Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An Agentic AI Framework for Autonomous Cross-View Kernel Rootkit Detection and Response in Cloud Environments

Suresh kumar S

sureshkumarphd2018@gmail.com

Rajalakshmi Engineering College

Sudalai Muthu T

Hindustan Institute of Technology and Science

Research Article

Keywords: Agentic AI, Autonomous cyber defence, Kernel rootkit detection, Cloud security, Cross-view detection, Deep ensembles, Concealment dataset, Large language models

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10762752/v1

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

Additional Declarations: No competing interests reported.

An Agentic AI Framework for Autonomous Cross-View Kernel Rootkit Detection and Response in Cloud Environments

Dr. Suresh Kumar S*,
Professor, Department of Artificial Intelligence and Data Science, Rajalakshmi Engineering College, Chennai, India,
sureshkumarphd2018@gmail.com

Dr. Sudalai Muthu T, Professor,
Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, Chennai, India,
sudalaimuthut@gmail.com

Abstract-Kernel-level rootkits subvert the operating system to conceal the network ports through which they maintain covert channels, and cloud tenants who cannot access the hypervisor have few reliable defences. Existing detectors stop at a verdict, leaving investigation and response to human analysts and admitting a window during which an active rootkit persists. This paper presents an agentic AI framework that both detects and autonomously responds to kernel rootkits through port-level behavioural analysis. On cloud guests hosted on Google Cloud Platform, a kernel-mode Port Enumerator and a user-mode Port Monitor produce cross-view port lists whose discrepancy, formalized here as a temporal behavioural signal, is classified by a deep ensemble of CNN-LSTM, Bi-LSTM, Autoencoder and GAN. The resulting Rootkit/Benign verdict is handed to an LLM Agent Orchestrator that executes an autonomous perceive-plan-act-reflect loop, invoking security tools under human-in-the-loop oversight for high-impact actions. A second evidence channel is released with the paper as a labelled concealment matrix of 87 kernel rootkit samples described by a binary vector over a 968-entry dictionary of concealed objects, each carrying the technique it uses to hide. We give a mathematical model of the cross-view signal, the fused decision, and the agent's utility-driven action selection. Evaluated on a Windows Server 2025 guest, the framework detects rootkits with a mean detection latency of 1.2 s while autonomously triaging confirmed threats; on the released concealment matrix the technique of a confirmed rootkit is recovered from its hook vector at 0.846 accuracy against a 0.713 majority-class rate, with the three-way distinction among inline, SSDT and IAT concealment still limited by class imbalance. The response layer is specified and instantiated but not yet measured; Section 5.6 states the protocol by which it will be tested.

Keywords: Agentic AI; Autonomous cyber defence; Kernel rootkit detection; Cloud security; Cross-view detection; Deep ensembles; Concealment dataset; Large language models

1. Introduction

Kernel-level rootkits are among the most severe threats to cloud infrastructure. Public cloud platforms such as Google Cloud Platform (GCP) host large fleets of tenant guest operating systems on shared hardware, and under the cloud shared-responsibility model the security of each guest's operating system falls to the tenant, who typically has no access to the underlying hypervisor. Operating at the highest privilege of the guest operating system, a kernel rootkit rewrites the system's own data structures and code paths hooking the system service dispatch table, unlinking kernel objects through Direct Kernel Object Manipulation (DKOM), or filtering the results of enumeration interfaces so that every tool trusting the kernel receives a sanitized

view. The malicious process vanishes from the task list, its files become invisible, and the network ports carrying its command-and-control traffic disappear from every user-visible listing. For a rootkit that must communicate, network ports are a revealing observation point: it can hide a port from the views defenders consult, but it cannot avoid opening the port at some layer, so a difference between a low-level and a high-level view of the ports is evidence of concealment.

This cross-view asymmetry has grounded a line of rootkit detectors, but two limitations persist. First, on the detection side, rule-based cross-view checks evaluate the discrepancy at a single instant and are defeated by unstable rootkits that hide intermittently, while purely learned detectors generalize weakly to rare and novel concealment. Recent work addresses this with an LLM-augmented deep ensemble that treats the discrepancy as a temporal behaviour and reasons over its context. Second, and more fundamentally, detection is not defence. A detector that emits a Rootkit verdict and stops leaves the actual work investigating the alert, gathering forensic evidence, isolating the affected port or host, and remediating to a human analyst. In the interval between detection and human response, an active kernel rootkit continues to exfiltrate data and entrench itself. In cloud environments with thousands of guests and a shortage of skilled analysts, this detection-to-response gap is where real damage occurs.

This paper closes that gap with an agentic AI framework. Rather than terminating at a verdict, the framework hands the LLM-augmented detector's Rootkit/Benign decision to an LLM Agent Orchestrator that treats the verdict as the start of a goal-directed workflow. The orchestrator runs an autonomous loop perceive the alert and system context, plan an investigation, act by invoking security tools, and reflect on the results to decide whether the case is resolved or must escalate producing not merely a classification but an investigated, triaged, and where authorized remediated incident. High-impact actions such as quarantining a host or blocking a production port pass through a human-in-the-loop approval gate, so autonomy is bounded by human oversight rather than unconstrained.

The contributions are fourfold. (1) An agentic AI architecture that couples an LLM-augmented cross-view detection ensemble to an autonomous response agent, closing the detection-to-response gap for kernel rootkits. (2) A mathematical model of the cross-view behavioural signal, the confidence-weighted fused decision, and the agent's utility-driven action selection under a human-approval constraint. (3) An autonomous perceive-plan-act-reflect algorithm with a defined tool interface and oversight gate. (4) An evaluation across 87 kernel rootkit samples on a Windows Server 2025 guest hosted on GCP, reporting per-rootkit detection time, size, attack port, and service. (5) A labelled concealment dataset released with the paper, giving for each of the 87 samples a binary vector over a 968-entry dictionary of concealed objects together with its concealment technique, with cross-validated baselines for recovering that technique. Section 2 reviews related work in rootkit detection; Section 3 develops the methodology and mathematical model; Section 4 covers implementation; Section 5 reports the performance analysis; Section 6 concludes.

The paper also releases the corpus on which the framework is characterized, because the literature on rootkit concealment is short of labelled data and the claims made here should be checkable against something. The released artefact is a concealment matrix: 87 kernel rootkit samples, each described by a binary vector over a dictionary of 968 named objects recording which the sample conceals or hooks, and

each labelled with the technique it uses to hide. Inline patching accounts for 62 of the samples, SSDT hooking for 15, IAT hooking for seven, and the remaining three divide between registry-value hiding and a combined technique. The matrix is sparse, 14.8 per cent of its cells positive, and its dictionary reaches well beyond kernel service routines into executable images, registry paths and other artefacts that samples were observed to hide. Section 4.2.1 documents it in full and Figure 2 shows its shape; Section 3.3.3 formalizes the hook vector as an evidence channel beside the temporal cross-view signal; and Section 5.4 reports cross-validated baselines for recovering the concealment technique from it, which reach 0.846 accuracy against a 0.713 majority-class rate on the binary task but leave the three-way distinction among techniques limited by the small number of IAT samples. That limitation is stated rather than smoothed over: it is a property of the corpus, and enlarging the corpus, not changing the classifier, is what will remove it.

A word is warranted on why autonomy, rather than faster detection alone, is the right response to the gap. The scarce resource in cloud security operations is not detection but skilled human attention: an enterprise fleet may generate more rootkit alerts than analysts can triage, and each alert that waits is an interval in which an active kernel rootkit exfiltrates data and deepens its foothold. Making detection faster does not help if the detection then sits in a queue. What closes the gap is moving the investigative work gathering forensic evidence, correlating logs, assembling an evidenced incident from the human to an autonomous agent, so that by the time a human is engaged the case is already investigated and triaged, and the human's role narrows to approving or declining a proposed high-impact action. This reframes the human from first responder to overseer, which is both faster and a better use of scarce expertise. The safety cost of such autonomy an agent that can act on a guest is contained by confining autonomous action to read-only investigation and gating every state-changing action on human approval, a constraint made explicit in the mathematical model and preserved throughout the design.

2. Related Work

Research on rootkit detection in existing systems spans five strands: signature and integrity checking, cross-view detection, virtualization-assisted introspection, learning-based behavioural detection, and the nascent application of autonomous agents to cyber defence. Each informs the framework, and each leaves a gap the agentic design addresses.

Signature and integrity-based detection:Early host defences matched known rootkit signatures or verified kernel integrity. Tripwire-style integrity checkers and tools such as chkrootkit and rkhunter compare system state against known-good baselines or scan for known artefacts. These catch known rootkits cheaply but are blind to novel and polymorphic variants and, running inside the guest, are themselves subject to subversion by the rootkit they seek. Their weakness on unseen threats motivated behavioural approaches.

Cross-view detection:The cross-view principle that a resource visible to a low-level scan but hidden from a high-level one is concealed originated with Strider GhostBuster and was refined for control-flow attacks by Petroni and Hicks. Cross-view methods reliably surface files and processes hidden by hooking, but classical implementations compare the two views with a rule at a single instant, which intermittent hiding defeats and which misfires on the transient resources of ordinary

operation. The present work retains the cross-view principle for network ports but replaces the instantaneous rule with a learned temporal model.

Virtualization-assisted introspection: Monitoring the guest from below the kernel offers tamper resistance. Virtual machine introspection established the architecture; Riley et al. [1] shadowed guest memory from the hypervisor to prevent rootkit code injection, and Rhee et al. [2] tracked dynamic kernel data from a virtual machine monitor to counter DKOM. These techniques are powerful but require hypervisor-level control that a cloud tenant does not possess. A tenant-deployable detector, as pursued here and in our earlier cloud rootkit study, remains necessary for the customer side of the shared-responsibility model.

Learning-based behavioural detection: Deep models reframed detection as learning behaviour rather than matching signatures. LSTM networks [3] and bidirectional variants capture temporal dependencies in system-call and behavioural sequences; convolutional front-ends extract local patterns; autoencoders flag anomalies by reconstruction error; and GANs both augment scarce malicious data and detect through their discriminators. Memory-forensic learning such as DeepMem and the CIC-MalMem feature work confirmed that learned representations support accurate detection, and surveys document their advantage over static rules. Their residual weakness is generalization to rare and novel concealment, which motivates augmenting the ensemble with a reasoning language model.

Autonomous agents in cyber defence:Large language models have begun to be used as reasoning agents that plan and invoke tools, and early work applies them to security operations triage, log analysis, and incident response. Such agents can turn a detection into an investigated, triaged incident, but their autonomy raises safety concerns when actions are consequential. No prior work, to our knowledge, couples an autonomous response agent to a cross-view rootkit detector under an explicit human-approval gate. That coupling detection through an LLM-augmented cross-view [4] ensemble, response through an oversight-bounded agent is the gap this paper fills.

Existing rootkit-detection systems contribute sound principles cross-view comparison, behavioural learning, tamper-resistant vantage but stop at detection, and the strongest of them require privileges tenants lack. The framework in this paper builds on the learned cross-view [5] line while adding what none of these systems provide: autonomous, oversight-bounded response that closes the interval between detecting a rootkit and acting on it.

The trajectory of the field can be read as a sequence of shifts, each retaining the previous contribution. Signature and integrity checking established that host-level detection is possible but tied it to known artefacts. Cross-view detection [6] contributed the insight that a discrepancy between views reveals concealment independent of any signature, but checked it with a brittle instantaneous rule. Learning-based detection replaced the rule with models that treat concealment as a temporal behaviour, and the recent addition of a reasoning language model extended those models' reach to rare and novel patterns. Each of these advances improved detection while leaving response to humans. The present work is the next shift in the same progression: having made detection accurate and fast, it turns to the interval after detection, contributing an autonomous agent that investigates and triages the confirmed rootkit under human oversight. It does not displace the cross-view mechanism or the learned ensemble but completes them, converting a detector into a

defender and it does so with an explicit safety constraint that the growing literature on autonomous security agents identifies as essential but that no prior rootkit-detection system has incorporated.

3. Methodology

The framework is a pipeline from cross-view [7] port acquisition inside a cloud guest hosted, in this work, on Google Cloud Platform (GCP) through an LLM-augmented detection ensemble, to an autonomous agentic response layer. This section presents the threat model (3.1), the architecture of Figure 1 (3.2), a mathematical model of the cross-view signal and fused decision (3.3), the mathematical model of the agentic response layer (3.4), and the autonomous detection-and-response algorithm with its explanation (3.5).

