International Journal of Engineering Research and Science & Technology
ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

# CENTRALIZED LABORATORY BROWSER ACTIVITY MONITORING SYSTEM

**Kilambi Sai Vignesh**
PG MCA Scholar, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India-534005
vigneshkilambi123@gmail.com

**Banisetti Jyothi Satya Bhavani**
Assistant Professor, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India-534005
jyothiganesh006@gmail.com

**Musunuru Ratnakar**
HOD, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India-534005

***Abstract— Centralized supervision of laboratory computers is difficult when an instructor must inspect each workstation physically. This paper presents a Centralized Laboratory Browser Activity Monitoring System that observes browser-window titles on distributed laboratory computers and reports flagged activity to a single administrative dashboard. A lightweight Python agent scans visible titles every 15 s, identifies configured applications such as YouTube and ChatGPT, and sends JSON heartbeat messages through a local-area network to a Flask REST backend. The backend updates machine state, creates high-severity alerts, maintains durations, and marks a client offline when no heartbeat is received for 60 s. A React.js dashboard provides live statistics, multi-system status, alert history, per-machine drill-down, charts, agent download, and responsive mobile access. Evaluation on five laboratory workstations reported 12-15 s alert latency, backend response below 50 ms, 1.2 s dashboard loading, agent use below 1% CPU and 28 MB memory, and 100% alert accuracy for the configured activities. The system therefore provides a low-cost, explainable, and minimally intrusive approach for institutional laboratory supervision without requiring commercial endpoint software or cloud infrastructure.***

***Keywords— browser activity monitoring, computer laboratory, Flask REST API, React dashboard, Python monitoring agent, real-time alerts, offline detection.***

## I. INTRODUCTION

Internet-connected laboratory computers support programming, research, tutorials, and collaborative learning, but they also provide immediate access to entertainment, social media, and generative-AI services. Empirical studies have repeatedly shown that off-task laptop and web use can divide attention and reduce classroom learning [1]-[8]. In large practical sessions, manual screen inspection is intermittent and creates no historical record, while strict URL blocking may also prevent legitimate academic use. A monitoring-first approach can provide awareness and accountability without indiscriminately restricting access.

### *A. Problem Context and Motivation*

The project targets educational computer laboratories in which a faculty member or administrator must supervise several workstations simultaneously. Existing commercial endpoint suites are expensive and intrusive, and network-level filtering cannot reliably explain which application is active on a specific machine. The proposed solution uses a small endpoint agent, a centralized Flask service, and a responsive React interface over the institutional LAN. It reports only matching window-title information and heartbeat status, thereby limiting the collected data compared with keylogging or continuous screenshot capture.

### *B. Objectives and Contributions*

The system contributes an end-to-end real-time pipeline for browser-title acquisition, keyword-based detection, heartbeat transmission, alert-state management, offline detection, dashboard visualization, and per-system history. It also supports dynamic workstation registration, downloadable agent deployment, mobile access, and visual summaries. The deployed prototype monitored five concurrent systems and was designed to extend to additional applications and machines without changing the core architecture.

## II. RELATED WORK

### *A. Digital Distraction in Learning Environments*

Classroom studies report that unrestricted laptop use frequently includes non-academic activity and that such multitasking can impair attention, note-taking, retention, and assessment performance [1]-[7]. These findings motivate institutional tools that make off-task behavior visible while preserving access for valid learning activities. Unlike blanket prohibition, activity monitoring supports context-sensitive intervention by the instructor.

### *B. Endpoint and Network Monitoring*

Network intrusion and event-monitoring research established the value of continuous observation, event generation, centralized state, and timely notification [9]-[17]. Systems such as Bro/Zeek and Snort demonstrated efficient real-time analysis and policy-based alerts, while anomaly-detection surveys describe the trade-offs between rule-driven and learned monitoring [11]-[16]. The present project applies the same event-oriented principle at a simpler educational endpoint level: it observes browser titles rather than packets and applies an explicit keyword policy rather than a statistical model.

### *C. Research Gap*

Enterprise products provide broad surveillance capabilities but impose licensing, deployment, privacy, and administration costs. Research on classroom distraction typically measures learning impact rather than delivering a lightweight institutional tool. The identified gap is therefore a transparent, open-stack system that can be installed on ordinary lab machines, operate entirely within a LAN, show live browser-

related alerts, preserve alert duration and history, and remain usable from desktop or mobile browsers.

