Privacy Intrusion Detection System: A Real-Time Behavior-Based Approach for User-Centric Endpoint Protection

R. Nithin Reddy[1], Venkata Surya, Suraj Nadagoudra[2], Aadith Sugeesh[3],
Saikumar Krishna Joshi[4], Saniya Abid Kanza[5]
[1,2,3,4,5] *Department of Forensic Science, JAIN (Deemed-to-be University), Bengaluru, India*

***Abstract*—The proliferation of applications with extensive system permissions has raised severe privacy concerns for end-users. Traditional antivirus and intrusion detection systems rely primarily on signature-based techniques, failing to identify novel threats or the misuse of legitimate permissions. This paper presents the design, implementation, and evaluation of a Privacy Intrusion Detection System (PIDS)—a lightweight, real-time security solution that monitors and controls access to sensitive resources such as webcams, microphones, file systems, and network interfaces. PIDS employs behavior-based analysis to detect anomalous application activities and provides a graphical user interface (GUI) that empowers users with actionable decisions: allow, block, or ignore. The system continuously audits running processes, correlates resource-access patterns with predefined normal-behavior profiles, and generates instant alerts for suspicious events. All activities are logged in a structured database for forensic analysis. Experimental evaluation using a prototype deployed on Windows environments demonstrates high detection accuracy for unauthorized camera/microphone access, low false positive rates, and minimal performance overhead. The PIDS GUI version bridges the gap between real-time threat detection and user awareness, offering a comprehensive privacy defense mechanism for modern computing environments.**

***Index Terms*—Privacy intrusion detection, behavior-based monitoring, endpoint security, camera/microphone protection, user-centric alerting, forensic logging.**

I. INTRODUCTION

The digital ecosystem has become deeply integrated into daily life, with users relying on an ever-increasing number of desktop and mobile applications. These applications often request broad permissions to access hardware components such as the webcam, microphone, and file system, as well as network connectivity. While many legitimate applications require such access to deliver their intended functionality, malicious or poorly designed software can exploit these permissions to spy on users, exfiltrate personal data, or perform unauthorized surveillance [1].

Conventional security mechanisms, including antivirus (AV) and host-based intrusion detection systems (HIDS), are predominantly signature-based. They compare program binaries or network traffic against databases of known malware signatures. This approach has two critical limitations: first, zero-day threats and polymorphic malware can easily evade signature detection; second, even benign applications that have been given legitimate permissions can later misuse them without triggering any signature-based alert [2]. An application that accesses the microphone silently in the background, for instance, may be fully signed and recognized as “safe” by an AV, yet still constitute a grave privacy violation from the user’s perspective.

The concept of behavioral intrusion detection has gained momentum in recent years. Instead of relying on static signatures, behavior-based systems observe the run-time actions of processes and flag deviations from established norms. In the domain of Android security, for example, systems like AppGuard [3] and TaintDroid [4] monitor information flows. On desktop platforms, however, user-focused privacy monitoring of hardware sensors remains underexplored,

particularly in a form that is both lightweight and accessible to non-expert users.

This paper introduces the Privacy Intrusion Detection System (PIDS) with a graphical user interface, a holistic security tool designed to fill this gap. PIDS continuously monitors all running processes, identifies when they access sensitive resources (camera, microphone, file system, network), and evaluates whether such access aligns with expected behaviour. It combines rule-based and anomaly-based detection techniques to flag suspicious activities. Crucially, the GUI version presents a centralized dashboard that displays real-time system status, active processes, resource-usage indicators, and instant alerts. Users are given granular control to allow, block, or ignore specific actions, thus retaining full authority over their system's privacy. All events are recorded in a structured log database suitable for subsequent forensic investigation.

The main contributions of this research are:

1. A lightweight process-monitoring framework that tracks camera, microphone, file, and network accesses in real time on Windows operating systems.
2. A hybrid detection engine that combines predefined permission-expectation rules with statistical anomaly scoring to reduce false positives.
3. An interactive GUI dashboard that bridges the gap between raw monitoring data and end-user decision-making.
4. A comprehensive experimental evaluation of detection accuracy, latency, and system resource consumption, demonstrating the practicality of PIDS for everyday use.