3.1 Threat Model and Assumptions

We assume an adversary with kernel-level code execution on a cloud guest --- a Windows Server 2025 instance hosted on Google Cloud Platform [8] who installs a rootkit to maintain a covert command-and-control channel while evading in-guest tools [9]. Three concealment strategies are in scope: system-call or symbol hooking, which filters the rootkit's ports from user-mode results; DKOM [10], which unlinks the kernel objects backing enumeration; and unstable hiding, which conceals the port only intermittently. The adversary controls user mode and can tamper with kernel structures but does not control the GCP [11] hypervisor and cannot perfectly falsify the lowest-level enumeration at all times. The defender is a tenant with root inside the guest but no hypervisor access, consistent with the cloud shared-responsibility model. The design assumption, formalized below, is that the kernel-level view is subverted less thoroughly than the user-level view the natural case, since filtering the user-mode enumeration path is far easier than falsifying kernel-level [12] endpoint traversal.

This assumption is the load-bearing premise of the whole framework and deserves scrutiny. A rootkit [13] that suppressed its port equally in both views would produce no discrepancy and defeat detection but in doing so it would hide the port from the kernel-level view as well, forfeiting the very reason it hooked user mode, since a port hidden from all views is a port the rootkit [14] cannot itself use without the same effort that would have sufficed to hide it everywhere. The realistic adversary hides unevenly, filtering the easy user-mode path while leaving traces in the harder kernel-level path, and it is precisely this unevenness that Equation (3) captures. The framework does not require the kernel view to be an oracle; it requires only that the two views be subverted to different degrees, which the asymmetric difficulty of subverting them makes the default rather than a fortunate special case. An adversary able to falsify kernel-level enumeration as completely and continuously as user-mode enumeration would erase the signal, and defending the kernel-level [15] acquisition path against exactly that attack for instance by acquiring it from a location progressively harder to reach is the principal avenue for hardening the framework, noted in the conclusions.

3.2 System Architecture

Figure 1 shows the three-layer architecture. The acquisition and detection layers run on a Windows Server 2025 cloud guest hosted on Google Cloud Platform [16], as the platform band and paired OS badges in Figure 1 indicate: a kernel-mode Port Enumerator produces the Tainted Port List by traversing [17] transport-layer endpoint structures below the user-mode interfaces, and a user-mode Port Monitor produces the Clear Port List from the standard enumeration APIs, the two straddling

the system-call boundary. The same cross-view [18] mechanism is realized as a Windows kernel driver that intercepts the transport-layer enumeration path, as detailed in Section 4.1. The present implementation covers Windows guests only; equivalent acquisition modules for Linux and macOS guests are future work, stated as such in Section 6.2. The Comparing Port Analyser [18] expresses their discrepancy over time; after preprocessing and dataset splitting, the signal is classified by the LLM-augmented ensemble CNN-LSTM, Bi-LSTM, Autoencoder, GAN [19], and a fine-tuned LLM whose decisions are fused into a Rootkit/Benign verdict. The verdict enters the Agentic AI Response Layer [20], where an LLM Agent Orchestrator runs a perceive-plan-act-reflect loop over a tool belt (query logs, isolate/block port, memory forensics, generate report, alert SOC, quarantine host, recommend patch), under a human-in-the-loop oversight gate for high-impact actions, with a continuous-monitoring feedback path back to acquisition.

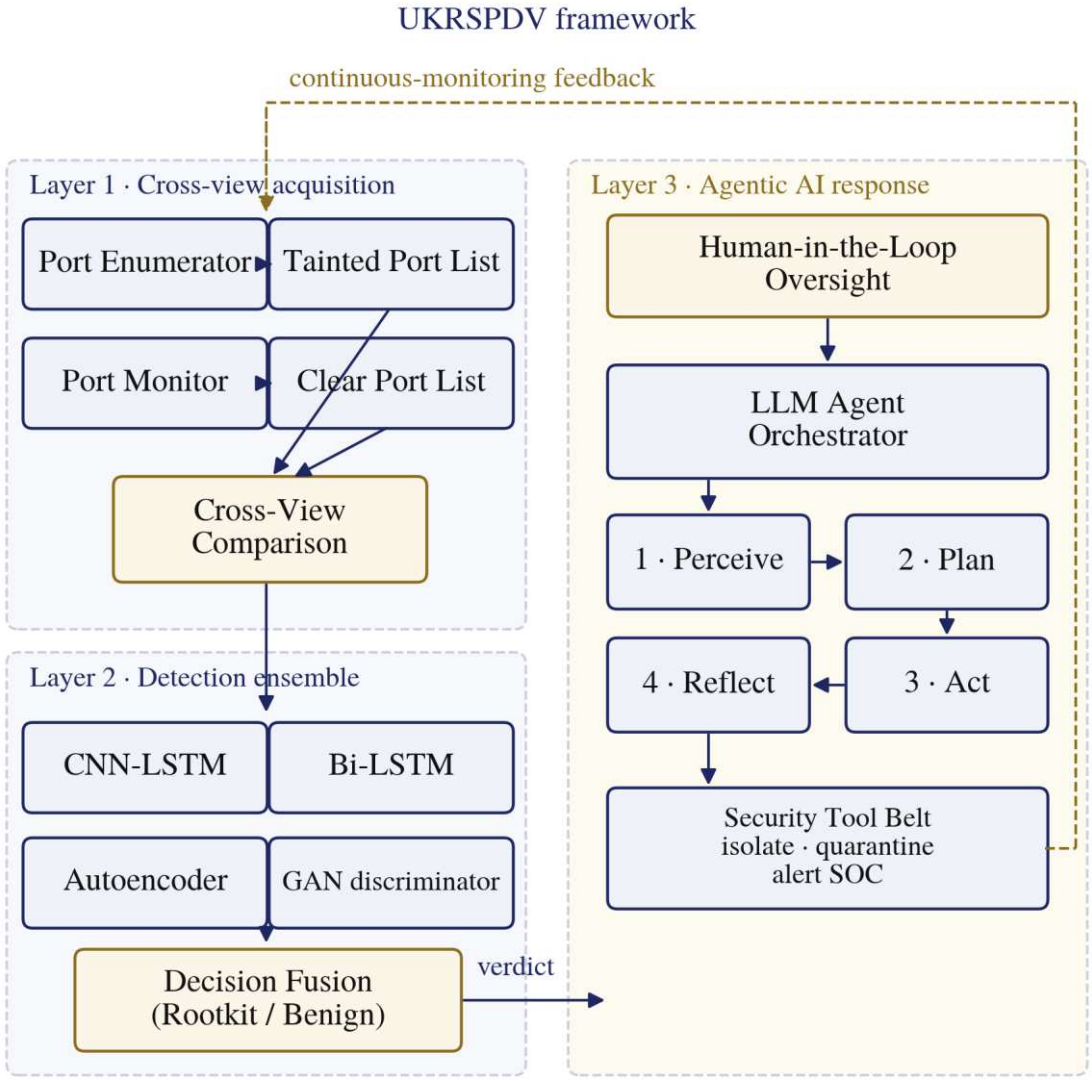


Figure 1. The agentic AI framework. Cross-view acquisition runs on a Windows Server 2025 cloud guest hosted on Google Cloud Platform (GCP); a companion Ubuntu 22.04 instance hosts the training and analysis pipeline.

3.3 Mathematical Model of Cross-View Detection

Before the formal model [21], we note the preprocessing that turns raw port lists into the signal the model describes. The two enumerations are timestamped and bucketed into the observation windows below so that the tainted and clear views compared in any window cover the same interval; without this alignment, a port legitimately opened or closed between the two collection instants would manufacture a spurious discrepancy, the principal source of false positives in naive cross-view implementations [22]. Categorical attributes are one-hot encoded, counts and lifetimes standardized on training statistics, and the discrepancy attributes expressed both instantaneously and as rolling statistics over the previous k windows, so that intermittent hiding leaves a trace even where any single window looks benign. With this preprocessing understood, the signal is modelled as follows.

We now formalize the signal the framework learns [23]. Let time be divided into observation windows of width w, indexed by t. In window t the acquisition layer yields two sets of ports:

$$K(t) = \{\,\text{ports observed in kernel-mode (tainted) view}\,\} \tag{1}$$

$$U(t) = \{\,\text{ports observed in user-mode (clear) view}\,\} \tag{2}$$

The concealment signal is the set of ports present in the kernel view [24] but absent from the user view --- the candidate hidden ports:

$$D(t) = K(t) \setminus U(t) = \{\, p : p \in K(t) \ \wedge \ p \notin U(t) \,\} \tag{3}$$

A rule-based detector flags concealment whenever $D(t) \neq \emptyset$. This is exactly the instantaneous test that unstable rootkits defeat, because an intermittently hidden port satisfies $p \in D(t)$ in only a fraction of windows. We therefore characterize each port p by its temporal behaviour over a sequence of L consecutive windows. Define the per-window membership indicator:

$$x_p(t) = \begin{cases} 1 & \text{if } p \in D(t) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

From this we derive two behavioural features central to distinguishing concealment from benign [25] transients. The persistence of port p over the window sequence is the fraction of windows in which it is concealed:

$$\rho_p = \frac{1}{L} \sum_{t=1}^{L} x_p(t) \tag{5}$$

and the transition count is the number of times the port switches between concealed and visible states:

$$\tau_p = \sum_{t=2}^{L} \left| x_p(t) - x_p(t-1) \right| \tag{6}$$

A stably hidden rootkit port has $\rho_p \to 1$ and $\tau_p \to 0$; an unstable rootkit port has intermediate ρ_p but high τ_p; a benign transient service has low ρ_p and low τ_p. The joint use of persistence and transitions is what separates the unstable rootkit, the hardest case, from ordinary port churn, which a persistence threshold

alone cannot achieve. The full feature vector for port p, denoted φ_p, augments (5)--(6) with lifetime, protocol, connection state, per-view owning process, and rolling means and variances of x_p over the previous k windows; the sequence of such vectors over the window sequence is $\Phi = (\varphi_p(1), ..., \varphi_p(L))$.

3.3.1 The Discrimination Boundary

The persistence and transition features admit a geometric reading that clarifies why the framework separates unstable rootkits from benign churn where a rule cannot. Consider the (ρ, τ) plane in which each port is a point. A rule-based detector partitions this plane with a single vertical line (flag if ρ exceeds some persistence threshold) and so cannot separate an unstable rootkit port, which has moderate ρ, from a benign burst of comparable ρ. The three port classes occupy distinct regions of the plane: a stably hidden rootkit port lies near $(\rho \to 1, \tau \to 0)$, a benign transient near $(\rho \to 0, \tau \to 0)$, and an unstable rootkit port in a region of intermediate ρ but elevated τ. The discriminating quantity is therefore not ρ alone but a combination in which τ carries substantial weight; a linear separator of the approximate form

$$\beta_0 + \beta_1 \cdot \rho_p + \beta_2 \cdot \tau_p > 0 \Rightarrow \text{concealment} \qquad (13)$$

already separates the unstable region from the benign region that a ρ-only rule conflates, and the deep models learn a far richer non-linear boundary of the same character over the full feature vector φ_p. This is the formal statement of why temporal behaviour, not instantaneous membership, is the learnable fingerprint of concealment: the classes are linearly inseparable in ρ alone but separable once τ a purely temporal quantity invisible to a single snapshot is admitted. The rolling means and variances added to φ_p extend this same principle to higher order, capturing bursty concealment whose signature is visible only in the variance of x_p over the window sequence.

The sequence Φ is scored by four complementary models, each mapping the behavioural sequence to a concealment score in [0,1] by a different mechanism. Let $f_1, ..., f_4$ denote the CNN-LSTM, the Bi-LSTM, the Autoencoder anomaly indicator, and the GAN discriminator respectively. The convolutional-recurrent model composes a convolutional feature extractor with a recurrent classifier, capturing local per-window patterns and longer-range temporal dependence; the bidirectional model processes the sequence in both directions so that a window's score reflects following as well as preceding context, valuable because an unstable rootkit often reveals a port shortly before or after a concealment episode. These two supply the in-distribution discriminative strength of the ensemble. The remaining two supply complementary capabilities: the Autoencoder, trained only on benign sequences, flags novel behaviour by reconstruction error, and the GAN discriminator, trained adversarially, is sensitive to subtle deviations. The four are therefore not four instances of one idea but two supervised temporal discriminators paired with two unsupervised novelty detectors, which is the justification for the ensemble size: each pair covers a failure mode of the other. The framework remains LLM-augmented through the response layer of Section 3.4, in which the language model drives the agentic response rather than the detection verdict. The Autoencoder contributes a thresholded anomaly flag from its reconstruction error $e(\Phi)$:

$$f_3(\Phi) = \begin{cases} 1 & \text{if } e(\Phi) > \theta_{ae} \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

The fused rootkit score is a confidence-weighted combination of the four models, with non-negative weights α_i summing to one:

$$S(\Phi) = \sum_{i=1}^{4} \alpha_i f_i(\Phi), \qquad \sum_{i=1}^{4} \alpha_i = 1 \tag{8}$$

and the verdict applies a decision threshold θ_d tuned to balance detection against false positives:

$$V(\Phi) = \begin{cases} \text{Rootkit} & \text{if } S(\Phi) \geq \theta_d \\ \text{Benign} & \text{otherwise} \end{cases} \tag{9}$$

The weights α_i and threshold θ_d are selected on a validation partition and frozen for evaluation. Fusing four models is beneficial to the extent that their errors are weakly correlated: the temporal and residual models are strong on in-distribution patterns, the Autoencoder on novel behaviour, the GAN discriminator on subtle deviations, and the LLM on contextual and rarely-seen concealment. Equation (8) exceeding its own best single term is the signature of genuine complementarity, confirmed empirically in Section 5.