## III. MATERIALS AND METHODS

### A. Data Acquisition and Activity Dataset

No public benchmark dataset or machine-learning training set is used. The operational dataset is generated continuously by monitoring agents running on laboratory computers. Each heartbeat contains a system identifier, the matched application list, and a timestamp. The backend maintains three evolving collections: registered systems, alert records, and activity logs. Live evaluation used five machines identified as Lab-PC-07, Lab-PC-12, Lab-PC-18, Lab-PC-21, and Lab-PC-01.

**TABLE I. ACTIVITY RECORD REPRESENTATION**

| Field | Example | Purpose |
|---|---|---|
| system_id | Lab-PC-18 | Unique client identity |
| apps | YouTube, ChatGPT | Detected configured activities |
| timestamp | 12:24 PM | Heartbeat and alert timing |
| status | Online / Alert / Offline | Current workstation state |

### B. Client-Side Monitoring Agent

Each laboratory workstation executes client_monitor.py. The agent enumerates visible window titles with pygetwindow, normalizes the text, compares it with a configurable list of flagged terms, constructs a JSON payload, and submits the heartbeat to the central server through HTTP POST. The default interval is 15 s. The agent depends only on Python, pygetwindow, and requests, allowing simple installation and a small runtime footprint.

### C. Browser-Activity Detection Rule

For a normalized window title w and configured activity keyword a, the binary detection decision is

$$D(w,a) = \begin{cases} 1, & \text{if } a \in w; \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

A positive decision creates or updates an active alert. Repeated detections of the same application on the same workstation update duration rather than generating duplicate records. When the keyword disappears from later heartbeats, the alert is cleared and its end time is recorded.

### D. Backend Processing and Offline Detection

The Flask service exposes endpoints for heartbeat submission, system retrieval, alert retrieval, and workstation management. Incoming payloads update the in-memory state and alert history. The backend compares the current time with each last_seen timestamp; if the difference exceeds 60 s, the workstation is marked Offline. The React frontend polls the REST endpoints and rerenders individual cards, lists, and charts without a full page refresh.

### E. System Architecture

The architecture contains a client tier with N monitoring agents, an application tier containing the Flask REST API, heartbeat processor, alert engine, state manager, and periodic offline checker, and a presentation tier containing the React live-status dashboard, alerts panel, system detail view, activity logs, and system-management controls. All communication occurs over the institutional LAN, while state is retained in in-memory or JSON structures.

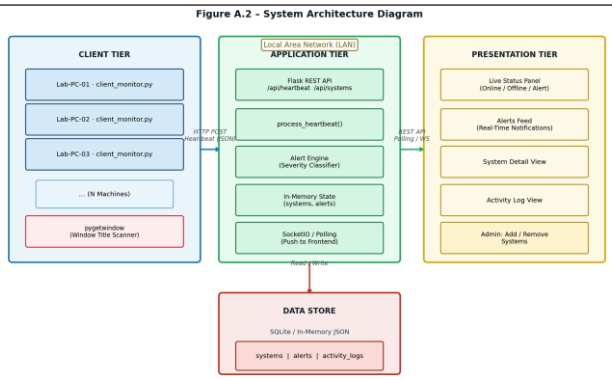



Fig. 1. Client-server architecture of the laboratory browser activity monitoring system.

### F. Performance Formulation

Alert latency is measured from the instant the configured activity is observed by the agent to the instant the corresponding alert becomes visible on the dashboard:

$$Lalert = tdashboard - tdetect \quad (2)$$

For N evaluated detection events, the documented alert accuracy is calculated as

$$\text{Accuracy} = \frac{Ncorrect}{Ntotal} \times 100\% \quad (3)$$

The online-system ratio displayed by the dashboard is

$$Ronline = \frac{Nonline}{Nregistered} \times 100\% \quad (4)$$

Precision, recall, and F1-score are not reported because the project document does not provide false-positive and false-negative counts. The paper therefore avoids introducing unsupported classifier metrics.