The remainder of this paper is organized as follows. Section 2 reviews related work in intrusion detection and privacy monitoring. Section 3 describes the system architecture and design principles. Section 4 details the implementation of the monitoring core, detection engine, and GUI. Section 5 presents the experimental setup and results. Section 6 discusses the findings and limitations, and Section 7 concludes the paper with directions for future work.

II. LITERATURE REVIEW

The field of intrusion detection has evolved substantially since the seminal work of Denning [5], who proposed the first model for real-time intrusion detection using audit records and statistical anomaly detection. Contemporary systems can be broadly categorized into network-based (NIDS) and host-based (HIDS) approaches. PIDS aligns primarily with the HIDS paradigm but focuses specifically on privacy-sensitive hardware resources.

2.1 Signature-Based vs. Behavior-Based Detection

Traditional antivirus engines and early HIDS rely on signature matching. Commercial products such as McAfee and Norton maintain large databases of malware fingerprints. While efficient against known threats, signature-based methods fail when attackers employ obfuscation, packing, or completely new infection vectors [6]. Furthermore, they offer no protection against "permission over-reach" by legitimate software.(Vishnu Venkatesh & Das, 2026)

Behaviour-based intrusion detection, on the other hand, builds a baseline of normal activity and triggers alerts upon deviations. Techniques range from simple threshold-based rules to sophisticated machine learning models. For instance, the work by Warrender et al. [7] modeled system call sequences using Hidden Markov Models to detect intrusions in Unix processes. Similarly, Forrest et al. [8] analyzed sequences of system calls to build a database of normal patterns for privileged processes. PIDS adopts a hybrid strategy: a rule-based core that captures explicit permission-expectation mappings (e.g., "Calculator should never access the microphone"), supplemented by anomaly scoring based on process-specific usage frequency and timing.

2.2 Privacy Monitoring on Mobile and Desktop Platforms

The rise of smartphones has spurred significant research on privacy monitoring. Android, being open source, has been a popular target. TaintDroid [4] provides system-wide dynamic taint tracking to follow the flow of sensitive data. AppIntent [9] analyzes GUI events to distinguish user-intended data transmissions from stealthy ones. These systems, however, require deep OS modifications and are not directly portable to desktop environments.(Venkatesh et al., 2023)

On the Windows platform, tools such as Process Monitor (Sysinternals) provide detailed real-time file system, registry, and process activity logs, but they lack automated threat classification and user-friendly

alerting. More recent commercial products like GlassWire focus on network activity visualization, while microphone/camera indicators built into Windows 10/11 only show that a resource is in use, without applying any judgement on whether the access is legitimate or malicious. PIDS fills this gap by adding a layer of semantic analysis and user control.

2.3 Endpoint Detection and Response (EDR)

Modern EDR systems, such as CrowdStrike Falcon and Microsoft Defender for Endpoint, combine signature, behaviour, and cloud-based analytics to detect advanced threats. They operate at kernel level to intercept system events and can quarantine suspicious processes. However, EDR tools are designed primarily for enterprise security operations centres (SOCs) and are often too heavy, costly, and complex for individual users. Moreover, their default policies may not prioritize privacy-specific events like camera-on events without an associated user-initiated action. PIDS targets a complementary niche: a lightweight, privacy-focused tool that runs efficiently on personal laptops and desktops, with a simple GUI that places the user in the decision loop.(Shukla et al., 2023)

2.4 User-Centric Alert Design

The effectiveness of a security system is not solely determined by its detection rate; user comprehension and response to alerts play an equally critical role. Studies in usable security [10] have shown that poorly designed alerts lead to habituation and click-through errors. PIDS addresses this by categorizing alerts into severity levels (low, medium, high) and presenting clear, plain-language descriptions: which process attempted what action, at what time, and why it was deemed suspicious. The "Allow / Block / Ignore" triadic response further empowers users without overwhelming them with technical jargon.(Shenoy et al., 2025)

III. SYSTEM DESIGN AND ARCHITECTURE

The Privacy Intrusion Detection System is designed around three core principles: real-time monitoring, semantic analysis, and user-centric notification. The architecture, depicted conceptually in Figure 1, consists of five main components: (1) a Process and Resource Monitor, (2) a Data Collection Engine, (3) a Behaviour-Based Detection Engine, (4) an Alert and Logging Subsystem, and (5) the Graphical User Interface.

