves an opened workstation, and the systems that can be controlled from it, exposed to a take-over. A lower FIR therefore reduces the avoidable computing cost of security and the exposure of the protected systems at the same time. Furthermore, in this comparison the more accurate module is also the cheaper one per decision (Table 3), so no accuracy–energy trade-off arises, in contrast to the practice of buying accuracy with additional computation that the Green AI agenda criticizes [17].

4.2 Reaction time and footprint of the agent

In an agent operating in critical infrastructure, inference latency determines the agent’s reaction time, and, at the same time, the time the processor is loaded by a single decision. The XGBoost cognitive module needs 63 ms on average, from the start to the end of the model prediction, to score one window, i.e., to reach a decision, about 2.86 times faster than TypeNet’s 180 ms per window; a TypeNet-based agent would therefore intervene with a noticeably longer delay. The two windows are not of equal length: the XGBoost module scores 15 keystrokes at once, whereas a TypeNet window is one sequence of 5 keystrokes, so per keystroke covered the difference is larger than this ratio. Under the equal-power assumption of Eq. (10), the same ratio means that the XGBoost module needs about $63/180 \approx 0.35$ of the compute energy of TypeNet

for each window-based decision, at any number of workstations. The XGBoost figures were observed at a CPU utilization of around 5% on standard desktop processors (Intel i7-7700K/i3-10100), without an accelerator.

Besides the compute effort of each decision, the agent also occupies resources for as long as it runs. The peak memory consumption of the XGBoost module during prediction is 20.52 MB, roughly 7 times less than TypeNet’s 145.90 MB. Because one agent runs on every operator workstation, this footprint is paid once per workstation. In a distributed IoT/edge deployment, the LSTM-based agent would therefore be a heavyweight process that scales poorly, whereas the small footprint of the XGBoost agent permits a mass, low-energy (“Green”) deployment [17] on the hardware already present at the network edge.

The trained XGBoost model occupies 0.18 MB, about 11 times less than the 1.99 MB of TypeNet, which is the volume that has to be stored on each workstation and would have to be transferred whenever a model is distributed or updated. Training takes 1.02 s on the CPU instead of 87.85 s, about 86 times less, so that under the same assumption one adaptation costs about 1/86 of the compute effort. Both times were measured on the same dataset with the same training and test partition, which is fixed by the random seed for reproducibility. A refinement adds only the reviewed windows, which are few compared with the existing training data, so the training time of an adaptation can be expected to vary only slightly as the dataset grows. This is what makes the human-reviewed adaptation of Section 3.9 practical on the workstation itself: the model is retrained where the data is, and neither raw behavioral data nor a model has to be transferred to a server, and no dedicated training hardware is needed, for an adaptation.

Figure 7 summarizes these ratios. The energy consumption itself was not measured in this study, and the ratios above are therefore indicators of relative compute effort, not of absolute energy (Section 5).

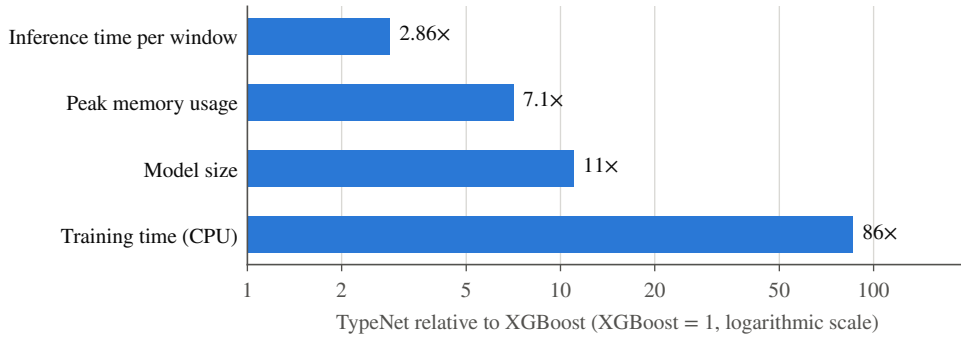


Fig. 7 Resource use of the TypeNet cognitive module relative to the XGBoost cognitive module (ratios computed from Table 3).