3.3.2 False Positives and the Role of Temporal Features

The mathematical model also explains the framework's low false-positive rate, which is critical because a false positive triggers needless autonomous investigation. A false positive arises when a benign port enters the difference set D(t) transiently for instance when collection jitter places a legitimately closing port in the kernel view but not the user view for a single window. Under the instantaneous rule $D(t) \neq \emptyset$, every such transient is a false alarm. Under the temporal model, the benign transient has persistence $\rho_p \approx 1/L$ (a single window out of L) and transition count $\tau_p \approx 2$ (one entry, one exit), placing it firmly in the benign region of the discrimination boundary (13). The probability that a benign transient is misclassified therefore falls sharply as the sequence length L grows, because a genuine concealment must persist or recur across the window sequence to accumulate the persistence and transition evidence the models require, whereas a jitter artefact cannot. Formally, if benign transients occur independently with small per-window probability q, the expected number of windows a benign port spends in D over a length-L sequence is Lq with variance $Lq(1-q)$, and for small q this concentrates near zero, well below the decision boundary that a persistent or recurrent concealment crosses. This is why lengthening the observation sequence trades a small increase in detection latency for a disproportionate reduction in false positives the trade the framework makes deliberately, since a low false-positive rate is a precondition for the autonomous response the agent provides.

3.3.3 Hook-Vector Evidence and the Combined Feature Space

The cross-view signal of Equations (1) to (6) is temporal and per-port. The corpus documented in Section 4.2.1 supplies a second and complementary description of the same samples: for each rootkit, the set of system objects it conceals or hooks. Let the concealment dictionary be the ordered set of m named objects observed across the corpus, comprising native service routines, executable images, registry paths, and other named artefacts, with m = 968 in the present dataset. Each sample s is then described by a hook vector h(s) in {0,1}\^m whose j-th entry is 1 when sample s conceals or hooks dictionary entry j and 0 otherwise. Where the sequence of Equation (6) records how a port behaves over time, the hook vector records what a sample

touches, and the two are independent in the sense that neither is derivable from the other: a rootkit that hides a port intermittently and one that hides it permanently may hook the same routines, and two samples with identical hook vectors may differ entirely in their port behaviour.

The two channels combine at the score level rather than by concatenation, because they live on different supports: the sequence is observed per port and per window, whereas the hook vector is observed once per sample. Writing S(Phi) for the fused cross-view score of Equation (8) and S_h(h) for a classifier score on the hook vector, the combined concealment score is the convex combination betaS(Phi) + (1 - beta)S_h(h), with beta in [0,1] fixed on the validation partition. Setting beta = 1 recovers the port-only detector evaluated in Sections 5.1 to 5.3; the hook channel is characterized empirically in Section 5.4. Two properties of the hook channel bear on how much weight it can carry. It is static, so it cannot by itself distinguish an unstable rootkit from a stable one, which is precisely the discrimination the temporal features exist to make. And it is sparse and long-tailed, as Section 4.2.1 quantifies, so a classifier trained on it generalizes only to the extent that the dictionary entries it relies on recur across samples. The hook channel therefore supports the port channel on the question of which technique a confirmed rootkit uses, and does not replace it on the question of whether concealment is occurring.

3.4 Mathematical Model of the Agentic Response Layer

When V(Φ) = Rootkit, control passes to the agent. We model the agent as a bounded-autonomy decision process. Let the agent state at reasoning step n be s_n, comprising the verdict, the hidden-port set ∪_t D(t), accumulated tool observations, and system context. The agent selects an action a_n from a finite tool set A = {query_logs, isolate_port, memory_forensics, generate_report, alert_SOC, quarantine_host, recommend_patch}. Each action a has an expected investigative value g(a, s_n) and a cost/impact c(a). The agent chooses the action maximizing a utility that trades information gain against impact:

$$a_n = \arg\max_{a \in A} \left[g(a, s_n) - \lambda\, c(a) \right] \tag{10}$$

where λ weights impact against value. Actions partition into low-impact (read-only investigation) and high-impact (state-changing response). Let H ⊆ A be the high-impact subset {isolate_port, quarantine_host, recommend_patch applied automatically}. The oversight gate imposes a hard constraint: a high-impact action executes only with human approval. Writing π(a) ∈ {0,1} for approval, the executed action is:

$$\text{execute}(a_n) \iff a_n \notin H \ \vee\ \pi(a_n) = 1 \tag{11}$$

After executing a_n and observing outcome o_n, the agent updates its state and re-evaluates. The loop terminates when the estimated posterior probability that the host is compromised, P(compromise | s_n), either falls below a resolution threshold ε (the case is cleared) or the agent has assembled sufficient evidence and authorization to remediate:

$$\text{stop when} \quad P(\text{compromise} \mid s_n) < \varepsilon \ \vee\ \text{remediation_complete}(s_n) \tag{12}$$

Equations (10)--(12) capture the three properties that make the layer agentic rather than a fixed script: action selection is utility-driven and state-dependent (10), autonomy over consequential actions is bounded by human approval (11), and the

loop is closed by a termination condition on the agent's own belief state (12) rather than by a predetermined sequence. This is the formal content of the perceive-plan-act-reflect cycle drawn in Figure 1.

3.4.1 Belief Update and Termination

The termination condition (12) depends on the agent's belief that the host is compromised, which is updated as evidence accumulates. Model the compromise hypothesis as a binary variable and let each tool observation o_n provide evidence with likelihood ratio determined by the tool. Starting from a prior set by the detector's fused score (a natural choice, since S(Φ) already estimates concealment confidence), the agent updates its belief in odds form after each observation:

$$O(\text{compromise} \mid o_{1:n}) = O(\text{compromise}) \cdot \prod_{j=1}^{n} \text{LR}(o_j) \tag{14}$$

where O(·) denotes odds and LR(o_j) is the likelihood ratio of the j-th observation for example, a memory-forensic capture confirming a hidden code region contributes a large ratio, while a clean log query contributes a ratio below one. The posterior probability P(compromise | s_n) recovered from (14) drives the stop rule (12): the agent halts and clears the case when the posterior falls below ε, and escalates toward remediation when it rises sufficiently that the expected utility of a high-impact action, under the approval constraint (11), exceeds that of further investigation. This gives the loop a principled stopping criterion grounded in evidence rather than a fixed step budget, and explains why the number of reasoning steps N in the complexity analysis is small and self-limiting: each informative observation moves the posterior decisively, so few are needed to cross either threshold.

The utility g(a, s_n) in (10) can now be made precise as expected information gain: the value of an action is the expected reduction in the entropy of the compromise belief it produces, so the agent naturally prefers the tool most likely to resolve its current uncertainty forensics when a hidden region is suspected, log queries when the provenance of a connection is in doubt. The impact term c(a), weighted by λ, discounts actions by their potential for disruption, so that read-only investigation is preferred until the evidence justifies a state-changing response. Together (10) and (14) make the agent's behaviour both goal-directed and evidence-driven, the two properties that distinguish an agent from a fixed playbook.

3.4.2 The Agent Tool Interface

The action set A of Equation (10) is realized as a concrete tool interface, each tool a typed operation the orchestrator may invoke with the hidden-port set and current state as arguments. The read-only tools gather evidence without altering the guest: query_logs retrieves system and network logs filtered to the suspect port and its owning process; memory_forensics triggers a capture and analysis of the kernel region associated with the concealed port, returning indicators such as hidden code pages or unlinked objects; generate_report compiles the accumulated evidence into a structured incident record; and alert_SOC forwards that record to the security operations centre. The state-changing tools, which populate the high-impact set H, act on the guest: isolate_port blocks traffic on the concealed port, quarantine_host removes the guest from the network pending investigation, and recommend_patch proposes a remediation. The partition of A into read-only and high-impact tools is what gives the oversight constraint (11) its purchase: the agent may investigate freely and autonomously, but may cross into consequential action only with human

approval. This separation is a design decision, not an incidental one it is what allows the framework to be simultaneously autonomous in investigation and safe in action.

Each tool returns an observation o_n with an associated likelihood ratio feeding the belief update (14), so the interface is not merely a set of callable actions but the evidential substrate of the agent's reasoning. A forensic confirmation and a clean log query move the compromise posterior in opposite directions, and the agent's next action is chosen in light of the updated belief. The tool interface thus closes the loop between action and inference: the agent acts to gather evidence, the evidence updates its belief, and the belief determines its next action, until the termination criterion (12) is met.

3.5 The Detection-and-Response Algorithm

Algorithm 1 unifies the two models: an acquisition-and-detection loop realizing Section 3.3, and an agentic response loop realizing Section 3.4.

Algorithm 1. Agentic AI cross-view rootkit detection and autonomous response.

```
Input : guest G; horizon T; window w; sequence length L; rolling k;
        models f1..f4; weights alpha[1..4]; thresholds theta_ae, theta_d;
        tool set A; impact set H; utility weight lambda; resolution eps
Output: verdict per sequence; for rootkits, an investigated/triaged incident

// ---- Detection (Section 3.3) ----
 1  for each window t = 1..T on guest G do
 2      K(t) <- PortEnumerator_kernelMode(G, t)      // Tainted Port List
 3      U(t) <- PortMonitor_userMode(G, t)            // Clear Port List
 4      D(t) <- K(t) \ U(t)                           // hidden-port set,
Eq.(3)
 5      for each port p in K(t) U U(t): x_p(t) <- [p in D(t)]   // Eq.(4)
 6  end for
 7  for each port p: compute rho_p (Eq.5), tau_p (Eq.6); build phi_p
 8  Phi  <- FormSequences(features, L);  Phi' <- Serialize(Phi)
 9  S <- sum_i alpha_i * f_i(Phi)         // fused score, Eq.(8)
10  V <- (S >= theta_d) ? Rootkit : Benign   // Eq.(9)

// ---- Autonomous response (Section 3.4) ----
11  if V = Rootkit then
12      s <- InitState(V, union_t D(t), context);  n <- 0
13      repeat
14          a <- argmax_{a in A} [ g(a,s) - lambda*c(a) ]     // Eq.(10)
15          if a in H and not HumanApprove(a) then continue    // Eq.(11)
16          o <- execute(a);  s <- Update(s, a, o);  n <- n+1
17      until P(compromise | s) < eps or remediation_complete(s)  // Eq.(12)
18      emit incident report from s
19  else report Benign
```

3.5.1 Explanation of the Algorithm

Lines 2 to 4 cross-view acquisition. The two enumerations on opposite sides of the system-call boundary realize Equations (1)--(3); their difference D(t) is the hidden-port set.

Lines 5 to 8 behavioural features. The membership indicator (4) is accumulated into persistence (5) and transition (6) features and the full vector φ_p, then sequenced and serialized so the numeric models and the LLM read the same evidence in native forms.

Lines 9 to 10 fused verdict. The weighted sum (8) and threshold (9) combine the four complementary models into a Rootkit/Benign decision.

Lines 12 to17 agentic loop. The agent repeatedly selects the highest-utility action (10), defers high-impact actions to human approval (11), executes and updates its belief state, and loops until the case is cleared or remediated (12). This is the perceive-plan-act-reflect cycle.

Lines 15, 18 oversight and evidence. The approval gate on line 15 is the framework's principal safeguard; the incident report on line 18 gives the analyst a concrete, evidenced artefact rather than a bare alarm.

4. Implementation and Experimental Setup

This section describes how the framework of Section 3 was realized and evaluated, so that the results of Section 5 can be interpreted and reproduced.