Figure 1: High-Level System Architecture of PIDS *(A block diagram showing the flow: Running Processes → Monitor → Data Collector → Detection Engine → Alert/Log → GUI ↔ User Actions.)*

3.1 Process and Resource Monitor

This component runs as a background service and continuously enumerates all active processes, recording their process ID (PID), name, executable path, and parent process. It subscribes to real-time system events related to resource access: camera power state changes, microphone capture session start/stop, file operations (open, read, write, delete) in sensitive directories (e.g., Documents, Desktop), and network socket connections. The monitoring hooks are implemented using the Windows Event Tracing (ETW) framework and polling via Windows Management Instrumentation (WMI), chosen for their low overhead and reliability.

3.2 Data Collection Engine

Raw events from the monitor are filtered, timestamped, and enriched with contextual metadata. For example, a camera-on event is accompanied by the name of the requesting process, the user session under which it runs, and whether any visible window is associated with the process (an indirect measure of user intent). This structured data is fed into a memory-resident buffer and simultaneously written to the log database (SQLite) for persistence..

3.3 Behaviour-Based Detection Engine

The detection engine forms the core intelligence of PIDS. It operates in two modes:

- Rule-Based Mode: A set of pre-defined rules expressed as "If [Process_Class] accesses [Resource] without [Condition], then flag as suspicious." Process classes are derived from a curated allowlist/denylist and user-configurable profiles. For instance, "Browsers may access microphone only when an active audio-capture tab is detected." Rules are implemented as decision trees for rapid evaluation.
- Anomaly Scoring Mode: For processes not covered by explicit rules, PIDS maintains a statistical profile of resource accesses over a

sliding time window. Baseline metrics include access frequency, typical duration, and correlation with user input (keyboard/mouse activity). A suspicion score is computed using an ensemble of heuristics: an out-of-hours microphone use, absence of foreground window, or sudden spikes in camera frame reads without a corresponding video call application open. When the score exceeds a dynamic threshold, an alert is raised with an associated severity level.

The engine combines the two modes: the rule-based decision takes precedence; if a match is found, action is immediate. Otherwise, the anomaly scorer provides a second layer of defence.

3.4 Alert and Logging Subsystem

When the detection engine flags an event, an alert object is created containing:

- Timestamp
- Process name and PID
- Resource accessed
- Detection reason (rule hit or anomaly score)
- Severity (Low, Medium, High)
- Suggested action

Alerts are pushed to the GUI via a lightweight message queue. Simultaneously, a log entry is written to the database with all forensic details, including a snapshot of related processes and the detection context. Logs are tamper-protected using a rolling hash chain to preserve integrity for later forensic use.

3.5 Graphical User Interface

The GUI, built using PyQt5 (Python), provides four main panels:

1. Dashboard: Real-time status indicators (green/yellow/red) for camera, microphone, file system, and network. A live process list with resource-usage tags.
2. Alerts Panel: A table of recent alerts, filterable by severity and resource type. Each alert row includes action buttons: [Allow] [Block] [Ignore].
3. Log Viewer: A searchable, sortable view of historical events with export capabilities for CSV/PDF.
4. Settings Panel: Configuration of detection sensitivity, allow/block lists, rule customisation, and notification preferences.

The design follows human-computer interaction principles to ensure low cognitive load: colour codes, clearly labelled buttons, and non-technical language. A system tray icon indicates protection status even when the main window is closed.

IV. IMPLEMENTATION DETAILS

PIDS is developed primarily in Python 3.11 for rapid prototyping and leverages native Windows APIs via the ctypes and pywin32 libraries. The following subsections detail the technical realisation of the core modules.

4.1 Process Enumeration and Monitoring

Process enumeration is performed every 2 seconds using psutil, a cross-platform library that provides access to process details. To detect new process creation and termination in real time, a WMI event watcher
queries Win32_ProcessStartTrace and Win32_Proces sStopTrace. This event-driven approach minimizes CPU usage compared to polling alone.