4.3 Operating point of the agent

Decision quality and cost per decision are properties of the cognitive module; how often the agent actually intervenes is governed by its operating point. The agent’s thresholds are derived with Eqs. (6)–(7) from the ROC curve of the deployed cognitive module. In the reference implementation of the agent, this derivation, performed by its training pipeline on a session-wise held-out test set for the model version deployed there (trained with the implementation’s default hyperparameters rather than those of Table 1), yields the operating point $\tau_w = 0.191$ and $\tau_c = 0.434$ (Table 4, Figure 8).

Table 4 Operating point of the agent in the reference implementation.

Quantity	Value
Warning threshold τ_w	0.191
Critical threshold τ_c (EER operating point)	0.434

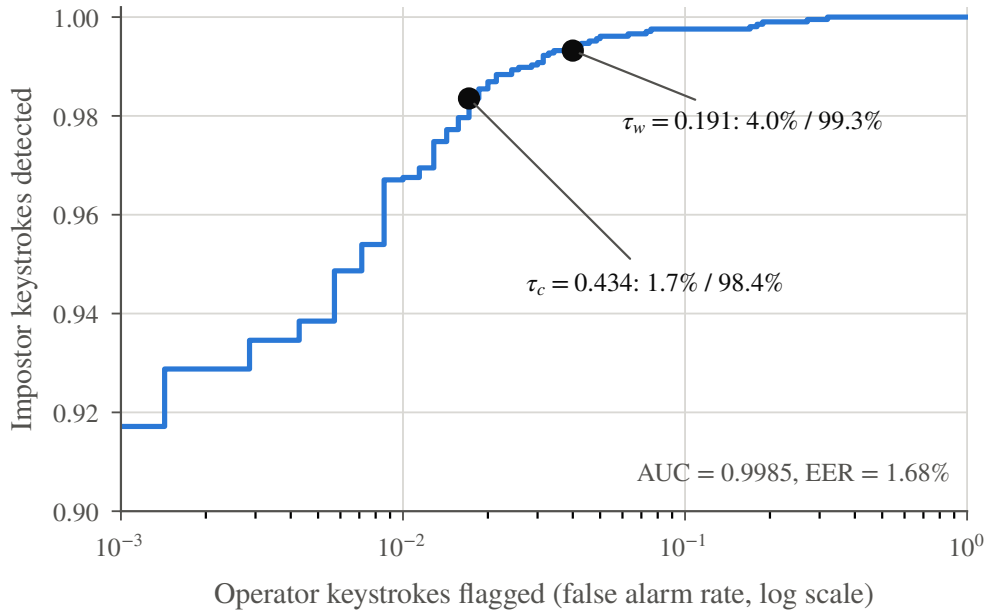


Fig. 8 ROC curve of the reference implementation’s cognitive module on its own session-wise held-out test set (keystroke level), with the operating points τ_w and τ_c of Table 4. Each point gives the share of the enrolled operator’s keystrokes that would be flagged and the share of impostor keystrokes that would be detected; the horizontal axis is logarithmic.

This model was trained and evaluated in a separate run, on a different computer and on its own session-wise test set of 2,764 keystrokes, so its error rates differ from

those of Table 3. The impostor-score scale is thereby divided into an ALLOW region ($s < 0.191$), a CHALLENGE region ($0.191 \leq s < 0.434$), and a region that can lead to an ALERT ($s \geq 0.434$). Since the thresholds are recomputed with every (re)training of the model, they characterize the procedure and the state of the reference implementation rather than being fixed constants.

The operating point decides how often the costly intervention path is entered. At τ_c , the keystroke-level false alarm rate on the Main user equals, by construction, the EER of the model version for which τ_c was computed, but the policy of Eq. (5) does not act on a single score: a challenge requires two consecutive windows at or above τ_w , an alert three consecutive windows at or above τ_c , and the SHAP explanation, the dialog and the report are produced only when a new intervention is created (Algorithm 1). In addition, τ_w and τ_c are derived from keystroke-level scores but applied to the window-level score s_t of Eq. (2), so the per-window false alarm rate at τ_c is not identical to the keystroke-level EER; it is measured end to end in Section 4.5. The number of interventions n_a in Eq. (9) is therefore expected to be lower than the raw error rate suggests, and a cognitive module with better separability (higher AUC) generally moves the ROC-derived operating point towards fewer false interventions.

4.4 Energy-oriented synthesis

Table 5 collects the results of this section together with their reading in terms of the four energy dimensions of Section 1. The values are those of Tables 3 and 4 and of the replay in Section 4.5, and the ratios are computed from Table 3.