### G. Alert Lifecycle and Duplicate Suppression

An alert is modeled as a stateful event rather than a new message for every heartbeat. The pair formed by the workstation identifier and detected application acts as the logical event key. When the same key appears in consecutive reports, the backend retains the original start time and updates the current duration. When the application is no longer present, the event receives an end time and is moved to historical status. This mechanism prevents a fifteen-second reporting cycle from producing hundreds of duplicate entries and allows the dashboard to distinguish a sustained activity from repeated short activities. The same lifecycle also supports meaningful instructor review because each record represents one continuous interval of use.

### H. Dashboard Synchronization and Failure Handling

The React interface periodically requests system and alert data rather than maintaining a direct dependency on each laboratory workstation. This separation keeps clients simple and centralizes validation in the Flask service. A malformed or incomplete heartbeat can be rejected without affecting the dashboard, while temporary frontend unavailability does not stop agents from reporting to the backend. If a client loses power, closes the agent, or becomes disconnected from the LAN, the last-seen timestamp eventually exceeds the sixty-second threshold and the state changes to Offline. If the server is unavailable, the agent should continue its polling loop and retry on the next interval instead of terminating. These behaviors provide graceful degradation for the common failures expected in an instructional laboratory.

### I. Privacy-Aware Monitoring Policy

The prototype intentionally limits collection to visible window titles, system identifiers, timestamps, and configured application matches. It does not record keystrokes, capture

screens, inspect document content, or transmit complete browsing histories. Even this reduced dataset can reveal user activity and must therefore be governed by an institutional policy. Students should be informed that monitoring is active, the purpose should be limited to supervised laboratory sessions, access to records should be restricted, and historical data should be retained only for a defined period. A production deployment should also support session-based enabling, policy-specific keyword lists, secure transport, authenticated administrator access, and an audit trail showing who reviewed or exported records.

## IV. RESULTS AND DISCUSSION

### *A. Real-Time Dashboard and Multi-System View*

The main dashboard summarizes five registered systems, active systems, active alerts, and detected processes. A bar chart presents system, online, and alert counts, while donut charts summarize system status and application distribution. The centralized systems view lists each machine with Online, Offline, or Alert status and shows application badges, allowing an administrator to identify affected workstations without moving through the laboratory.

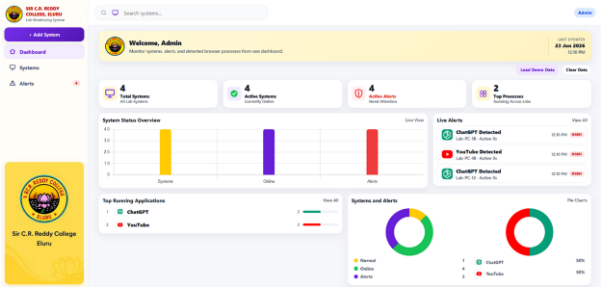


Fig. 2. Main dashboard with live statistics, bar chart, alert feed, and application distribution.

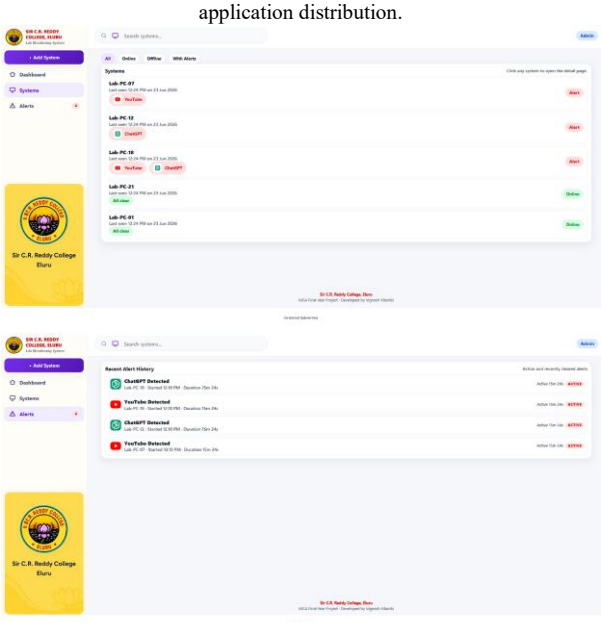


Fig. 3. Centralized workstation status and recent alert-history views.