4.2 Camera and Microphone Access Detection

On Windows, camera access is monitored via the Windows.Devices.Enumeration API and ETW providers related to *CameraCaptureActivity*. The system registers a background listener for the Microsoft-Windows-Camera event provider, which emits events when a camera capture session begins (StartPreview, StartRecord) and ends. The event payload includes the process ID of the requesting application. Similarly, microphone activity is tracked by monitoring the audio session manager (IAudioSessionManager2 through COM interop), which provides notifications when an audio capture stream is created or destroyed.

To ensure that no kernel-level bypass goes undetected, PIDS also periodically queries the camera's device status via DirectShow and cross-references it with the known process accessing it. A normal-state file (JSON) is maintained for authorised applications, allowing the user to pre-approve trusted apps like Zoom or Teams.

4.3 File System and Network Monitoring

File system monitoring focuses on directories commonly containing personal data: Documents,

Pictures, Desktop, and custom user-specified folders. The watchdog Python library is used to receive events for file creation, modification, and deletion, filtered by process name using psutil correlation. Network monitoring employs scapy sniffing in a low-volume background thread, capturing DNS queries and TCP connections, matched to PIDs via the Windows netstat equivalent.

4.4 Detection Engine Internals

The rule base is stored in a YAML configuration file for easy editing. A sample rule entry:

```
yaml
- rule_id: R-101
  process_pattern:
"chrome.exe|firefox.exe|msedge.exe"
  resource: microphone
  condition: "browser_tab_active_webrtc == false"
  severity: high
  action: alert_block
```

The condition is evaluated by a Python mini-interpreter that can call helper functions (e.g., checking if a browser tab with active WebRTC exists via UIAutomation). Anomaly scoring uses exponential moving averages (EMA) of access intervals. The score, *S*, for a process *p* accessing resource *r* at time *t* is calculated as:

$$S(p,r,t) = w_1 \cdot \Delta T^{-1} + w_2 \cdot U + w_3 \cdot H$$

Where ΔT^{-1} is the inverse of the time since last access (sudden re-access after long idle increases score), *U* is a binary lack-of-user-interaction flag (foreground window check), and *H* is a heuristic that checks if the access occurs during historically unusual hours. Weights w_1, w_2, w_3 are empirically set to 0.5, 0.3, and 0.2.

4.5 GUI and User Control Flow

When an alert is generated, a callback function emits a signal to the GUI's alert handler. The GUI then:

1. Displays a non-blocking toast notification near the system tray.
2. Logs the alert in the list and colours the corresponding resource indicator amber or red.
3. Awaits user action. If the user selects "Block," PIDS attempts to terminate the offending process (using TerminateProcess) or, in a more refined version, suspends the specific access handle via the Windows API. If "Allow" is selected, a temporary allow rule is added for the session. "Ignore" simply dismisses the notification but retains the log entry.

All actions are time-stamped and recorded for accountability.

V. EXPERIMENTAL EVALUATION

To assess PIDS's effectiveness, we conducted a series of controlled experiments on a testbed comprising five laptops running Windows 10/11 (64-bit) with varying hardware specifications (Intel i5/i7, 8-16 GB RAM).

5.1 Test Scenario and Benchmark

We simulated three categories of applications: benign-expected (Zoom, Chrome browser video call, Notepad accessing documents), benign-unexpected (a custom Python script that captures 5 seconds of microphone audio without a visible window, mimicking a spyware behaviour), and malicious (a known remote access trojan sample specifically sandboxed and stripped of network payload, only retaining camera-access capability). Each scenario was repeated 50 times under different user states (user present/away, foreground/background process). The ground truth was manually verified. We measured:

- True Positive Rate (TPR) – proportion of truly suspicious activities correctly flagged.
- False Positive Rate (FPR) – proportion of benign activities incorrectly flagged.
- Detection Latency – time between resource access initiation and alert presentation.
- System Overhead – CPU and memory usage of the PIDS service.

5.2 Detection Accuracy

Table 1 summarises the detection performance.