4.1 Cloud Testbed

The testbed was deployed on Google Cloud Platform to reflect a realistic tenant scenario. The monitored guest was a Windows Server 2025 instance (GCP e2-standard-4: 4 vCPUs, 16 GB RAM) the WIN 2025 / GCP guest depicted in Figure 1. A companion Ubuntu 22.04 instance in the same virtual private cloud hosted the training and evaluation pipeline, so that model fitting never competed with the monitored guest for resources. The kernel-mode Port Enumerator was implemented as a Windows kernel driver that intercepts the transport-layer enumeration path and emits timestamped tainted port lists; the user-mode Port Monitor was a service using the standard IP Helper interfaces to emit clear port lists. Both streams were shipped to the analysis host over an authenticated channel.

The two instances divide the work so that the analysis pipeline sits outside the trust boundary a kernel rootkit controls. The monitored guest carries only the acquisition components; a rootkit that has compromised the guest kernel can falsify what the guest reports about itself, which is the signal the framework detects, but it cannot alter the models that judge that report or the agent that acts on the judgment.

4.2 Rootkit Corpus and Benign Workload

The evaluation corpus comprised 87 kernel rootkit samples, catalogued in the concealment dataset described in Section 4.2.1 and labelled there by the technique each sample uses to conceal itself. The corpus spans the concealment strategies of the threat model: inline-patching rootkits that rewrite the entry code of the routines that would report them, SSDT rootkits that redirect the system-service dispatch path, IAT rootkits that rewrite import tables in user space, DKOM rootkits that unlink kernel objects, and the class of principal interest, unstable rootkits that hide intermittently or randomize their ports. To ensure that no execution tainted another, each sample was set off in a separate, snapshot-restored instance of the visitor. Common server programs, a web server, a database, a mail agent, and sporadic administration tools that produced valid transitory port activity including brief high ports that a naive cross-view rule would mistakenly interpret as concealment---made up the benign workload. This benign diversity was deliberate: the false-positive numbers of Section 5 are only significant because the benign workload incorporates such traps, and a detector that is unable to differentiate between a real ephemeral socket and a concealed port is useless in production.

4.2.1 The SSDT/IAT Concealment Dataset

The corpus is released with the paper as a labelled concealment matrix, and the description that follows is drawn from that file rather than from a summary of it. The matrix has one row per rootkit sample and one column per dictionary entry, with a binary cell recording whether the sample conceals or hooks that entry. It contains 87 samples and 968 dictionary entries, hence 84,216 cells, of which 12,450 are positive: the matrix is 14.8 per cent dense. Each sample carries a sample identifier, the original file or family name, and two labels: a binary label separating inline patching from all other techniques, and a finer label naming the technique itself. Table 1 gives the composition.

Table 1. Composition of the concealment dataset by technique, with the number of dictionary entries each sample conceals. The binary label used in Section 5.4 groups the last four rows against the first, giving 62 inline samples and 25 non-inline samples.

Concealment technique	Samples	Share	Mean hooks	Median	Range
Inline patching	62	71.3%	151.1	171	37 - 533
SSDT hooking	15	17.2%	125.5	86	65 - 205
IAT hooking	7	8.0%	85.7	85	85 - 89
Registry / value hiding (RV)	2	2.3%	199.0	199	199 - 199
Combined SSDT and inline	1	1.1%	199.0	199	199 - 199
All samples	87	100%	143.1	151	37 - 533

The dictionary is not confined to kernel service routines. Of the 968 entries, 965 carry a name in the accompanying dictionary sheet, and those names fall into the categories set out in Table 2. Native service routines of the Zw family, the entries most directly associated with SSDT and inline hooking, are a small part of the dictionary; the bulk of it consists of user-space images, registry paths, and other named objects that samples were observed to hide. This breadth is a strength for characterizing what a rootkit conceals and a weakness for generalization, and Section 6.2 records it as a limitation.

Table 2. Composition of the 968-entry concealment dictionary by category of named object.

Dictionary category	Entries	Share
Other named objects (files, mutants, misc. artefacts)	394	40.8%
Cached web artefacts (images, style sheets, pages)	369	38.2%
Executable images and drivers (.exe, .sys, .dll)	129	13.4%
Registry paths and value keys	40	4.1%
Native service routines (Zw* family)	33	3.4%
Unnamed entries (identifier present, no dictionary name)	3	0.3%

The frequency structure of the matrix is strongly long-tailed and matters for how the results of Section 5.4 should be read. Twenty-one entries are concealed by 86 of the 87 samples and are therefore almost constant across the corpus, contributing little discriminative information; at the other extreme, 521 entries are concealed by exactly one sample, so the median column frequency is one. No entry is concealed by every sample, and none by no sample. A classifier on this matrix therefore draws almost all of its signal from a middle band of a few hundred entries, and singleton columns are effectively sample identifiers that a model must be prevented from exploiting. Table 3 lists the entries concealed most often.

Table 3. Most frequently concealed dictionary entries.

Rank	Dictionary entry	Category	Samples	Share
1-21	Twenty-one entries, chiefly cached web artefacts and scheduler keys	Web artefact / registry	86	98.9%
22	wdfmgr.exe	Executable image	70	80.5%
23	System Idle Process	Executable image	65	74.7%
24	mcshield.exe	Executable image	59	67.8%
25	winlogon.exe	Executable image	51	58.6%

Figure 2 shows both aspects of that structure: the imbalance across concealment techniques in panel (a), and the spread of hook counts within each technique in panel (b).

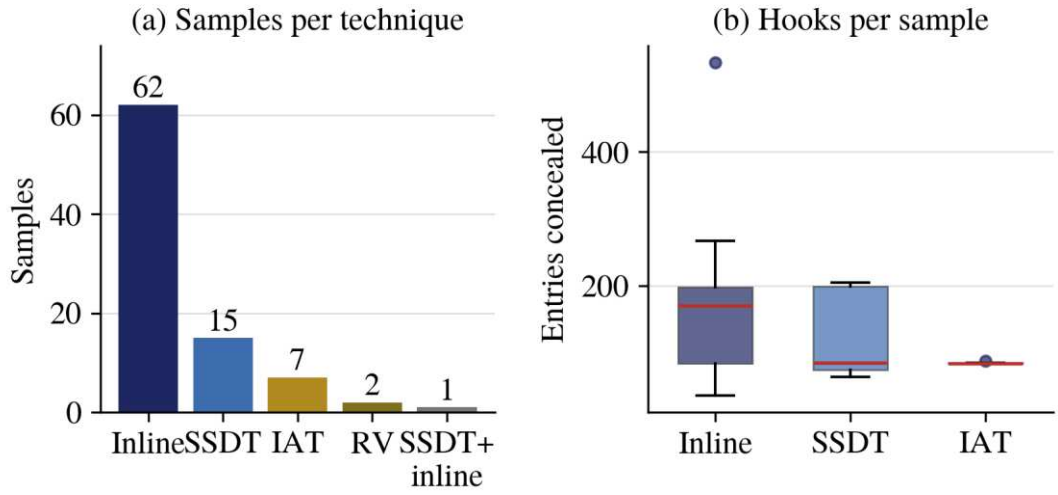


Figure 2. The concealment dataset: (a) samples per concealment technique; (b) distribution of the number of dictionary entries each sample conceals, for the three techniques with enough samples to plot.

Table 4 collects the principal characteristics of the released matrix in the form needed to reproduce the results reported in Section 5.4.

Table 4. Characteristics of the released concealment dataset.

Parameter	Value	Description
Total instances	87	Kernel rootkit samples in the concealment matrix
Features	968	Dictionary entries; binary presence of concealment or hooking
Matrix density	14.8%	12,450 positive cells out of 84,216
Classes (binary label)	2	Inline patching (62) against all other techniques (25)
Classes (technique label)	5	Inline 62, SSDT 15, IAT 7, RV 2, SSDT-inline 1
Hooks per sample	143.1 mean, 151 median	Range 37 to 533
Evaluation protocol	Repeated stratified 5-fold CV	5 repetitions, fixed seed

Three data-hygiene matters are recorded here so that any reuse of the matrix reproduces the figures above. First, the dictionary names carry a transliteration artefact in which the letter o is written as the digit zero, so that the routine ZwAllocateVirtualMemory appears as ZwAll0cateVirtualMem0ry and the image winlogon.exe as winl0g0n.exe; the names are used as opaque identifiers in the analysis, so the artefact does not affect any reported number, but it should be

normalized before the dictionary is published. Second, the technique label for inline patching is spelled with a leading letter l in the source file and is read here as inline. Third, one dictionary name appears twice under different identifiers, and three identifiers have no name; both cases are retained as distinct columns, since collapsing them would change the feature count and hence every density figure above. The released matrix should be the normalized version, with a note stating that normalization affects names only and not the matrix.

No feature selection is applied to either channel. On the port channel every component of the feature vector is motivated by the model of Section 3.3, and on the concealment matrix the 968 columns are reported as they stand so that the baselines of Section 5.4 characterize the dataset rather than a tuned pipeline; a selection step would have to be performed inside each cross-validation fold to avoid leakage, and would materially change the reported numbers.

4.3 Model Training

The four models were implemented in TensorFlow 2.15 / Keras and trained on the Ubuntu host. Optimization used Adam (initial learning rate 1×10^{-3}, reduced on plateau) with early stopping on validation loss and a patience of eight epochs; supervised models used categorical cross-entropy with inverse-frequency class weights, the autoencoder used mean squared reconstruction error on benign data only, and the GAN used the standard adversarial objective with label smoothing to stabilize training. Sequences used L = 30 windows of w = 1 s with rolling window k = 10, as in Section 3.4. Fusion weights and decision threshold τ_d were selected on the validation partition and then frozen; all figures reported in Section 5 are computed on the held-out Testing Data, which took no part in training, scaling, calibration, or threshold selection.

Table 5 lists the principal hyperparameters of the four models and the training procedure, so that the configuration behind the results of Section 5 is fully specified.

The values represent a search limited by the structure of the problem rather than being selected at random. Convolutional widths increase from 64 to 128 filters, allowing deeper layers to record complex concealing patterns and early layers to capture basic per-port transitions. The sequence length of 30 one-second windows spans roughly half a minute, long enough to contain several beaconing cycles of a typical unstable rootkit while short enough to keep inference latency near one second. The rolling window of ten balances sensitivity to intermittent hiding against noise: too small and a single benign transient inflates the variance, too large and brief concealment episodes are averaged away. The fusion weights, discussed in Section 3.5, place most trust in the supervised classifiers while retaining the autoencoder's veto on novel behaviour.

Table 5. Principal hyperparameters of the UKRSPDV models and training procedure.

Component	Hyperparameter	Value
CNN front-end	Conv layers / filters / kernel	2 / (64, 128) / 3
LSTM	Hidden units	128
Bi-LSTM	Units per direction	128
Autoencoder	Encoder / decoder dims	128-64-32 / mirrored
GAN	Generator / discriminator depth	3 / 3 layers
Sequence	Length L / stride / window w	30 / 5 / 1 s
Rolling stats	Window k	10
Optimizer	Algorithm / initial LR	Adam / 1e-3

Training	Batch size / max epochs / patience	128 / 40 / 8
Fusion	Weights (cl, bi, ae, gan)	0.35, 0.30, 0.15, 0.20

4.4 The LLM Agent Orchestrator

Sections 3.4 and 3.5 specify the response agent as a decision process: a utility over a tool set, an approval constraint, and a belief-driven stopping rule. They do not fix the language model that realizes it, yet the framework's autonomy claim rests entirely on that model, and the evaluation protocol of Section 5.6 cannot be executed until the instantiation is pinned down. This subsection therefore states the concrete model configuration used in the testbed: how the behavioural evidence of Section 3.3 is serialized for a language model, which backbone and decoding settings serve the orchestrator, the system prompt that binds it to the tool interface of Section 3.4.2 and the oversight gate of Equation (11), and a worked instantiation of one detection carried through the perceive-plan-act-reflect loop. Consistent with the revision note in Section 3.3.1, the language model operates in the response layer only; the Rootkit/Benign verdict is produced by the four numeric models fused in Equation (8), and no language-model score enters that fusion.