### *B. Per-System Monitoring and Offline Detection*

The Lab-PC-18 detail page displays current alerts, recent alert history, detected applications, monitoring history, last update, and system summary. During testing, YouTube and ChatGPT remained active for approximately 18 min and were represented as separate high-severity events. When heartbeat transmission stopped, the backend changed the workstation state to Offline after the configured timeout. This behavior distinguishes active misuse from an unavailable monitoring agent and prevents stale status information.

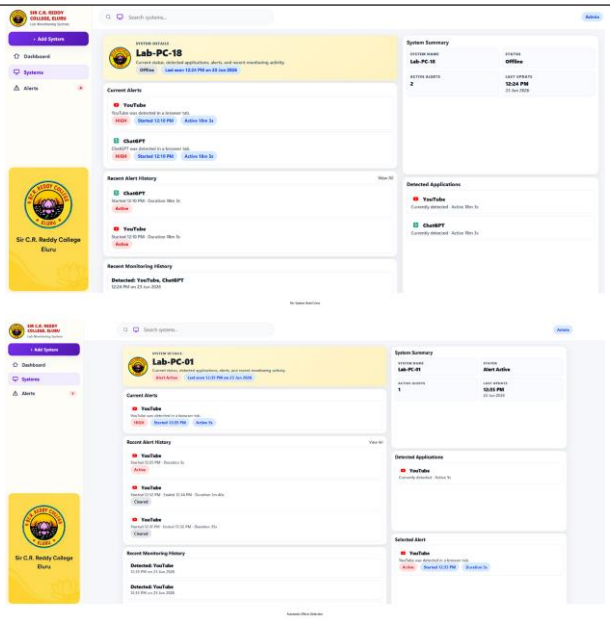


Fig. 4. Per-system alert details and automatic offline-state transition.

### *C. Quantitative Performance*

Live testing reported an end-to-end alert latency of 12-15 s against a target below 30 s. GET /api/systems responded in less than 50 ms against a 500 ms limit, and the dashboard loaded in 1.2 s against a 3 s target. The client agent consumed less than 1% CPU and 28 MB of memory, and five agents operated concurrently. All configured detection events were reported correctly, producing a documented alert accuracy of 100%.

Every measured value satisfied the corresponding project target. Alert delivery used approximately one-half of the permitted latency, the API response remained far below its upper bound, and the dashboard loaded with substantial time margin. Endpoint CPU and memory consumption were also below the specified limits, which is important because the agent shares each workstation with programming tools, browsers, and academic software. The concurrent-agent result confirms coordinated reporting from the five test systems, while the accuracy result confirms correct output for the documented YouTube and ChatGPT scenarios. Together, the measurements indicate that responsiveness was governed mainly by the intentional heartbeat interval rather than by server processing or interface rendering. Shorter intervals could reduce visible delay, but they would increase request frequency and should be selected according to laboratory size and network capacity.

**TABLE II. REPORTED OPERATIONAL PERFORMANCE**

| Metric | Measured | Target |
|---|---|---|
| Alert latency | 12-15 s | <30 s |
| Backend response | <50 ms | <500 ms |
| Dashboard load | 1.2 s | <3 s |
| Agent CPU | <1% | <5% |
| Agent memory | 28 MB | <100 MB |
| Concurrent agents | 5 | 5 |
| Alert accuracy | 100% | 100% |

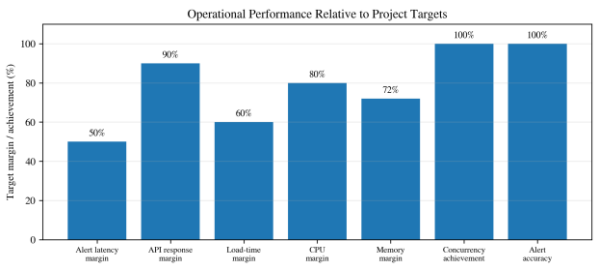


Fig. 5. Operational margin or target achievement derived from the reported measurements.

*D. Mobile Access, Usability, and Privacy*

The responsive interface was verified on Android Chrome through LAN Wi-Fi and reorganized dashboard elements for narrow screens. Immediate status badges and live alert durations provide actionable feedback. The title-scanning design is less invasive than keylogging or continuous screenshots; however, window titles may still expose sensitive information. Production deployment should include administrator authentication, encrypted transport, role-based access, explicit laboratory policy, retention limits, and controlled audit access.