Read together, the results place the agent on the favorable side of every dimension: it is cheaper per decision, smaller to hold and to distribute, and cheaper to adapt than the LSTM baseline, while it is also the more accurate module. What Table 5 cannot provide are absolute energy values, which require power measurements on the workstation.

Table 5: Energy-oriented reading of the results. Dimensions: (i) compute effort of a decision, (ii) footprint of holding, distributing and adapting the model, (iii) costly work triggered by errors, (iv) data leaving the workstation. Ratios are TypeNet relative to XGBoost.

Result	Value	Energy-related reading
Average response time per window	63 ms vs 180 ms (2.86 $\times$)	(i) Compute effort per decision; about 0.35 of TypeNet under Eq. (10)
CPU utilization	around 5% (i7-7700K / i3-10100)	(i) The agent runs on the workstation’s own CPU, without an accelerator
Peak memory	20.52 MB vs 145.90 MB (about 7.1 $\times$)	(ii) Resident footprint that is paid once per workstation; decides whether a mass edge deployment scales
Model size	0.18 MB vs 1.99 MB (about 11 $\times$)	(ii) Storage per workstation and data volume of each model distribution or update

Table 5: Energy-oriented reading of the results (continued)

Result	Value	Energy-related reading
Training time (CPU)	1.02 s vs 87.85 s (about $86\times$)	(ii) Compute effort per adaptation; enables refinement on the workstation without transferring behavioral data to a server
Accuracy and EER (FIR)	99.8% and 0.0017% vs 98.6% and 0.0167%	(iii) Fewer false interventions mean fewer executions of the explanation, dialog and report path; fewer missed intrusions mean less exposure of the protected systems
Operating point	$\tau_w = 0.191$, $\tau_c = 0.434$ (reference implementation); keystroke-level false alarm rate at $\tau_c =$ EER of that model version	(iii) Sets how often the costly path can be entered; the consecutive-window requirement suppresses single-window noise (no false intervention in the replay of Section 4.5, against two for a single-window policy)
Privacy boundary (Table 2)	decisions and aggregate timing statistics only	(iv) The raw stream, the per-keystroke feature vectors and the model stay on the workstation, so only decisions and summary statistics have to be communicated (about 1 kB per window, 2.7 kB per incident report; Table 7)

4.5 End-to-end behavior of the agent

The preceding subsections characterize the cognitive module and the thresholds separately; the replay below shows how they act together in the policy. To evaluate the complete agent, the held-out test sessions of the reference implementation (its own session-wise split; 48 sessions) were replayed through the agent’s code. The recorded keyboard events were cut into windows of $w = 15$ keystrokes exactly as the live keylogger forms them, scored by the agent’s inference module, and passed to the policy of Eq. (5) with the thresholds of Table 4. Each typist’s sessions form one continuous stream in chronological order. The human responses were simulated: the enrolled operator re-authenticates at every challenge and alert, whereas an impostor closes every dialog at once without re-authenticating, so that a challenge becomes an alert, a dismissed alert arms the agent, and the next suspicious window leads to the critical lockout. Table 6 reports the result.

The agent locked the workstation for three of the four impostors after 45–60 keystrokes, i.e., within 15–22 s of typing. The typing of Impostor 3 was scored as the enrolled operator in almost every window (4 of 48 windows reached τ_w , 1 reached τ_c), so no intervention was raised for this typist. The operator’s own stream shows how the policy handles false alarms. On the enrolled operator’s stream, 2 of 42 windows (4.8%) reached τ_w and 1 (2.4%) reached τ_c , which is the per-window false alarm rate at the operating point. Because these windows were not consecutive, the policy raised no intervention on the operator’s 630 keystrokes, whereas a single-window policy (an

Table 6 End-to-end replay of the held-out sessions through the agent ($w = 15$, thresholds of Table 4). The last three columns give the first challenge, the first alert and the critical lockout, counted from the start of the typist’s stream in keystrokes and in seconds of active typing; the impostor closes every dialog without re-authenticating.