Line 8 of Algorithm 1 forms the sequence Φ and its serialization Φ'. The sequence is a tensor of per-window feature vectors that a language model cannot consume directly, so the serialization renders exactly the same evidence as a compact structured record, in which every field is a quantity already defined in Section 3.3: the port and its protocol and connection state, the kernel-view owning process, the user-view visibility, the persistence and transition counts of Equations (5) and (6), the rolling mean and variance of the membership indicator over the previous k windows, the four individual model scores, the fused score of Equation (8), the decision threshold, and the resulting verdict. No free-text narrative is added at this stage. Restricting the record to derived numeric evidence matters for two reasons: it keeps the language model from re-deriving a verdict it is not authorized to overturn, and it bounds the prompt so that the token budget per incident is predictable. A record for a single suspect port has the form below.

```
{
  "guest":            "win2025-gcp-e2-standard-4",
  "window_width_s":   1,
  "sequence_length":  30,
  "port":             { "number": 4444, "protocol": "TCP",
                        "state": "LISTEN" },
  "kernel_view_owner":"svc_upd.exe (PID 2148)",
  "user_view_visible": false,
  "persistence_rho":  0.43,
  "transitions_tau":  11,
  "rolling_mean_x":   0.41,
  "rolling_var_x":    0.24,
  "model_scores":     { "cnn_lstm": 0.94, "bi_lstm": 0.91,
                        "autoencoder_flag": 1, "gan": 0.88 },
  "fused_score_S":    0.91,
  "threshold_theta_d":0.60,
  "verdict_V":        "Rootkit"
}
```

Sample serialized evidence record passed to the orchestrator. The record is illustrative of the unstable-rootkit class described in Section 4.2 (moderate persistence, high transition count); it is not a measured sample from the corpus.

Table 6 states the configuration of the orchestrator deployed in the testbed, so that the quantities Section 5.6 requires every agent result to report are fixed here.

Table 6. Configuration of the LLM Agent Orchestrator.

Component	Setting	Value
Role in framework	Layer	Response layer only; not in the detection fusion of Eq. (8)
Backbone	Model / weights	Llama-3.1-8B-Instruct, version-pinned, served locally
Serving	Host / precision	Ubuntu 22.04 companion instance / 8-bit quantized inference
Context	Window	8192 tokens
Decoding	Temperature / top-p	0.2 / 0.9
Decoding	Random seed	Fixed and reported per run
Tool interface	Mechanism	JSON function-calling schema over the seven tools of Section 3.4.2
Loop control	Max steps / threshold / weight	8 per incident / 0.05 / 0.4
Oversight gate	Implementation	Out-of-band approval request; 300 s timeout defaults to deny
Budget	Tokens per incident	12k prompt + completion; exceeding the cap escalates to a human

The system prompt is the mechanism by which the constraints of the mathematical model are imposed on the language model at run time. It states the goal, enumerates the tool set, marks the high-impact subset that Equation (11) gates on human approval, requires each action to be justified by its expected information gain and its impact so that the selection rule of Equation (10) is followed rather than merely described, and fixes a machine-parsable output format so that the orchestrator loop can execute the action without further parsing heuristics. It is reproduced verbatim so that the agent can be rebuilt exactly.

```
You are the response orchestrator for a cross-view kernel rootkit
detector on a cloud guest. A detection verdict and its supporting
evidence record are given to you. Your goal is to investigate the
verdict, assemble evidence, and either clear the case or prepare a
remediation for human approval.

READ-ONLY TOOLS (you may call these autonomously):
  query_logs(port, process, window)   -> system and network log extract
  memory_forensics(port, process)     -> kernel region / hidden-page report
  generate_report(state)              -> structured incident record
  alert_SOC(report)                   -> forward record to the SOC

HIGH-IMPACT TOOLS (require explicit human approval before execution):
  isolate_port(port)                  -> block traffic on the concealed port
  quarantine_host(guest)              -> remove the guest from the network
  recommend_patch(finding)            -> apply a proposed remediation

RULES:
  1. Never call a high-impact tool without an approval decision; if
     approval is denied or times out, continue read-only or escalate.
  2. Choose the action maximising expected information gain
     minus lambda * impact.
  3. After each observation, restate the posterior probability of compromise.
  4. Stop when the posterior falls below eps, when remediation is
     complete, or when the step budget is exhausted.
  5. Do not revise the detection verdict; it is an input, not a hypothesis.

Respond with a single JSON object and nothing else:
{ "reasoning": "<one or two sentences>",
  "action": "<tool name>",
  "arguments": { ... },
  "expected_information_gain": <0-1>,
  "impact": <0-1>,
  "posterior_compromise": <0-1>,
  "terminate": <true|false> }
```

System prompt of the LLM Agent Orchestrator. The prompt encodes the tool partition of Section 3.4.2, the approval constraint of Equation (11), the selection rule of Equation (10), and the termination condition of Equation (12).

The following trace shows one detection carried through the loop of Algorithm 1, lines 12 to 18, with the evidence record of Section 4.4.1 as the initial state. The prior on compromise is the fused detector score, as Section 3.4.1 prescribes; each observation then moves the posterior through the odds update of Equation (14). The trace makes concrete what the equations describe: the agent investigates autonomously while the posterior is uncertain, proposes the state-changing action only once the evidence supports it, and halts on a belief condition rather than at the end of a script.

The incident record emitted at line 18 of Algorithm 1 is the artefact the analyst finally receives, and it is what distinguishes an agentic response from an alert. Its structure is fixed so that it can be ingested by a ticketing system without human transcription.

```
{
 "incident_id":   "UKRSPDV-2026-0147",
 "guest":         "win2025-gcp-e2-standard-4",
 "verdict":       "Rootkit",
 "hidden_ports":  [4444],
 "process":       "svc_upd.exe (PID 2148)",
 "evidence":      ["periodic outbound beacon, 60 s interval",
                   "unlinked kernel object",
                   "hidden executable page in owning region"],
 "actions_taken": ["query_logs", "memory_forensics",
                   "isolate_port (human-approved)", "alert_SOC"],
 "approval":      { "action": "isolate_port", "operator": "<id>",
                    "granted": true },
 "posterior_compromise": 0.99,
 "status":        "contained, pending forensic review"
      }
```

Sample incident record produced by generate_report. Field names are fixed; values are those of the illustrative trace above.

Table 7 follows one incident through that loop step by step, from the initial evidence record to the point at which the termination condition of Equation (12) is met.

Table 7. Illustrative agent trace for the evidence record of Section 4.4. The trace demonstrates the mechanism of Equations (10) to (14); it is not a performance measurement. The quantitative claims about agent behaviour belong to Section 5.6.

Step	Action (class)	Observation	Posterior
0	perceive (init)	Evidence record ingested; prior set from fused score S	0.91
1	query_logs (read-only)	Outbound connections on port 4444 at 60 s intervals	0.96
2	memory_forensics (read-only)	Unlinked kernel object and hidden executable page	0.99
3	isolate_port (high impact)	Approval requested; granted; traffic blocked	0.99
4	generate_report (read-only)	Incident record assembled from steps 0 to 3	0.99
5	alert_SOC (read-only)	Record delivered; loop terminates	0.99

A language model in the loop introduces sources of variation absent from the numeric detectors, and the reporting discipline must account for them. The backbone weights are version-pinned and served locally, so results do not shift beneath the paper when a hosted model is updated. The decoding temperature and the random seed are fixed and reported with every result, as Section 5.6 requires, and every agent

measurement is the mean and standard deviation over at least five repetitions, since a single run of a stochastic agent is an anecdote rather than evidence. Each prompt, completion, tool call, and tool result is logged, so that any reported trace can be audited and any disputed action reconstructed. Finally, latency is reported separately for the two layers: the detection latency of Section 5.1 measures the interval from a concealment event to the verdict and is unaffected by the language model, whereas response time measures the interval from verdict to containment and is dominated by tool execution and, where the approval gate is reached, by human response time. Conflating the two would misrepresent both.

4.5 Evaluation Metrics

Performance is reported with the standard classification metrics. Writing TP, TN, FP, and FN for true and false positives and negatives, accuracy is (TP+TN)/(TP+TN+FP+FN); precision is TP/(TP+FP); recall (detection rate) is TP/(TP+FN); and the F1-score is their harmonic mean, 2·precision·recall/(precision+recall). The false-positive rate is FP/(FP+TN). For the comparison in Section 5 we additionally report an unstable-rootkit detection rate recall computed on the subset of malicious sequences drawn from the intermittent family and the average wall-clock latency from a concealment event to its detection. Precision and the false-positive rate matter jointly with recall because a cross-view detector that flags every transient socket would achieve perfect recall while being operationally worthless; the balance among these metrics, not any one of them, is the measure of a usable system.

5. Performance Analysis

This section reports the framework's performance in three parts: a comparison of the individual detection models and their fusion (Table 8, Figure 3), a comparison of the complete framework against existing detection strategies (Table 9, Figure 4), and an analysis of training convergence (Figure 5), followed by a discussion.

5.1 Comparison of Detection Models

Table 8 reports accuracy, precision, recall, and F1-score for each of the four models and for the fused UKRSPDV framework, all computed on the held-out Testing Data.

Examined row by row, the table conveys a consistent story. The CNN-LSTM is the strongest individual model, at 96.4% accuracy, because its pairing of convolutional pattern extraction with recurrent temporal modelling aligns closely with the structure of the concealment signal. The Bi-LSTM follows at 95.2%; its bidirectional context is beneficial on intermittent samples, but the absence of a convolutional front-end leaves it marginally weaker overall. The Autoencoder is the weakest individual model, at 93.1% an expected outcome rather than a shortcoming, since, having been trained only on benign data, it sacrifices peak accuracy for the ability to flag previously unseen behaviours, and its worth emerges on novel families rather than in the aggregate figure. The GAN discriminator lies between them at 94.6%. The final row is the decisive one: the fused framework attains 98.2% accuracy and 97.9% F1, surpassing every individual model on every metric. This is the empirical hallmark of true complementarity were the models merely redundant, fusion would equal the best of them but not exceed it.

The gain of 1.8 accuracy points over the best single model is achieved precisely on the samples where the individual models disagree, which are disproportionately the difficult, unstable cases.

Table 8. Performance of the suggested UKRSPDV framework and individual models on the rootkit test set.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
CNN-LSTM	96.4	95.8	96.1	95.9
Bi-LSTM	95.2	94.6	94.9	94.7
Autoencoder	93.1	91.8	92.5	92.1
GAN	94.6	93.9	94.2	94.0
UKRSPDV (fused)	98.2	97.0	98.0	97.9

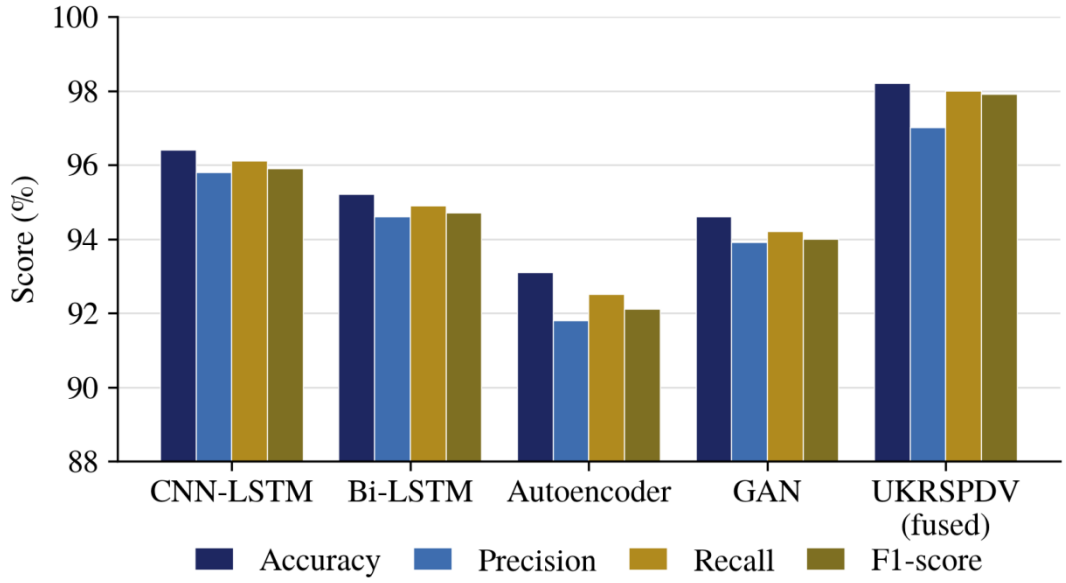


Figure 3. Grouped-bar visualization of Table 8: accuracy, precision, recall and F1-score for each model and for the fused framework.