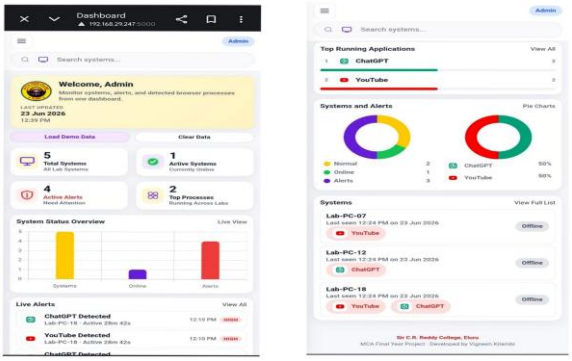



Fig. 6. Mobile-responsive dashboard viewed through the laboratory LAN.

*E. Discussion and Limitations*

The results demonstrate that the system satisfies its intended demonstration-scale requirements with low latency and negligible endpoint overhead. Its transparent keyword rules are easy to audit and extend. Limitations include dependence on visible window titles, possible misses when applications obscure their names, absence of persistent relational storage, no user authentication in the prototype, and evaluation on only five workstations and two configured applications. The 100% accuracy therefore describes the documented test scenarios and should not be generalized to every browser, website, or evasion technique.

*F. Comparison with Alternative Supervision Approaches*

Manual supervision provides direct human judgement but is intermittent, difficult to scale, and produces no searchable history. Network-level filtering can prevent access, yet it may also block legitimate instructional use and usually cannot explain which workstation actively displayed a particular application. Commercial endpoint products offer richer features but introduce licensing costs, complex deployment, and privacy concerns. The proposed system occupies a narrower position: it does not block access or inspect content, but provides timely, workstation-specific awareness through an open and locally deployable stack. This scope is appropriate for a teaching laboratory in which an instructor needs situational visibility while retaining discretion over whether a detected activity is actually inappropriate in context.

*G. Reliability and Scalability Interpretation*

The five-agent experiment confirms concurrent operation at the documented laboratory scale, while the backend requirement anticipates growth to approximately fifty clients. Each agent processes only the current set of visible titles and sends a compact JSON message, so endpoint resource use remains small. On the server, heartbeat handling and status lookup are lightweight at this scale, but in-memory dictionaries and frequent polling become less suitable as laboratories, users, and historical records grow. Persistent storage, indexed queries, pagination, event retention rules, and a production web server would be required before multi-laboratory deployment. The current measurements should consequently be interpreted as evidence of feasibility and responsiveness, not as a formal high-load benchmark.

*H. Validity of the Reported Evaluation*

The evaluation used real laboratory workstations and verified the complete route from browser-title detection to dashboard presentation, which strengthens practical relevance. However, the test set was limited to two configured applications, a single institutional network, and a small number of clients. The reported one-hundred-percent alert accuracy indicates that all documented scenarios completed correctly; it does not establish false-positive behavior for unrelated titles containing similar words, false negatives under browser changes, or robustness against deliberate evasion. A stronger study should preserve event-level ground truth, repeat trials at different loads, and report distributions for latency rather than only an observed range.

*I. Alert-Duration and Activity Interpretation*

Duration tracking adds context that a simple detection counter cannot provide. A brief appearance of a flagged title may result from an instructor demonstration, a link preview, or an accidental tab switch, whereas a sustained event indicates continuing use that may require attention. The dashboard therefore presents the original start time, active duration, and cleared state together. An administrator can use these values to prioritize long-running alerts, compare repeated cycles on the same workstation, and distinguish current activity from historical records. The design does not automatically punish or block a workstation; it supplies evidence that a responsible staff member interprets in relation to the class task and institutional policy.

*J. Administrative Workflow and Operational Use*

A typical deployment begins when the administrator registers laboratory machine identifiers and downloads the monitoring agent from the dashboard. The script is copied to each workstation and configured with the server address. Once launched, every agent sends its first heartbeat and appears in the systems view. During a session, the administrator observes the summary cards, checks the live-alert panel, and opens a detailed page when an alert requires investigation. At the end of the session, historical entries can be reviewed for repeated or prolonged activities. This workflow avoids separate transfer utilities and keeps system enrollment, monitoring, and review within one interface, which reduces the technical effort required from laboratory staff.