Typist	Keystrokes	Windows $s_t \geq \tau_w$	Challenge	Alert	Lockout
Enrolled operator	630	4.8%	no intervention		
Impostor 1	135	100%	30 (14.7 s)	30 (14.7 s)	45 (21.6 s)
Impostor 2	810	59.3%	45 (11.0 s)	45 (11.0 s)	60 (15.2 s)
Impostor 3	720	8.3%	not detected		
Impostor 4	180	100%	30 (15.7 s)	30 (15.7 s)	45 (20.7 s)

intervention at the first window above a threshold) raised two, one challenge and one alert. The confirmation buffer thus removed both false interventions, at the cost of one additional window (15 keystrokes) before the first intervention on an impostor. The simulated impostor response also affects when the lockout occurs. If the impostor leaves the challenge open instead of closing it, the alert follows as soon as three consecutive windows reach τ_c (or after the 30 s timeout), and the lockout came one to two windows (15–30 keystrokes) later than in Table 6. The replay covers only the held-out part of the dataset (1.4 min of the operator’s typing), so these rates are indicative.

Besides detection, the replay also allows the cost of each component of the agent to be measured. Table 7 breaks the cost of the agent down by component.

Table 7 Cost of the agent’s components, measured with the agent’s code on the 165 held-out windows of Table 6 (medians). The measurement was made on a different, newer workstation (AMD Ryzen 9 5900X, Python 3.13, XGBoost 3.4) than Table 3, so the absolute times are lower than there; the table shows how the cost divides between the components. CPU time includes all threads of the process; memory is the peak of the Python and NumPy allocations within the component. The report was sent to a dashboard on the same workstation, so network latency is not included. The dialog is not included, since its duration is set by the operator.

Component	When	CPU / wall (ms)	Memory (MB)
Feature preparation (sensory)	each window	11.4 / 11.3	0.16
Prediction (cognitive)	each window	28.0 / 27.1	0.57
TreeSHAP reasons	intervention	1.9 / 0.9	0.08
Report (local round trip)	intervention	– / 1.7	–
Data		Size	
Raw window (15 keystrokes)	stays local	2.8 kB	
Feature matrix (15×609)	stays local	73.1 kB	
Status update	each window	1.0 kB	
Report with one incident	intervention	2.7 kB	

Every window costs feature preparation and prediction, about 38 ms of computation together on the measurement workstation, of which the cognitive module takes about 70%. The explanation and the report, which are executed only when an intervention is raised, add about 2.6 ms, i.e., less than a tenth of the per-window cost; the computing part of E_{int} in Eq. (9) is therefore small compared with the cognition term, and a false intervention costs mainly operator attention rather than computation. On the communication side, the status update sent after a window is about 1 kB, roughly a third of the raw window it summarizes and about 1/70 of its feature matrix, and a report with one incident is 2.7 kB; the raw window and its feature vectors never leave the workstation.

Quality of the explanations.

An intervention is only useful to the operator if its explanation reflects what actually drove the model. Three properties of the explanation channel (Section 3.7) were measured on the 59 held-out windows whose impostor score reached τ_w , i.e., the windows that can lead to an intervention. *Fidelity*: resetting the five reported features of a window to the operator’s usual (median) values lowered the model’s impostor probability for the window vector $\bar{x}_t$ by 0.174 on average, nearly all of its mean value of 0.182, whereas resetting five randomly chosen timing features lowered it by 0.062; the reported reasons had the larger effect in every window. *Stability*: in consecutive explained windows of the same typist, on average three of the five reasons recurred (mean Jaccard overlap of the reason sets 0.45). *Verifiability*: 96.6% of the reported values lay outside the operator’s usual range, so nearly every reason is a concrete deviation that the operator can check, such as a hold time of 184 ms against a usual 77–94 ms. The explanations therefore point to the features that drive the model’s output and are expressed in terms that the operator can verify.

4.6 Reproducibility

To ensure the transparency and reproducibility of this research, the complete source code of the agent, including the keylogger, the processing and training pipeline, the decision policy, the explanation channel, the editable action hooks and the supervisor dashboard, is made publicly available in a dedicated GitHub repository [32]. The code of the conference study, including the TypeNet baseline, remains available in its own repository [33].

5 Discussion

Although the cognitive module identifies the operator with 99.8% accuracy, full autonomy would be dangerous in a security-critical environment: a false lockout of the operator at a critical moment, for example during a disturbance in a power-system control room, could endanger the controlled process. The agent is therefore designed for supervisory control rather than full autonomy [20]. Its autonomous actions are limited and reversible by the human, while the decision with the highest impact is left to humans. A challenge is resolved by the operator; an alert is informational and does not lock anything; the lockout at the critical level, which is reached only after an

alert was dismissed, can be lifted at any time by the operator’s own operating-system re-authentication; and the real enforcement action is, by default, a dry run whose activation remains a reviewed decision of the organization. The tree-based structure of XGBoost makes this human-in-the-loop operation practical: TreeSHAP computes exact attributions for tree ensembles efficiently on the workstation [19], whereas explanations of an LSTM rely on approximate and computationally more expensive attribution methods [18]. The agent can thus justify its suspicion, for instance, that the operator’s key hold times in the last window were above their usual range, and the supervisor can take the final decision in high-risk cases on the basis of that justification.