Test Scenario	TPR (%)	FPR (%)
Spy script (microphone)	100.0	–
RAT (camera)	98.0	–
Benign video call (Zoom)	–	2.0
Benign browser audio (Chrome WebRTC)	–	1.5
Benign file access (Notepad)	–	0.5

Table 1: Detection Accuracy of PIDS

The perfect detection of the spy script arises because the rule-based engine instantly flags any microphone access from a process not in the allowlist and lacking a visible window. The false positive cases for Zoom and Chrome occurred when the detection heuristic misjudged a brief audio stream re-initialisation during call setup as an anomaly due to temporary loss of foreground window focus. Adjusting the timing window reduced FPR to under 1% in subsequent tuning.

5.3 Latency and Overhead

The average detection latency was 1.2 seconds (standard deviation 0.3 s), measured from OS event emission to GUI alert display. This includes buffer processing and rule/anomaly evaluation. Such latency is acceptable for a user-interactive system.

Regarding resource overhead, PIDS's background service consumed an average of 2.3% CPU (on an Intel i5-1135G7) and 85 MB RAM. Monitoring subscriptions had negligible impact on camera/microphone latency during legitimate video calls, as confirmed by benchmark call quality measurements (no dropped frames, audio jitter within normal range).

5.4 User Study (Pilot)

A small pilot usability study was conducted with 10 participants from our university. They were asked to install PIDS, perform normal tasks (video calls, document editing), and respond to three injected suspicious events (a background microphone access). All participants successfully identified and blocked the suspicious events. The average time to understand the alert and act was 8 seconds. Qualitative feedback praised the clear language and traffic-light indicators, though two users suggested adding a permanent "learning mode" that automatically creates rules based on repetitive user allowances. This feedback is incorporated into future work.

VI. DISCUSSION

The experimental results demonstrate that PIDS effectively identifies unauthorised access to sensitive resources while maintaining a low false positive rate. Its hybrid detection strategy overcomes the limitations of purely signature-based systems. By examining process behaviour—specifically, the relationship between resource access and user-initiated interface actions—PIDS can detect stealthy surveillance attempts that would otherwise go unnoticed.

The GUI proved to be a significant enabler of user engagement. Unlike traditional HIDS that dump logs to a file or display cryptic pop-ups, the dashboard provides situational awareness at a glance. The three-way decision (Allow/Block/Ignore) respects user autonomy, transforming the user from a passive potential victim into an active decision-maker. This aligns with the principles of usable security [10], where the human is considered an integral component of the security loop rather than the weakest link.

Nevertheless, several limitations must be acknowledged. First, the current prototype operates on Windows only; Linux and macOS have different process and resource monitoring APIs. Second, evading the monitor through kernel-level rootkits could bypass user-mode ETW hooks, though PIDS could be augmented with a kernel driver in the future. Third, the anomaly scoring weights were determined empirically; application of machine learning on larger behavioural datasets could refine thresholds and reduce false positives further. Fourth, resource file-system monitoring is currently confined to predefined directories; a dynamic identification of user-important files via machine learning would enhance coverage. Finally, the user study was small and only assessed short-term usage; long-term habituation effects and alert fatigue need further investigation.

From a forensic perspective, the structured logging mechanism provides a valuable audit trail. Each alert is linked to process metadata, resource type, and user response timestamps, enabling digital forensic investigators to reconstruct privacy breach timelines and assess culpability. The tamper-resistant log chain adds evidentiary weight.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a Privacy Intrusion Detection System (PIDS) that combines real-time monitoring, behaviour-based analysis, and an interactive GUI to protect users from unauthorised access to cameras, microphones, files, and network resources. The system successfully detected simulated spyware and legitimate-but-unexpected accesses with

high accuracy, low latency, and minimal performance overhead. By giving users clear alerts and control, PIDS enhances endpoint privacy without requiring technical expertise.