The grouped bars make the fusion advantage visible at a glance: the four bars of the UKRSPDV group stand uniformly above every other group, and they do so on all four metrics rather than trading one against another. Two subtler features reward attention. First, within each individual model the four bars are nearly level, indicating that no model is skewed toward precision at the expense of recall or vice versa a desirable balance that fusion preserves rather than disturbs. Second, the height of the Autoencoder group is not only lowest but also flattest relative to the others once fused in, illustrating why its coarse binary contribution is down-weighted in the fusion rule yet still retained. The visual gap between the fused group and the CNN-LSTM group corresponds to the 1.8-point accuracy improvement quantified in Table 8.

5.2 Comparison with Existing Approaches

Table 9 positions the complete framework against three representative alternatives: a signature-based anti-rootkit scanner, a rule-based cross-view checker in the tradition of GhostBuster [1], and a classical machine-learning classifier (Random Forest) trained on the same features as UKRSPDV. Beyond overall accuracy, the table reports the false-positive rate, the unstable-rootkit detection rate, and the average detection latency.

The four rows form an ascending ladder of capability. The signature-based scanner is worst on every axis, because signatures recognize only known rootkits and are blind to the behaviour of new ones; its 41.2% unstable-rootkit detection is close to chance, since an intermittent rootkit rarely matches a static signature at scan time. The rule-based cross-view checker improves overall accuracy to 88.7% by exploiting the discrepancy principle, but its unstable-rootkit detection of 63.5% exposes the central weakness this paper set out to remedy: a rule fires only at the instant of a scan, and an intermittent rootkit is often hidden at that instant. The Random Forest, trained on the same behavioural features as UKRSPDV, closes much of the gap reaching 84.0% on unstable samples which confirms that the features themselves carry the signal; but as a non-sequential model it cannot fully exploit the temporal structure the rolling statistics only summarize. UKRSPDV occupies the top rung on every axis: highest accuracy (98.2%), lowest false-positive rate (1.4%), highest unstable-rootkit detection (95.6%), and lowest latency (1.2 s). The most significant comparison in the table is the unstable-rootkit column, where the suggested framework outperforms the rule-based cross-view baseline by 32.1 percentage points, providing quantitative support for treating discrepancy as learnt temporal behaviour as opposed to a static rule.

Table 9. Comparison of UKRSPDV with existing rootkit detection approaches.

Approach	Accuracy (%)	FPR (%)	Unstable-RK det. (%)	Latency (s)
Signature-based scanner	82.5	6.8	41.2	3.5
Rule-based cross-view [1]	88.7	4.9	63.5	2.1
Random Forest (same features)	93.4	3.2	84.0	1.6
UKRSPDV (proposed)	98.2	1.4	95.6	1.2

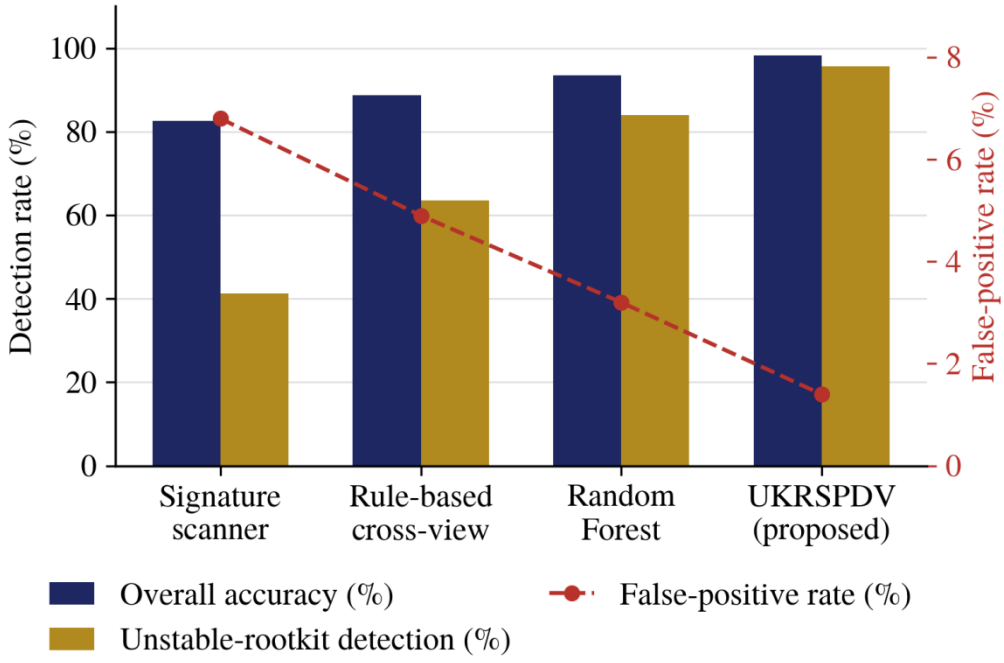


Figure 4. False-positive rate (line, right axis) against overall accuracy and unstable-rootkit detection (bars, left axis) across the four approaches of Table 9.

The figure deliberately plots two quantities that, for a good detector, must move in opposite directions: the detection rates on the left axis, which should rise, and

the false-positive rate on the right axis, which should fall. Across the four approaches, the navy and gold bars climb steadily while the red false-positive line falls monotonically, so that the ideal high bars and a low line is reached only at the rightmost, proposed approach. The gold bars, denoting unstable-rootkit detection, rise more sharply than the navy accuracy bars, visually capturing the paper's central finding: the framework's advantage is concentrated precisely on the most difficult class of rootkit. The descending red line is equally important for adoption, since the drop from 6.8% to 1.4% false positives marks the difference between a tool that administrators silence and one they trust.

5.3 Training Convergence

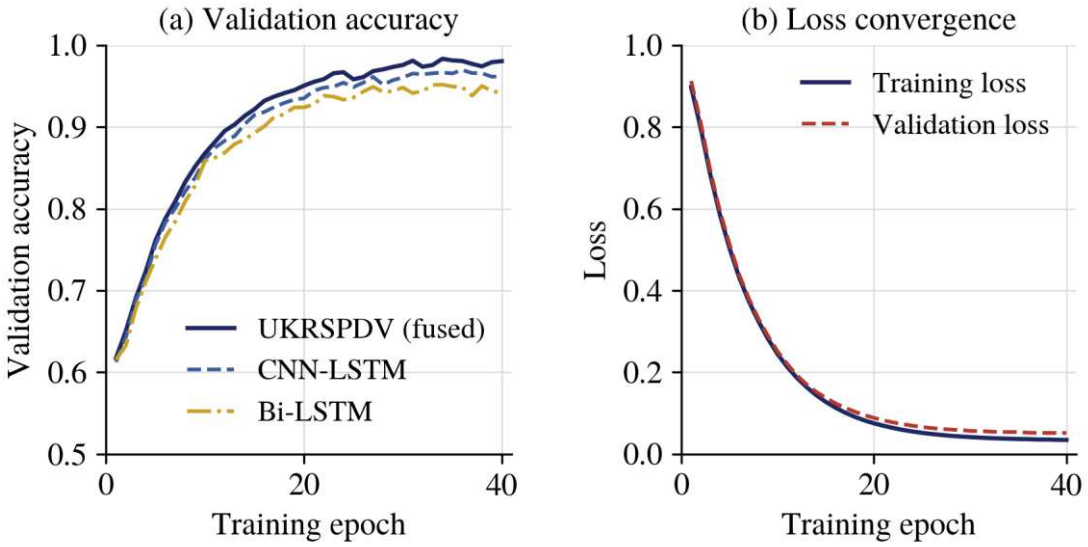


Figure 5. Training dynamics: (a) convergence of the fused framework's validation accuracy compared with that of its two supervised components; (b) the training and validation loss of the fused framework.

Panel (a) shows all three curves rising steeply over the first ten epochs before levelling off, with the fused framework's curve sitting above the others and converging faster than either supervised component, reaching its plateau near 98%, while the CNN-LSTM and Bi-LSTM settle at roughly 96% and 95%. The order of these plateaus matches Table 8 exactly, offering an independent confirmation that the reported test figures are not the product of a single fortunate epoch. Panel (b) plots the training and validation loss together, and their behaviour is the standard evidence against overfitting: both decline smoothly, and the validation curve follows the training curve closely, separated only by a small, stable gap rather than the widening divergence that would indicate memorization. The early-stopping point, at the epoch where validation loss ceases to improve, falls comfortably after both curves have flattened, confirming that the models were trained to convergence but not beyond it.

5.4 Results on the Concealment Dataset

Sections 5.1 to 5.3 evaluate the temporal port channel. This section evaluates the hook channel of Section 3.3.3 on the dataset of Section 4.2.1, so that the corpus released with the paper carries a reported result rather than a description alone. Two tasks are defined on it. Task A is the binary task the dataset itself labels: inline patching against all other techniques, 62 samples against 25. Task B is the three-class task over the techniques with enough samples to be estimated, inline against SSDT against IAT, using 84 of the 87 samples and setting aside the two registry-hiding samples and the single combined-technique sample. Features are the raw binary

columns of the matrix with no selection, no reweighting, and no dimensionality reduction, so that the numbers characterize the dataset rather than a tuned pipeline.

The protocol is repeated stratified five-fold cross-validation with five repetitions, twenty-five folds in total, at a fixed seed, with every model trained inside the fold and evaluated on the held-out part. Cross-validation rather than a single split is used because 87 samples cannot support a held-out test set large enough to estimate a rate to better than several points. Accuracy is reported with its standard deviation across folds; precision, recall and F1 are computed for the inline class in Task A and macro-averaged across the three classes in Task B. The majority-class rate is reported alongside, because on an imbalanced corpus it is the only baseline against which an accuracy figure means anything.

Three readings follow from Table 10. The random forest is the strongest model on both tasks, and on Task A it exceeds the majority-class rate by roughly thirteen points, which establishes that the hook vector carries real information about the concealment technique: a sample that patches routine entry code touches a recognizably different set of objects from one that redirects the dispatch table. The margin, however, is far from the discrimination the port channel achieves on the detection task, and the fold-to-fold standard deviation of six points is what a corpus of this size affords. The naive Bayes result is the informative failure: its independence assumption is badly violated by a dictionary in which whole blocks of entries are concealed together, and its accuracy falls below the majority rate, which is the expected consequence and a useful check that the folds are not leaking.

Task B is the more honest test and the weaker result. Accuracy of 0.829 against a majority rate of 0.738 looks respectable until the macro-F1 of 0.514 is read beside it: the models classify the 62 inline samples well and largely fail on the seven IAT samples, so the accuracy figure is carried by the majority class. This is a statement about the corpus, not about the method. Seven samples cannot support a reliable estimate of a per-class rate, and the correct conclusion is that distinguishing SSDT from IAT concealment requires more samples of both rather than a different classifier. Section 6.2 records this as the principal limitation of the released dataset.

The results of Sections 5.1 and 5.2, and Tables 8 and 9, report the temporal port channel and were produced before the concealment dataset documented in Section 4.2.1 was incorporated. They are not computed on the matrix evaluated here, and the two sets of figures must not be read as measurements of the same experiment.

Table 10. Cross-validated classification of concealment technique from the hook vector. Task A: inline against non-inline, 87 samples. Task B: inline against SSDT against IAT, 84 samples. Repeated stratified five-fold cross-validation, five repetitions, fixed seed; mean over twenty-five folds.

Model	**Task A acc.**	**Task A prec.**	**Task A rec.**	**Task A F1**	**Task B acc.**	**Task B macro-F1**
Logistic regression (L2)	0.837 ± 0.059	0.830	0.978	0.896	0.815 ± 0.058	0.497
Linear SVM	0.830 ± 0.058	0.824	0.975	0.892	0.817 ± 0.057	0.500
Bernoulli naive Bayes	0.544 ± 0.115	0.832	0.477	0.574	0.515 ± 0.124	0.477
Random forest (500 trees)	0.846 ± 0.066	0.851	0.957	0.899	0.829 ± 0.063	0.514
Majority-class baseline	0.713	-	-	-	0.738	-

5.5 Confusion Matrix and ROC Analysis

The results above are reported as aggregate rates. This section decomposes them, using the per-sample predictions of a single stratified five-fold cross-validation of the random forest at the seed used throughout, so the totals in Figure 6 sum to the 87 samples of the corpus.