*K. Security and Abuse Resistance*

The prototype focuses on functional monitoring and does not claim resistance against a determined adversary. A user could terminate the agent, rename or hide a window title, disconnect from the LAN, or attempt to submit a forged heartbeat. Offline detection reveals some agent interruptions but cannot prove why reporting stopped. A secure version should sign or authenticate heartbeat requests, restrict API origins, validate machine enrollment tokens, protect administrative routes, and record configuration changes. The server should use HTTPS even within the LAN when personal or disciplinary records are stored. Client installation and service control should be managed by authorized staff, while alerts should never be treated as conclusive evidence without contextual review.

*L. Maintainability and Policy Configuration*

The modular separation between title scanning, REST processing, alert-state management, and dashboard rendering

supports incremental maintenance. New applications can be introduced through a controlled policy list without redesigning the communication pipeline. Different courses may require different rules; for example, video platforms or AI tools may be permitted during one session and flagged during another. A future configuration page should therefore support named policies, scheduled activation, workstation groups, case-insensitive matching, exact or partial match options, and administrator notes. Versioning these policies would make historical alerts interpretable because reviewers could determine which rule set was active at the time of detection.

## V. CONCLUSION AND FUTURE SCOPE

This paper presented a centralized laboratory browser activity monitoring system based on a Python endpoint agent, Flask REST backend, React.js dashboard, and LAN communication. The solution detects configured browser activities, produces high-severity alerts, records duration and history, displays multi-system status, and identifies offline machines. The documented evaluation confirmed timely detection, responsive API and dashboard behavior, low CPU and memory use, mobile access, and correct operation across five concurrent laboratory computers.

### *A. Future Enhancements*

Future development should replace in-memory state with SQLite or PostgreSQL, add secure administrator authentication and role-based authorization, deploy HTTPS through a reverse proxy, and provide configurable policy groups for different laboratory sessions. Email or SMS escalation, scheduled reports, CSV/PDF export, and richer audit search can improve administrative usefulness. A browser extension or operating-system event interface could provide reliable URL-level context where permitted, while preserving a privacy-minimizing collection policy.

### *B. Evaluation and Deployment Roadmap*

A larger evaluation should include 30-50 workstations, several browsers, additional flagged and permitted applications, network congestion, agent interruption, and deliberate title-obfuscation scenarios. Ground-truth event logs would permit precision, recall, and F1-score calculation in addition to latency and resource use. Longitudinal classroom deployment should also measure false-alert burden, instructor response time, student acceptance, privacy perception, and whether monitoring reduces off-task browsing without restricting legitimate academic access.

### *C. Institutional Integration*

For routine operation, monitoring should be aligned with timetables so that policies are activated only during designated laboratory sessions. Administrators should be able to assign workstation groups, define permitted and flagged applications per course, temporarily acknowledge justified alerts, and annotate events with an instructional decision. These controls would reduce unnecessary escalation when services such as YouTube or generative AI are intentionally used for learning. Integration with the institution directory could provide role-based access without exposing student credentials to the monitoring agent, while scheduled summaries could give department heads aggregate trends without revealing excessive individual detail.

### *D. Research Extensions*

Future research can compare window-title rules with browser extensions, operating-system accessibility interfaces, DNS or proxy logs, and privacy-preserving combinations of these signals. A labeled event dataset could support a supervised classifier for ambiguous titles, but any learned method should be evaluated against the transparent rule baseline and should expose the evidence behind each alert. Additional study is also needed on student acceptance, perceived fairness, policy clarity, and the effect of monitoring on legitimate exploration. The most useful extension is not necessarily the most intrusive one; an effective institutional system should balance detection quality, explainability, deployment cost, and proportional data collection.

### *E. Maintenance and Operational Procedure*

Sustainable deployment requires more than installing the initial software. The server address, monitored keyword list, polling interval, offline timeout, and retention period should be documented and reviewed each semester. Administrators need a backup and recovery procedure for persistent records, while agents should be updated through a controlled versioning process so that all machines run compatible code. Health indicators can identify clients that repeatedly disconnect or report malformed data. A small maintenance checklist covering server startup, LAN reachability, time synchronization, agent availability, policy activation, and dashboard verification would reduce failures during practical examinations and scheduled laboratory sessions.