This division of authority between agent and human builds on the PEP-AI requirements introduced in Section 1. The three PEP-AI properties are realized by distinct mechanisms. Precision is achieved by deriving thresholds from the model’s measured error characteristics rather than choosing them by hand, and by requiring sustained evidence before an intervention. Explainability is provided by the SHAP channel, which translates the model’s reasons into the operator’s own typing statistics. Provability is provided by the audit trail, in which every intervention, its explanation, the thresholds in force and the human response are recorded by the supervisor dashboard; while the dashboard is unreachable, these records are held in a bounded buffer on the workstation and delivered when the connection returns.

The results relate to energy in two directions. First, they concern the energy consumed by the protection. The efficiency results of Section 4.2 show that the security function can be deployed on every operator workstation without dedicated accelerators or permanent connectivity, and that a decision, a model update, and an adaptation are cheaper than with the LSTM baseline by the factors of Table 5. Because the ratios do not depend on the number of workstations (Eq. (10)), the relative saving holds from a single workstation to a fleet of edge nodes, whereas the absolute saving depends on the typing activity, the processor power, and the number of interventions in long-term operation, which were not measured. Second, they concern the protected system, which in the energy sector means the security of supply. The graded policy avoids the cost of a wrongly triggered intervention, both in computing effort and, more importantly, in operator attention at critical moments, and the measured error characteristics indicate, on the benchmark data, how likely a takeover of an opened workstation is to remain unnoticed. Security and energy efficiency are therefore not opposing goals in this design: the cheaper cognitive module is also the more accurate one, and its lower cost makes it easier to protect every workstation rather than a select few. Because the agent decides locally and reports only decisions and summary statistics (Table 2), a failure of the central dashboard does not disable the protection of the individual workstations, and no raw behavioral stream has to be transmitted (Figure 9). None of these properties depends on the energy domain: they follow from the agent organization itself (local, event-driven decisions, explanations and incident reports only on intervention, and decisions instead of raw data crossing the network), so they carry over to any organization that protects its workstations in the same way.

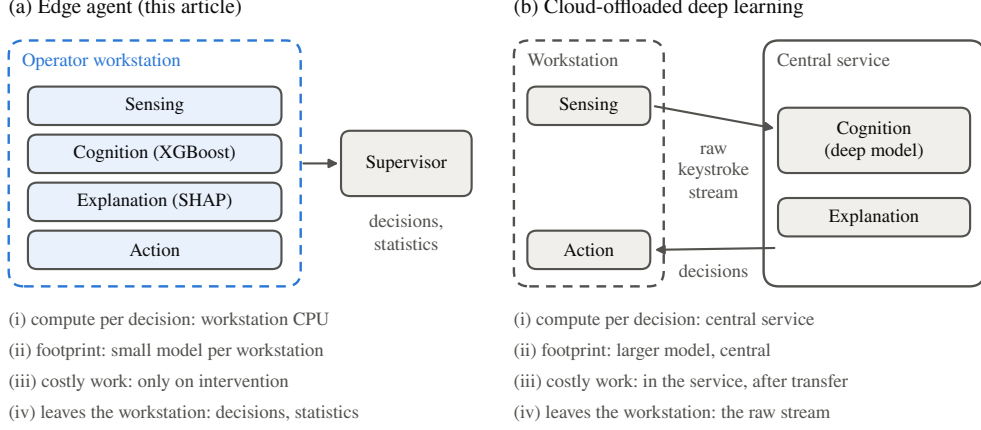


Fig. 9 Conceptual comparison of the edge agent with a cloud-offloaded architecture in terms of where computation happens and which data leave the workstation.

Beyond the energy perspective, the results can also be placed among earlier keystroke-dynamics studies. Our findings agree with recent studies in which gradient-boosted trees, and XGBoost in particular, perform strongly on fixed- and free-text keystroke dynamics [10, 23–25], although the reported figures are not directly comparable with ours because the datasets, numbers of users and evaluation protocols differ [11].