Future development will focus on several directions:

1. Cross-Platform Support: Porting the monitoring core to Linux (using inotify, ALSA, V4L2) and macOS (using Endpoint Security Framework).
2. Machine Learning Integration: Replacing static rule conditions with a supervised model trained on sequences of system events, potentially using LSTMs or Transformers, to capture complex temporal behaviour patterns.
3. Automatic Rule Generation: Implementing a "learning period" during which PIDS observes user-approved accesses and automatically adds them to the allowlist, reducing initial configuration effort.
4. Network Context Enrichment: Incorporating threat intelligence feeds to evaluate whether a process communicating with a known malicious IP while accessing the microphone should raise a higher severity alert.
5. Enterprise Deployment: Developing a central management console that aggregates PIDS alerts from multiple endpoints, useful for organisational privacy oversight.
6. Deep Forensic Integration: Adding secure export of log chains in standard forensic formats (e.g., XML or JSON-LD) and ensuring compatibility with digital forensics tools like Autopsy.

By evolving toward a cross-platform, AI-driven privacy guardian, PIDS has the potential to become an essential tool in the personal security toolbox, defending against the growing threat of stealthy surveillance while respecting user autonomy and system performance.

REFERENCES

[1] P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM Conference on Computer and Communications Security, 2011, pp. 627–638.

[2] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in Proceedings of the 12th USENIX Security Symposium, 2003.

M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. von Styp-Rekowsky, "AppGuard – enforcing user requirements on Android apps," in Tools and Algorithms for the Construction and Analysis of Systems, 2013.

[3] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of OSDI, 2010.

[4] D. E. Denning, "An intrusion-detection model," IEEE Transactions on Software Engineering, vol. 13, no. 2, pp. 222-232, 1987.

[5] M. Sikorski and A. Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012.

[6] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in Proceedings of the 1999 IEEE Symposium on Security and Privacy, 1999.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996.

[8] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: analyzing sensitive data transmission in Android for privacy leakage detection," in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, 2013.

[9] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: a usability evaluation of PGP 5.0," in Proceedings of the 8th USENIX Security Symposium, 1999.

[10] Microsoft Corporation, "Event Tracing for Windows (ETW)," Microsoft Docs, 2023. [Online].
Available: https://docs.microsoft.com/en-us/windows/win32/etw/

[11] psutil documentation, "Cross-platform library for process and system monitoring." [Online]. Available: https://psutil.readthedocs.io/

[12] D. K. Nilsson and U. E. Larson, "Conducting forensic investigation of cyber attacks using digital evidence," in Digital Investigation, vol. 5, supplement, pp. S13-S21, 2008.

[13] Y. Chen and R. Sion, "On securing untrusted clouds with cryptography," in Proceedings of the

9th Annual ACM Workshop on Digital Rights Management, 2009.

[14] P. Szor, The Art of Computer Virus Research and Defense. Addison-Wesley, 2005.

[15] VENKATESH, M. N. V., & Rajiv, D. A., & Das, M. P., & Warrier, M. S. (2026). Vantage Point Recreation: A Novel Approach in Endpoint Security for Smart Homes. International Journal of Innovative Research in Technology (IJIRT). https://doi.org/doi.org/10.64643/IJIRTV12I8-191180-459

[16] Shukla, M., Srivastav, V., Khare, M. D., & Venkatesh, N. V. (2024). IoT-Driven solutions for VANET trustworthiness: Examining misconduct and position security challenges. Multidisciplinary Reviews, 6, 2023ss059. https://doi.org/10.31893/multirev.2023ss059

[17] SUNIDHI SUDHEER SHENOY, N VISHNU VENKATESH, “A Predictive Framework for Real-Time Courtroom Assistance Using AI-Based Mock Legal Advisor", IJRAR - International Journal of Research and Analytical Reviews (IJRAR), E-ISSN 2348-1269, P- ISSN 2349-5138, Volume.12, Issue 2, Page No pp.440-444, May 2025, Available at: http://www.ijrar.org/IJRAR25B2617.pdf

[18] Natarajan, Vishnu Venkatesh & Singhal, Priyank & Pandey, Digvijay & Sharma, Meenakshi & Rautdesai, Rupal & Khubalkar, Deepti & Gupta, Ankur. (2023). Crime Forecasting Using Historical Crime Location Using CNN-Based Images Classification Mechanism. 10.4018/978-1-6684-8618-4.ch013.

[19] Natarajan, Vishnu Venkatesh & Das, Priyanksha & Rajiv, Asha. (2026). A robust detect and avoid system for autonomous drone navigation. NexusTech. 1. 2026004. 10.31893/tech.2026004.