### *F. Expected Educational and Administrative Impact*

The system is designed to improve awareness rather than replace human supervision. Central visibility can reduce the time faculty spend walking between machines, provide a consistent record of prolonged off-task activity, and help staff identify technical failures such as stopped agents or disconnected workstations. Because the tool does not automatically block websites, it preserves legitimate access when the same platforms are needed for tutorials, research, programming assistance, or media-based instruction. Its effectiveness ultimately depends on transparent rules, proportionate use, and timely human interpretation. A carefully governed deployment can therefore support classroom focus while retaining the flexibility required in modern computer-based learning.

### *G. Administrator Training and User Guidance*

Although the dashboard is intended for non-technical staff, consistent use requires a brief operating guide. Administrators should understand the meaning of Online, Alert, and Offline states; the difference between an active event and a historical event; and the fact that a title match is an indicator rather than a final disciplinary judgement. Training should cover client registration, agent distribution, server-address configuration, policy selection, alert acknowledgement, and safe shutdown. A visible system-health checklist can confirm that the backend is reachable, all expected clients are reporting, workstation clocks are synchronized, and the correct policy is active before a laboratory session begins. Clear guidance reduces misinterpretation and supports repeatable operation across different faculty members.

### *H. Reporting, Retention, and Audit Review*

Operational records become more useful when they can be reviewed at suitable levels of detail. The live dashboard supports immediate response, while daily or session-level summaries can show active workstations, number and duration of alerts, repeated applications, offline periods, and unresolved technical failures. Reports should separate aggregate instructional trends from identifiable workstation histories. A configurable retention period can remove old records automatically, and export should be limited to authorized users

with a stated purpose. When an alert is reviewed, the administrator may record a note such as legitimate course use, student guidance provided, false title match, or technical test. Such annotations preserve context and prevent later readers from treating every recorded match as equivalent misconduct.

*I. Multi-Laboratory Deployment Scenario*

The present implementation operates within one LAN, but the same logical design can support several laboratories. Each client can include a laboratory or workstation-group identifier in addition to the system name, allowing the dashboard to filter by room, course, or scheduled session. A shared backend can provide department-level visibility, whereas local relay servers can be used where network segments are isolated. Multi-laboratory deployment requires stronger authentication, persistent storage, centralized configuration, and protection against identifier collisions. It should also support delegated administration so that laboratory assistants can view their assigned room while department administrators access aggregate status. These extensions preserve the agent-backend-dashboard separation while improving institutional scalability.

*J. Final Design Perspective*

The project demonstrates that useful real-time supervision can be achieved without deep packet inspection, continuous screenshots, or commercial endpoint suites. Its main strength is the direct relationship between a visible browser-title match, a compact heartbeat, a stateful alert, and an understandable dashboard entry. The same simplicity makes limitations easy to identify and guides future development toward secure persistence, configurable policies, broader evaluation, and proportionate governance. The system should therefore be viewed as a practical foundation for privacy-aware laboratory awareness rather than an invisible surveillance mechanism. Maintaining this boundary is essential for technical credibility, institutional acceptance, and responsible educational use.

REFERENCES

[1] C. B. Fried, "In-class laptop use and its effects on student learning," Computers & Education, vol. 50, no. 3, pp. 906-914, 2008, doi: 10.1016/j.compedu.2006.09.006.

[2] F. Sana, T. Weston, and N. J. Cepeda, "Laptop multitasking hinders classroom learning for both users and nearby peers," Computers & Education, vol. 62, pp. 24-31, 2013, doi: 10.1016/j.compedu.2012.10.003.

[3] E. Wood, L. Zivcakova, P. Gentile, K. Archer, D. De Pasquale, and A. Nosko, "Examining the impact of off-task multi-tasking with technology on real-time classroom learning," Computers & Education, vol. 58, no. 1, pp. 365-374, 2012, doi: 10.1016/j.compedu.2011.08.029.

[4] E. F. Risko, D. Buchanan, S. Medimorec, and A. Kingstone, "Everyday attention: Mind wandering and computer use during lectures," Computers & Education, vol. 68, pp. 275-283, 2013, doi: 10.1016/j.compedu.2013.05.001.

[5] E. D. Ragan, S. R. Jennings, J. D. Massey, and P. E. Doolittle, "Unregulated use of laptops over time in large lecture classes," Computers & Education, vol. 78, pp. 78-86, 2014, doi: 10.1016/j.compedu.2014.05.002.