The present study has the following limitations. The dataset was collected over a short period with a limited number of sessions per user, so the temporal stability of the features and the long-term drift of typing behavior were not examined. Variations of physical working conditions and of the operator’s state (fatigue, stress, distraction) were not recorded. Energy efficiency is argued for based on latency, memory, model size, and training time rather than being measured. The energy ratios of this article rest on the assumption of a comparable power draw of the two cognitive modules (Eq. (10)), ignore the power drawn by memory and idle components, and refer to the average response time per window. The energy of the explanation and reporting path and of communication with the dashboard was not measured directly; only their computing time and data volume were (Table 7). Because the test split is drawn at random over sessions (with a fixed seed only for reproducibility) and is recomputed over the enlarged dataset after each refinement, the test partition changes, and the thresholds and test metrics change between model versions.

6 Conclusions

This article extended an accurate and efficient keystroke-dynamics classifier into a lightweight, autonomous security agent to protect operator workstations in general, with critical cyber-physical systems such as smart grids as the most demanding setting. The XGBoost model became the belief-forming cognitive module of an agent

that continuously senses the typing stream, evaluates the operator’s identity, and executes a graded, explainable intervention policy whose thresholds are derived from the model’s own error characteristics. The cognitive module’s per-window prediction time of 63 ms and memory footprint of 20.52 MB, compared with 180 ms and 145.90 MB for an LSTM-based alternative, and its 1.02 s training and 0.18 MB model, compared with 87.85 s and 1.99 MB, translate into a 2.86 times lower compute effort per decision and an about 86 times lower effort per adaptation under a comparable power draw. In an end-to-end replay of held-out sessions, the agent locked the workstation for three of four impostors within 15–22 s of typing without a false intervention on the operator. Together with a low keystroke-level false intervention rate, which limits how often the explanation and incident-reporting path is expected to be used, this makes a mass edge deployment in line with Green AI feasible, while the SHAP-based explanation channel, the audit trail, and the human-in-the-loop design keep the final authority in high-risk situations with the human operator.

The limitations discussed in Section 5 point to the next steps. Future work will quantify the energy of the agent’s components and of its communication, evaluate the complete agent over time and in operational settings (e.g., on a SCADA testbed), study the operators’ understanding of the explanations, extend the sensory module to additional modalities such as mouse dynamics and touchscreen input, replace one-hot encoding with learned embeddings to reduce dimensionality, and test the enforcement actuator of the critical level and the communication between the agents and the supervisor in the deployment setting, where their design is left to the operator of the system and can be adapted freely (Section 3.6), so that the agent layer can itself be trusted.

Abbreviations. AUC: Area under the ROC curve; EER: Equal Error Rate; FIR: False Intervention Rate; FNR: False negative rate; FPR: False positive rate; HITL: Human-in-the-loop; LSTM: Long short-term memory; MAS: Multi-agent system; OT: Operational technology; PCA: Principal component analysis; PEP-AI: Precise, Explainable, Provable AI; ROC: Receiver operating characteristic; SCADA: Supervisory control and data acquisition; SHAP: SHapley Additive exPlanations; XAI: Explainable artificial intelligence; XGBoost: eXtreme Gradient Boosting.

Declarations

- Funding: This research received no specific funding.
- Competing interests: The authors declare no competing interests.
- Ethics approval and consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Availability of data and materials: Not applicable.
- Materials availability: Not applicable.
- Code availability: The complete source is available at <https://github.com/Je-RICO-h/Lightweight-Autonomous-Security-Agents-for-Operator-Workstations>.
- Authors’ contributions: E.P.: conceptualisation, methodology, software, measurements, formal analysis, writing original draft. A.H. and R.L.: conceptualisation,

supervision, writing review and editing. All authors read and approved the final manuscript.