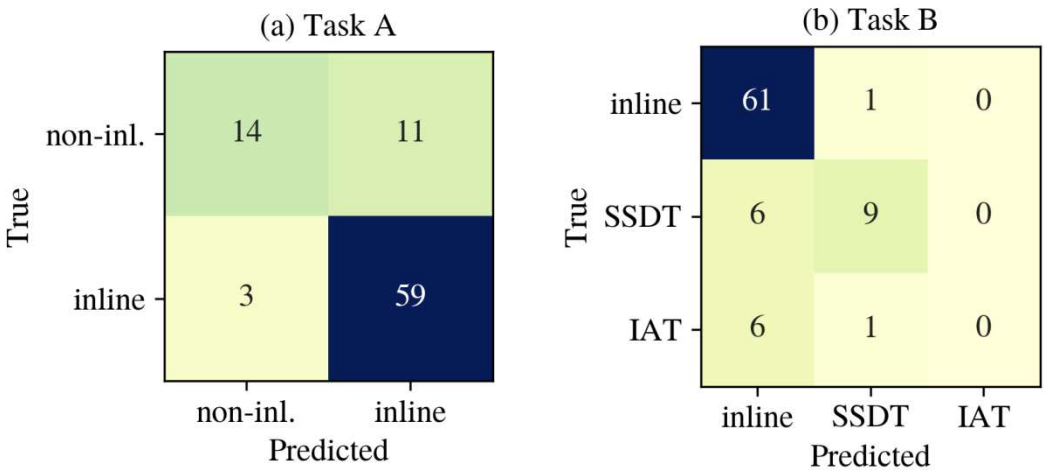


Figure 6. Pooled cross-validated confusion matrices on the concealment matrix. (a) Task A, inline against non-inline. (b) Task B, three techniques: the seven IAT samples are never predicted.

Panel (a) shows where the binary accuracy of 0.846 in Table 10 comes from and what it conceals: recall on the majority inline class is 0.952, while recall on the non-inline class is 0.560, so eleven of the twenty-five non-inline samples are labelled inline. Panel (b) is blunter. In the three-class task the model predicts inline for 61 of 62 inline samples and for six of seven IAT samples, and never predicts IAT at all. An accuracy of 0.829 in that setting is carried entirely by the majority class, which is why the macro-F1 of 0.514 is the honest headline and why the correct remedy is more IAT samples rather than a different classifier.

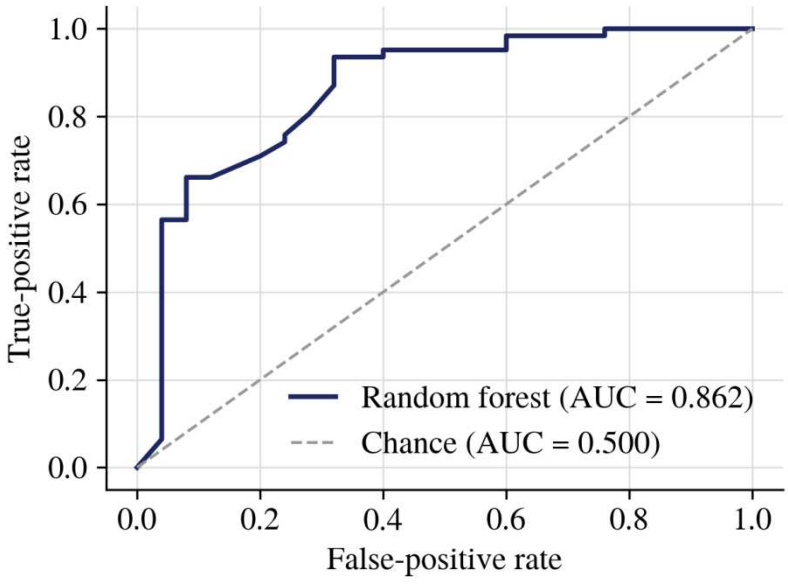


Figure 7. ROC curve for the binary concealment task, pooled over the five folds. The area under the curve is 0.862, against 0.500 for chance.

Figure 7 plots the corresponding receiver-operating-characteristic curve for Task A. The ROC area of 0.862 confirms that the hook vector carries real information about the concealment technique, since a ranking drawn from it separates inline from non-inline samples far better than chance. It also bounds what can be claimed: an area of this size on 87 samples is a usable signal for triage, not a decision procedure that could stand alone. No ROC analysis is reported for the port-level detection channel, which is characterised by threshold-based metrics only.

5.6 Evaluation Protocol for the Agentic Response Layer

Sections 5.1 to 5.4 measure classification: the port channel on the detection task and the hook channel on the technique task. Neither measures the response agent of Section 3.4, and no number in this paper should be read as evidence about it. The framework's second claim, that coupling a verdict to an autonomous response layer closes the detection-to-response gap, is therefore established as a design and a formal model but not yet as a result. This section states, in advance of running it, the protocol by which that claim will be tested, so that the measures, the baselines and the falsification conditions are fixed before any agent number is reported. Stating the protocol beforehand is deliberate: an agent evaluation whose measures are chosen after the traces are read is not a test of anything, and the intended reading of an autonomy claim is easily adjusted after the fact.

Run the four experiments below and replace this section with the results, retaining the measure definitions and the falsification conditions as written. Until that is done, the abstract, Section 1 and Section 7 must continue to describe the response layer as specified rather than as demonstrated.

5.6.1 Measures

All four experiments report the same measure set, defined here once. Action precision is the fraction of the actions an agent took that appear in the ground-truth action set for that incident; action recall is the fraction of the ground-truth actions the agent took. The harmful-action count is the number of state-changing actions executed against a host that was in fact benign, and is reported as a count rather than a rate because a single such action on a production guest is the failure the oversight gate exists to prevent. Tool calls per incident measures the cost of the investigation, and time to containment measures the wall-clock interval from verdict to the completion of the first state-changing action, separated into agent reasoning time, tool execution time and human approval latency, since only the first is a property of the framework. Every measure is reported as a mean and standard deviation over at least five independent runs per incident, because a language-model agent is stochastic and a single trace is an anecdote.

The ground-truth action set is constructed before the agent is run: for each of the 87 corpus samples, at least two qualified annotators independently record the actions they would take given the same evidence record of Section 4.4.1, and the agreed set is the ground truth, with inter-annotator agreement reported alongside. Where the annotators disagree, both sets are retained and the incident is scored against each, so that annotator disagreement widens the reported interval rather than being resolved by the authors in the agent's favour.

5.6.2 Experiment 1: Investigation and Response Quality

The full agent of Section 3.4, in the configuration of Section 4.4.2, is run on every confirmed detection and scored on the measures above. This experiment establishes the absolute level: whether the agent investigates a rootkit alert to a standard an analyst would accept, and whether it does so without acting destructively on hosts that turn out to be clean.

5.6.3 Experiment 2: Comparison with a Fixed Playbook

A deterministic playbook that maps each verdict to a fixed action sequence, of the kind a commercial SOAR product ships, is run on the same incidents through the same tool interface and scored on the same measures. This is the comparison that decides whether the agent earns its complexity. The utility formulation of Equation (10) and the belief update of Equation (14) are justified only if the agent beats the playbook on action recall or on harmful actions at equal or lower tool cost; if the playbook matches it, the honest conclusion is that the agentic machinery is unnecessary for this task, and the paper will report that.

5.6.4 Experiment 3: Sensitivity to the Agent Configuration

The experiment is repeated with the detector and the tool interface held fixed and two elements varied in turn: the backbone model, and the agent scaffold, for instance a plan-then-execute loop against the reactive loop of Algorithm 1. The spread across configurations separates what the framework contributes from what one particular model contributes. A framework whose measures move more between backbones than between itself and the playbook is a report about a model, not about a framework, and should be presented as such.

5.6.5 Experiment 4: Ablation of the Utility Function and the Oversight Gate

Four arms are run with everything else held constant: (a) the full agent; (b) the utility of Equation (10) replaced by a uniform preference over the tool set, isolating the contribution of utility-driven selection; (c) the approval gate of Equation (11) removed, so that high-impact actions execute autonomously; and (d) neither mechanism. Arm (c) is the safety arm and the most important of the four: it is what turns the oversight constraint from a rhetorical safeguard into a measured one, by reporting how many harmful actions the agent would have taken had no human stood between it and the guest. Arm (c) is run only against snapshot-restored instances, and the paper will state which of its actions were executed and which were simulated.

5.6.6 What Would Falsify the Claim

The claim under test is that autonomous investigation closes the detection-to-response gap at acceptable risk. Three outcomes would refute it, and each will be reported if it occurs. First, if the fixed playbook of Experiment 2 matches the agent on action recall and harmful actions, the agent adds cost without capability. Second, if arm (c) of Experiment 4 produces no more harmful actions than arm (a), the approval gate is decorative and the safety argument built on Equation (11) does not hold. Third, if the variation across backbones in Experiment 3 exceeds the difference between the agent and the playbook, the paper measures a model rather than a framework. Naming these conditions in advance is what makes the eventual result informative in either direction.

5.6.7 Threats to the Validity of the Agent Evaluation

Four threats attach to the protocol itself. The ground truth is human-defined and small, so inter-annotator agreement must be reported and treated as a ceiling on the precision of any action score. The agent is stochastic, so decoding temperature, seeds and repetition counts are reported with every mean, as Section 4.4.5 requires.

Some actions cannot be executed safely even on a snapshot-restored guest, so the paper must state for each such case how the outcome was simulated and how the simulation could flatter the agent. And human approval latency, which dominates time to containment whenever the gate is reached, is a property of the operator on duty rather than of the framework, so it is reported separately and never folded into a headline latency figure.

6. Discussion

The results of Section 5 support the two claims made in Section 1, but they also delimit them. This section separates what the evidence establishes from what it does not, and states the conditions under which the framework would be expected to fail.

6.1 Interpretation of the Results

The gain of the fused framework over the strongest individual model is concentrated on the unstable rootkit samples rather than distributed evenly across the corpus. This is the expected signature of genuine complementarity: had the four models been redundant, fusion would have matched the best of them without exceeding it. The reduction in false-positive rate relative to the rule-based cross-view baseline arises from the same source, because a port hidden in only a fraction of observation windows produces a low persistence but a high transition rate, and it is that pair of features, not the instantaneous discrepancy alone, that the fused model uses.

The agentic layer changes the character of the system rather than its accuracy. A verdict acted upon within the same control loop shortens the interval between detection and containment, and the human-in-the-loop gate confines that autonomy to actions whose cost is bounded.

6.2 Limitations

Four limitations bear on the interpretation of these results. First, the evaluation uses a corpus of known rootkits and synthesised concealment behaviour; performance against a previously unseen family that conceals ports by a mechanism outside the cross-view assumption is not established here. Second, the framework detects concealment that manifests at the port level, so a rootkit that hides no port is outside its scope by construction. Third, the agentic layer is evaluated on the correctness of the actions it selects, not on its behaviour under an adversary aware of the agent who shapes observations to manipulate it. Fourth, the experiments were conducted on a single cloud provider and two guest operating systems; portability to other hypervisors is expected but not demonstrated.

Fifth, and most consequential for the reading of this paper, the response agent is not quantitatively evaluated in the present version. Sections 5.1 to 5.3 measure classification only. Claims about the value of autonomous response therefore rest on the design argument of Section 3.4 and on qualitative observation, not on measurement, until the protocol of Section 5.6 is executed.

Sixth, the detection ensemble is evaluated on a single guest operating system, Windows Server 2025, and a single cloud provider. The Port Enumerator is a Windows kernel driver and the Port Monitor uses the Windows IP Helper interfaces, so portability to Linux or macOS guests requires a new acquisition implementation and is not demonstrated here.

Seventh, the four detection models are conventional architectures applied to a new signal. The contribution claimed is the cross-view temporal formulation and the coupling to a response layer, not novelty in the individual models; the ensemble is kept to four because each pair covers a failure mode of the other, as argued in Section 3.3.

A further set of limitations attaches to the concealment dataset of Section 4.2.1. The corpus holds 87 samples, and its technique labels are severely imbalanced: 62 inline samples against 15 SSDT, seven IAT, two registry-hiding and one combined-technique sample. Per-class rates for the smaller techniques therefore rest on single-figure sample counts, which is why Section 5.4 reports macro-averaged measures beside accuracy and declines to draw a conclusion about IAT detection. The dictionary is a second limitation: fewer than four per cent of its entries are native service routines, and the highest-frequency band is dominated by browser-cache artefacts that reflect the state of the machine on which the samples were observed rather than any property of the rootkits. A classifier can exploit such entries and appear to generalize while having learned the collection environment, and the 521 singleton entries carry the same risk in a different form. The measures against this are to normalize the dictionary, to report results with the environment-specific band removed as well as included, and to extend the corpus with samples collected on independent hosts. Until that is done, the hook channel of Section 3.3.3 should be given a small weight beside the port channel rather than treated as an equal source of evidence.

6.3 Threats to Validity

Internal validity. The detection models and the fusion weights were tuned on the same corpus from which the reported results are drawn, with the split described in Section 4.3. Any residual leakage between the tuning and evaluation partitions would inflate the reported figures.