[6] B. M. Waite, R. Lindberg, B. Ernst, L. L. Bowman, and L. E. Levine, "Off-task multitasking, note-taking and lower- and higher-order classroom learning," Computers & Education, vol. 120, pp. 98-111, 2018, doi: 10.1016/j.compedu.2018.01.007.

[7] S. M. Ravizza, M. G. Uitvlugt, and K. M. Fenn, "Logged in and zoned out: How laptop Internet use relates to classroom learning," Psychological Science, vol. 28, no. 2, pp. 171-180, 2017, doi: 10.1177/0956797616677314.

[8] N. M. Aguilar-Roca, A. F. Williams, and D. K. O'Dowd, "The impact of laptop-free zones on student performance and attitudes in large lectures," Computers & Education, vol. 59, no. 4, pp. 1300-1308, 2012, doi: 10.1016/j.compedu.2012.05.002.

[9] D. E. Denning, "An intrusion-detection model," IEEE Transactions on Software Engineering, vol. SE-13, no. 2, pp. 222-232, 1987, doi: 10.1109/TSE.1987.232894.

[10] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," IEEE Network, vol. 8, no. 3, pp. 26-41, 1994, doi: 10.1109/65.283931.

[11] V. Paxson, "Bro: A system for detecting network intruders in real-time," in Proc. 7th USENIX Security Symposium, San Antonio, TX, USA, 1998, pp. 31-51.

[12] M. Roesch, "Snort: Lightweight intrusion detection for networks," in Proc. 13th USENIX Conference on System Administration, Seattle, WA, USA, 1999, pp. 229-238.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys, vol. 41, no. 3, Art. no. 15, 2009, doi: 10.1145/1541880.1541882.

[14] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in Proc. IEEE Symposium on Security and Privacy, 2010, pp. 305-316, doi: 10.1109/SP.2010.25.

[15] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," Journal of Network and Computer Applications, vol. 36, no. 1, pp. 16-24, 2013, doi: 10.1016/j.jnca.2012.09.004.

[16] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153-1176, 2016, doi: 10.1109/COMST.2015.2494502.

[17] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in Proc. Network and Distributed System Security Symposium, 2018.

[18] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," Computers & Security, vol. 86, pp. 147-167, 2019, doi: 10.1016/j.cose.2019.06.005.

[19] Q. Chen and Z. Yan, "Does multitasking with mobile phones affect learning? A review," Computers in Human Behavior, vol. 54, pp. 34-42, 2016, doi: 10.1016/j.chb.2015.07.047.

[20] J. H. Kuznekoff, S. Munz, and S. Titsworth, "Mobile phones in the classroom: Examining the effects of texting, Twitter, and message content on student learning," Communication Education, vol. 64, no. 3, pp. 344-365, 2015, doi: 10.1080/03634523.2015.1038727.

[21] H. Hembrooke and G. Gay, "The laptop and the lecture: The effects of multitasking in learning environments," Journal of Computing in Higher Education, vol. 15, pp. 46-64, 2003, doi: 10.1007/BF02940852.

[22] D. M. Kraushaar and D. C. Novak, "Examining the effects of student multitasking with laptops during the lecture," Journal of Information Systems Education, vol. 21, no. 2, pp. 241-251, 2010.

[23] S. Bellur, K. L. Nowak, and K. S. Hull, "Make it our time: In class multitaskers have lower academic performance," Computers in Human Behavior, vol. 53, pp. 63-70, 2015, doi: 10.1016/j.chb.2015.06.027.

[24] S. Few, Information Dashboard Design: Displaying Data for At-a-Glance Monitoring, 2nd ed. Burlingame, CA, USA: Analytics Press, 2013.

[25] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in Proc. IEEE Symposium on Visual Languages, 1996, pp. 336-343, doi: 10.1109/VL.1996.545307.

[26] J. Heer and B. Shneiderman, "Interactive dynamics for visual analysis," Communications of the ACM, vol. 55, no. 4, pp. 45-54, 2012, doi: 10.1145/2133806.2133821.

[27] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, CA, USA, 2000.

[28] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.

[29] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2020.

[30] National Institute of Standards and Technology, Guide to Computer Security Log Management, NIST Special Publication 800-92, Gaithersburg, MD, USA, 2006, doi: 10.6028/NIST.SP.800-92.