- **Use of AI tools:** The authors used a large language model (Claude, Gemini) as an assistant while revising the text of the manuscript and while developing parts of the analysis scripts. The research questions, the experimental design and all measurements are the authors' own. Every reported number is computed by the published scripts from the measured data, and all derivations, tables and figures were checked by the authors and can be reproduced with the code repository. The authors take full responsibility for the content of this article.
- **Prior publication:** This article extends the conference paper [11] (E. Pál, R. Lakatos, and A. Hajdu, "Accelerated Keystroke Dynamics: An XGBoost Approach to Continuous Biometric Authentication"). While the prior conference paper focused on establishing the core XGBoost keystroke-dynamics classifier and evaluating its baseline performance against deep learning models, this journal article significantly expands the scope and application of the research. The main novel contributions of this extended work include:
 - The formalization of a lightweight, autonomous edge security agent architecture, integrating sensory, cognitive, and action modules.
 - The development of a state-dependent, graded decision policy (allow, challenge, alert, critical) utilizing dynamic thresholds derived from ROC and Equal Error Rate characteristics.
 - The introduction of a SHAP-based explanation channel that provides transparent, human-readable justifications for interventions, aligning with PEP-AI (Precise, Explainable, Provable AI) principles and a human-in-the-loop design.
 - The inclusion of an energy-oriented evaluation framework, demonstrating the agent's efficiency in compute effort, memory footprint, and adaptation costs in the context of Green AI for critical infrastructures.

Acknowledgements. This research was supported by the Faculty of Informatics at the University of Debrecen.

Appendix A Raw features captured by the sensory module

Table A1 Raw features recorded for every keystroke by the sensory module [11].

Feature	Description
Key identifier	Letter, number or special key (e.g., <i>Shift</i> , <i>Space</i>)
Relative start time	Press time in seconds from the start of the data collection
Absolute start timestamp	Unix timestamp of the press (bookkeeping only, not a model input)
Keys per second	Cumulative typing speed
Combination	Keys held simultaneously with the current key
Error flag	Whether the key was later erased with <i>Backspace</i>
Seek time	Interval between the release of the previous key and the press of the current key
Error rate	Running average of errors
Accuracy	Running average of typing accuracy
Relative end time	Release time in seconds from the start of the data collection
Absolute end timestamp	Unix timestamp of the release (bookkeeping only, not a model input)
Hold time	Duration between press and release of the key

Appendix B Editable action hooks

The response of the agent at each level is defined by three functions that a deployment can edit without modifying the policy. The default implementation is:

References

- [1] Corrado C, Haskel J, Iommi M, Jona-Lasinio C. The value of data in digital-based business models: Measurement and economic policy implications. Paris: OECD Publishing; 2022. OECD Economics Department Working Papers No. 1723.
- [2] Verizon Business. 2025 Data Breach Investigations Report [Internet]; 2025 [cited October 2025]. Available from: <https://www.verizon.com/business/resources/reports/dbir/>.
- [3] Khoshaba FS, Askar S, Hamad SA, Maghdid SS. Cyber Security Challenges in Industry 4.0: A Review. Indonesian Journal of Computer Science. 2024;13(2):2360–2372. <https://doi.org/10.33022/ijcs.v13i2.3848>.
- [4] Wang W, Lu Z. Cyber security in the Smart Grid: Survey and challenges. Computer Networks. 2013;57(5):1344–1371. <https://doi.org/10.1016/j.comnet.2012.12.017>.

- [5] Stouffer K, Pease M, Tang C, Zimmerman T, Pillitteri V, Lightman S, et al. Guide to Operational Technology (OT) Security. National Institute of Standards and Technology; 2023. NIST Special Publication 800-82r3.
- [6] Electricity Information Sharing and Analysis Center (E-ISAC), SANS Industrial Control Systems. Analysis of the Cyber Attack on the Ukrainian Power Grid: Defense Use Case. Washington, DC: E-ISAC; 2016.
- [7] Patel VM, Chellappa R, Chandra D, Barbello B. Continuous User Authentication on Mobile Devices: Recent progress and remaining challenges. *IEEE Signal Processing Magazine*. 2016;33(4):49–61. <https://doi.org/10.1109/MSP.2016.2555335>.
- [8] Joyce R, Gupta G. Identity authentication based on keystroke latencies. *Communications of the ACM*. 1990;33(2):168–176. <https://doi.org/10.1145/75577.75582>.
- [9] Monroe F, Rubin AD. Authentication via keystroke dynamics. In: *Proceedings of the 4th ACM Conference on Computer and Communications Security*. Zürich, Switzerland; 1997. p. 48–56.
- [10] Shadman R, Wahab AA, Manno M, Lukaszewski M, Hou D, Hussain F. Keystroke Dynamics: Concepts, Techniques, and Applications. *ACM Computing Surveys*. 2025;57(11):1–35. <https://doi.org/10.1145/3733103>.
- [11] Pál E, Hajdu A, Lakatos R. Accelerated Keystroke Dynamics: An XGBoost Approach to Continuous Biometric Authentication. In: *2026 IEEE 4th Conference on Information Technology and Data Science (CITDS 2026)*. Debrecen, Hungary: IEEE; 2026. 27–28 August 2026.
- [12] Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA; 2016. p. 785–794.
- [13] Acien A, Morales A, Monaco JV, Vera-Rodriguez R, Fierrez J. TypeNet: Deep Learning Keystroke Biometrics. *IEEE Transactions on Biometrics, Behavior, and Identity Science*. 2022;4(1):57–70. <https://doi.org/10.1109/TBIOM.2021.3112540>.
- [14] Russell S, Norvig P. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson; 2021.
- [15] Wooldridge M. *An Introduction to MultiAgent Systems*. 2nd ed. Chichester, UK: John Wiley & Sons; 2009.
- [16] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*. 2016;3(5):637–646. <https://doi.org/10.1109/>