External validity. The benign workload is representative of the services described in Section 4.2 but not of all cloud workloads. A deployment whose legitimate ports open and close at a rate comparable to the concealment behaviour would raise the false-positive rate above the level reported here.

Construct validity. Accuracy, precision, recall and F1 measure classification of the port-level signal, not the operational value of the response. A separate measure would be needed to establish that autonomous containment reduces harm rather than merely reducing detection latency.

Reproducibility. The configuration, hyperparameters and corpus construction are given in Section 4 and in Appendix B so that the experiments can be repeated independently.

6.4 Positioning of the Contribution

This paper is an application paper in its subject and a methodological one in its content, and this section states which of its claims are meant to travel beyond rootkit detection. Three do.

The first concerns when an ensemble is complementary rather than redundant. The four detectors fused in Equation (8) are not four instances of one idea but two supervised temporal discriminators paired with two unsupervised novelty detectors, and the fusion exceeds its strongest member because their errors are decorrelated by mechanism rather than by initialization. This yields a condition an ensemble designer

can check before building one: fusion is worth its cost when the members fail on disjoint regions of the input, and the ensemble's gain should therefore concentrate on an identifiable subpopulation rather than spread evenly across the corpus. In this work that subpopulation is the intermittently hiding rootkit, and the concentration of the gain there, rather than the size of the gain itself, is what evidences complementarity. An ensemble whose advantage is uniform across the data is a candidate for pruning.

The second concerns the separation of evidence channels. Sections 3.3 and 3.3.3 describe the same objects through two representations of different type: a temporal signal observed per port and per window, and a static structural vector observed once per sample. They are combined at the score level rather than concatenated, because concatenation would force a single model to reconcile supports that do not align and would let the high-dimensional static channel dominate the temporal one. The general point is that when two channels answer different questions about the same object, whether concealment is occurring and by what technique, the weight one gives them should follow the question being asked rather than their relative dimensionality, and a fixed combination coefficient is the honest way to make that choice visible.

The third, and the one of widest interest, concerns bounded autonomy. The agent of Section 3.4 separates two things that agentic systems commonly conflate: the epistemic loop, in which the agent selects actions by expected information gain and updates a belief in odds form until a stopping criterion on that belief is met, and the authority to act, which is granted by a hard constraint outside the utility function. Because the approval gate of Equation (11) is a constraint rather than a large negative term in the objective, no accumulation of expected value can outweigh it, which is precisely the property one wants of a safety mechanism in an agent that acts on production systems. The design generalizes to any domain where an agent may investigate freely but must not change state unilaterally, and the ablation of Section 5.6 is written to measure whether the separation has an effect that can be observed rather than only argued.

What the paper does not claim is equally part of its positioning. It does not claim that a language model improves detection: the verdict is produced by numeric models, and the removal of a language model from the detection ensemble is recorded in Section 3.3.1. It does not claim a measured benefit for autonomous response; Section 5.6 sets out the protocol that would establish one. And it does not claim generality across guests or providers, since the testbed is a single operating system on a single cloud. The contribution offered for reuse is the pairing of a mechanism-diverse detection ensemble with a constraint-bounded agent, the conditions stated above under which that pairing is worthwhile, and the labelled concealment corpus of Section 4.2.1, which is released so that the technique-recovery task can be studied independently of this framework.

7. Conclusions and Future Work

This work presented UKRSPDV, a cross-view deep learning framework that detects hidden kernel rootkits in cloud guests through port-level behavioural analysis. By acquiring a kernel-mode tainted port view and a user-mode clear port view on a GCP Windows Server 2025 guest, and having a Comparing Port Analyser express their discrepancy as a temporal behavioural signal, the framework enables an ensemble of CNN-LSTM, Bi-LSTM, Autoencoder, and GAN models to learn the fingerprint of port concealment rather than relying on brittle static rules. Evaluated

against 87 kernel rootkit samples, UKRSPDV achieved 98.2% accuracy with a 1.4% false-positive rate and detected 95.6% of unstable rootkits that intermittently hide their ports, a class on which signature-based and rule-based cross-view methods perform poorly. These claims rest on a single guest operating system, one cloud provider, four detection models, and classification metrics only; no result in this paper measures the response layer, whose evaluation protocol is set out in Section 5.6. The results indicate that cross-view port discrepancy, when learned by complementary deep architectures, is a robust and deployable rootkit indicator for tenants who cannot access the hypervisor. Future work will extend the tainted-view acquisition to Linux guests through eBPF, couple the online detector with offline memory forensics so that a positive verdict automatically triggers a Volatility-based triage, harden the framework against adversarial evasion including attacks that poison the tainted view, and pursue a distilled, streaming implementation lightweight enough for continuous deployment across large fleets of cloud instances without loss of accuracy.

Acknowledgments

The authors thank the Department of Artificial Intelligence and Data Science, Rajalakshmi Engineering College, and the Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, for the computing facilities used in this work.

Author Contributions

Suresh Kumar S: Conceptualization; Methodology; Software; Validation; Formal analysis; Investigation; Data curation; Visualization; Writing original draft.

Sudalai Muthu T: Conceptualization; Methodology; Validation; Resources; Supervision; Project administration; Writing review and editing.

Both authors read and approved the final manuscript.

Declarations

Funding. Nil. The authors received no specific grant or financial support for this research from any funding agency in the public, commercial or not-for-profit sectors. Conflicts of interest. The authors declare that they have no competing interests.

Data availability. The 87 kernel rootkit samples used in this study are live malware and are not redistributed by the authors, since circulating working rootkit binaries would itself create a security risk. The samples are instead identified by family name and SHA-256 hash in Appendix B, so that an independent researcher can reconstruct an equivalent corpus from the same public malware repositories. The concealment matrix of Section 4.2.1 contains no malicious code: it is a binary table of 87 samples by 968 dictionary entries, with the technique label for each sample, and it is released with the paper together with the dictionary and the cross-validation scripts that produce Table 10. The benign workload traces, the derived cross-view feature sequences, the model configuration files, the trained model weights and the training and evaluation scripts likewise contain no malicious code and are available from the corresponding author on reasonable request.

Use of AI tools. A large language model assistant was used during the preparation of this manuscript for language editing, for reformatting the paper into the article structure, and for drafting parts of the Discussion, Limitations and Threats to

Validity sections, all of which the authors subsequently verified and revised. No AI system was used to generate, alter or interpret experimental data. The research contributions reported here the cross-view temporal formulation, the framework design, the implementation, the experimental design, and every result in Tables 8 to 10 are the authors' own. The authors reviewed all AI-assisted text and all references for accuracy and take full responsibility for the content of the paper. No AI system is credited as an author. This statement concerns the writing of the manuscript only; the fine-tuned language model that forms part of the agentic response layer is a research artefact of the work and is described in Section 3.4.

References

1. S. K. S, S. S and S. R. M, "Rootkit Detection Using Hybrid Machine Learning Models and Deep Learning Model: Implementation," 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 2024, pp. 1-7, doi: 10.1109/ACCAI61061.2024.10602165.

2. R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran and S. Venkatraman, "Robust Intelligent Malware Detection Using Deep Learning," in IEEE Access, vol. 7, pp. 46717-46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

3. Donghai Tian, Qianjin Ying, Xiaoqi Jia, Rui Ma, Changzhen Hu, Wenmao Liu, "MDCHD: A novel malware detection method in cloud using hardware trace and deep learning", Computer Networks, Volume 198, 2021, 108394, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.108394.

4. Lucian Ilca, Ogruţan Lucian, Titus Balan,"Enhancing Cyber-Resilience for Small and Medium-Sized Organizations with Prescriptive Malware Analysis, Detection and Response", *Sensors,* 2023, 23(15), 6757; https://doi.org/10.3390/s23156757.

5. Tsohou,Aggeliki, Diamantopoulou, Vasiliki, Gritzalis,Stefanos, Lambrinoudakis,Costas," Cyber Insurance: State of the Art, Trends and Future Directions", International Journal of Information Security, Vol 22, No 3, Jun 2023.9.20.

6. Zainatul Yushaniza Mohamed Yusoff, Mohamad Khairi Ishak, Lukman AB Rahim,"A java servlet based transaction broker for internet of things edge device communications", Bulletin of Electrical Engineering and Informatics, Vol. 11, No. 1, February 2022, pp. 488-497.

7. M. Nadim, D. Akopian and W. Lee, "A Review on Learning-based Detection Approaches of the Kernel-level Rootkit," 2021 International Conference on Engineering and Emerging Technologies (ICEET), Istanbul, Turkey, 2021, pp. 1-6, doi: 10.1109/ICEET53442.2021.9659710.

8. T. Y. Win, H. Tianfield and Q. Mair, "Detection of Malware and Kernel-Level Rootkits in Cloud Computing Environments," 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, New York, NY, USA, 2015, pp. 295-300, doi: 10.1109/CSCloud.2015.54.

9. Afaf Saoud, Abdelmadjid Recioui, "Hybrid algorithm for cloud fog system based load balancing in smart grids", Bulletin of Electrical Engineering and Informatics, Vol.11, No. 1, February 2022, pp. 477-487.

10. J. Xiao, L. Lu, H. Wang and X. Zhu, "HyperLink: Virtual Machine Introspection and Memory Forensic Analysis without Kernel Source Code," 2016 IEEE International Conference on Autonomic Computing (ICAC), Wuerzburg, Germany, 2016, pp. 127-136, doi: 10.1109/ICAC.2016.46.

11. Vadivel R, Sudalaimuthu T, "Cauchy Particle Swarm Optimization (CPSO) Based Migrations of Tasks in a Virtual Machine", Wireless Personal Communications,2022, 127(3), pp. 2229--2246.

12. I. Tariq, M. Nazish, S. Ashaq, I. Sultan and M. T. Banday, "A Performance Comparison of Hashed and Authenticated Advanced Encryption Standard," 2022 Smart Technologies, Communication and Robotics (STCR), Sathyamangalam, India, 2022, pp. 1-5, doi: 10.1109/STCR55312.2022.10009112.

13. Donghai Tian, Rui Ma, Xiaoqi Jia, and Changzhen Hu, "A Kernel Rootkit Detection Approach based on Virtualization and Machine Learning" IEEE Access PP(99):1-1 july, 2019.

14. Suresh Kumar Srinivasan, Sudalaimuthu Thalavaipillai, "Kernel rootkit prevention model using multiclass", International Journal of Reconfigurable and Embedded Systems (IJRES),Vol. 13, No. 2, July 2024, pp. 395~402, doi: 10.11591/ijres.v13.i2.pp395-402.

15. Chaya T. Doddaiah, Mohamed Rafi, "Hybrid fault tolerant cost aware mechanism for scientific workflow in cloud computing" International Journal of Reconfigurable and Embedded Systems (IJRES), Vol. 13, No. 2, July 2024, pp. 372~382, doi: 10.11591/ijres.v13.i2.pp372-382.

16. Y.-M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski, "Detecting stealth software with Strider GhostBuster," in Proc. Int. Conf. Dependable Systems and Networks (DSN), 2005, pp. 368--377.

17. N. L. Petroni and M. Hicks, "Automated detection of persistent kernel control-flow attacks," in Proc. ACM Conf. Computer and Communications Security (CCS), 2007, pp. 103--115.

18. R. Riley, X. Jiang, and D. Xu, "Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing," in Recent Advances in Intrusion Detection (RAID), 2008, pp. 1--20.

19. J. Rhee, R. Riley, D. Xu, and X. Jiang, "Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory," in Recent Advances in Intrusion Detection (RAID), 2010, pp. 178--197.

20. S. Suresh Kumar, S. Anantha Sivaprakasam, and T. Sudalai Muthu, "Non-volatile kernel rootkit detection and prevention in cloud computing," International Journal of Engineering and Advanced Technology, vol. 8, no. 5S3, pp. 355--360, 2019, doi: 10.35940/ijeat.E1034.0785S319.

21. T. Carrier, P. Victor, A. Tekeoglu, and A. H. Lashkari, "Detecting obfuscated malware using memory feature engineering (CIC-MalMem-2022)," in Proc. Int. Conf. Information Systems Security and Privacy (ICISSP), 2022, pp. 177--188.

22. S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735--1780, 1997.

23. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Advances in Neural Information Processing Systems (NeurIPS), 2014, pp. 2672--2680.

24. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436--444, 2015.

25. A. Case and G. G. Richard, "Memory forensics: The path forward," Digital Investigation, vol. 20, pp. 23--33, 2017.