- [17] Schwartz R, Dodge J, Smith NA, Etzioni O. Green AI. *Communications of the ACM*. 2020;63(12):54–63. <https://doi.org/10.1145/3381831>.
- [18] Lundberg SM, Lee SI. A Unified Approach to Interpreting Model Predictions. In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*; 2017. p. 4765–4774.
- [19] Lundberg SM, Erion G, Chen H, DeGrave A, Prutkin JM, Nair B, et al. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*. 2020;2(1):56–67. <https://doi.org/10.1038/s42256-019-0138-9>.
- [20] Parasuraman R, Sheridan TB, Wickens CD. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*. 2000;30(3):286–297. <https://doi.org/10.1109/3468.844354>.
- [21] Amershi S, Cakmak M, Knox WB, Kulesza T. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine*. 2014;35(4):105–120. <https://doi.org/10.1609/aimag.v35i4.2513>.
- [22] Killourhy KS, Maxion RA. Comparing anomaly-detection algorithms for keystroke dynamics. In: *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; 2009. p. 125–134.
- [23] Chang HC, Li J, Wu CS, Stamp M. Machine Learning and Deep Learning for Fixed-Text Keystroke Dynamics. In: Stamp M, Visaggio CA, Mercaldo F, Di Troia F, editors. *Artificial Intelligence for Cybersecurity. Advances in Information Security*. Cham: Springer; 2022. p. 309–329.
- [24] Kuzminykh I, Mathur S, Ghita B. Performance Analysis of Free Text Keystroke Authentication Using XGBoost. In: *Advances in Computer Science for Engineering and Education VI (ICCSEEA 2023)*. vol. 181 of *Lecture Notes on Data Engineering and Communications Technologies*. Cham: Springer; 2023. p. 429–439.
- [25] Kaluarachchi N, Kandanaarachchi S, Moore K, Arakala A. DEFT: A New Distance-Based Feature Set for Keystroke Dynamics. In: *2023 International Conference of the Biometrics Special Interest Group (BIOSIG)*. Darmstadt, Germany: IEEE; 2023. p. 1–6.
- [26] Bentéjac C, Csörgő A, Martínez-Muñoz G. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*. 2021;54(3):1937–1967. <https://doi.org/10.1007/s10462-020-09896-5>.

- [27] Barredo Arrieta A, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*. 2020;58:82–115. <https://doi.org/10.1016/j.inffus.2019.12.012>.
- [28] Biggio B, Roli F. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*. 2018;84:317–331. <https://doi.org/10.1016/j.patcog.2018.07.023>.
- [29] Palmer M. pynput: Send and control input devices [Internet]; 2026. Available from: <https://pynput.readthedocs.io/en/latest/>. Python library to control and monitor input devices.
- [30] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011;12:2825–2830.
- [31] Fawcett T. An introduction to ROC analysis. *Pattern Recognition Letters*. 2006;27(8):861–874. <https://doi.org/10.1016/j.patrec.2005.10.010>.
- [32] Pál E, Lakatos R, Hajdu A. Lightweight Autonomous Security Agents for Operator Workstations [GitHub repository]; 2026. Available from: <https://github.com/Je-RICO-h/Lightweight-Autonomous-Security-Agents-for-Operator-Workstations>.
- [33] Pál E, Lakatos R, Hajdu A. Mesterséges intelligencia alapú biometrikus azonosítás [GitHub repository]; 2026. Available from: <https://github.com/Je-RICO-h/Mesterseges-intelligencia-alapu-biometrikus-azonositas>.