

LLM-Enabled Cloud-Native Dynamic Honeypot Systems: Architecture, Ethical Governance, and Empirical Evaluation

Shang E. Tsai

sean@mail.cjcu.edu.tw

Chang Jung Christian University

Ting T. Tsai

Financial Supervisory Commission

Meng H. Aun

Chang Jung Christian University

Research Article

Keywords: Cyber deception, honeypots, large language models, cloud-native security, ethical governance, containment

Posted Date: March 9th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9003091/v1>

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

[Read Full License](#)

Additional Declarations: No competing interests reported.

LLM-Enabled Cloud-Native Dynamic Honeypot Systems: Architecture, Ethical Governance, and Empirical Evaluation

Shang E. Tsai^{1*}, Ting T. Tsai² and Meng H. Aun¹

¹Department of Computer Science and Information Engineering, Chang Jung Christian University, Tainan City 71101, Taiwan.

²Financial Supervisory Commission, New Taipei City 22041, Taiwan

*Correspondence: sean@mail.cjcu.edu.tw; Tel.: +886-6-2785123 ext. 6154

Abstract

Large Language Models (LLMs) can substantially improve honeypot interaction realism, but naïve integration increases operational risk (e.g., prompt injection, unsafe guidance, state inconsistency, and resource exhaustion). This paper presents an LLM-enabled, cloud-native dynamic honeypot architecture that treats the LLM as a strictly text-only synthesizer behind deterministic policy gating and state verification. The exposed SSH/Web surfaces are mediated by a session broker that never executes attacker commands; instead, commands are classified into deterministic emulation, bounded LLM synthesis, plausible error simulation, or quarantined payload capture. The system is deployed as decomposed microservices with deny-by-default networking, controlled egress, authenticated internal service calls, and centralized tamper-evident telemetry. To make the deployment ethically and legally defensible, we operationalize a principlist governance framework into concrete controls including data minimization, bounded retention, access governance, and abuse-rate limiting. Finally, we provide an IJIS-aligned evaluation protocol that separates background Internet scanning noise from adaptive interactive sessions and reports realism, engagement, fingerprint resistance, and safety metrics, including timing-based distribution tests against a real OpenSSH baseline. The resulting design offers a practical path to high-fidelity deception with auditable containment and reproducible measurement.

Keywords

Cyber deception; honeypots; large language models; cloud-native security; ethical governance; containment

1. Introduction

Internet-facing services face a continuous stream of automated probing, credential stuffing, and opportunistic exploitation. While perimeter controls (e.g., firewalls, IDS/IPS, and rate limiting) can block known patterns, they are primarily reactive and provide limited insight into attacker intent and workflows. Honeypots complement these defenses by attracting adversaries to controlled decoy environments, enabling richer telemetry capture and supporting threat intelligence.

Recent Internet-wide measurement platforms (e.g., GreyNoise) characterize a substantial fraction of inbound attempts against newly exposed services as “Internet noise,” generated by distributed scanners and survey infrastructure rather than targeted,

interactive adversaries [1]. This motivates an explicit separation between (i) background scanning noise and (ii) adaptive probing sessions in both our threat model and evaluation protocol, ensuring that engagement- and realism-oriented metrics are interpreted under realistic Internet exposure conditions.

A long-standing tension in honeypot design is the realism–risk trade-off. Low-interaction honeypots are safer but often produce shallow interactions and are susceptible to fingerprinting. High-interaction honeypots can yield deeper behavioral traces but introduce containment, liability, and operational risks—especially if attackers can pivot outward.

LLMs create an opportunity to improve realism without granting execution capability. However, naive LLM integration can itself increase risk (e.g., unconstrained guidance, state inconsistency, or prompt injection). This paper argues that LLMs must be integrated as a tightly constrained component within a policy-driven architecture and cloud-native isolation model.

Contributions: (1) a policy-gated, state-aware LLM interaction layer for SSH/Web deception; (2) a cloud-native microservice deployment model with deny-by-default containment and centralized telemetry that supports attack-chain reconstruction across distributed sensors; (3) an ethical and legal governance framework grounded in principlism and mapped to technical controls; and (4) an IJIS-aligned evaluation protocol and result templates to support reproducible comparison.

2. Related Work

This section positions our contribution at the intersection of cyber deception, cloud-native honeypot engineering, and LLM-enabled interactive realism. We emphasize (i) how prior deception systems balance fidelity vs. operational risk, (ii) how modern containerized platforms operationalize multi-service honeypots, and (iii) how emerging LLM-honeypots introduce new evaluation and governance requirements. Our key novelty is an IJIS-oriented, end-to-end design that couples LLM-driven interaction with explicit policy gating, cloud-native isolation, and auditable ethical controls.

2.1 Honeypots and cyber deception

Honeypots are widely used for early warning and threat intelligence, but their effectiveness is constrained by detectability and limited interaction depth. Recent surveys and comparative frameworks on cyber deception highlight a broad design space—interaction levels, fingerprint resistance, adaptive content, and operational safeguards—and emphasize that realism must be paired with containment and measurement to remain useful against evolving adversaries [2-3].

2.2 Cloud-native honeypot platforms and microservice deployment

At scale, operational concerns (repeatable provisioning, geographically distributed vantage points, and centralized telemetry) motivate cloud-native deployment. T-Pot exemplifies an "all-in-one" multi-honeypot platform that orchestrates multiple services via containers and integrates telemetry pipelines for analysis [4]. For interactive SSH deception, Cowrie provides medium/high-interaction capabilities, including an emulated filesystem backed by a pickle metadata store with per-session isolation [5]. For Web deception, SNARE/TANNER decouple the sensor from a decision engine to produce dynamic responses that improve camouflage against scanners and fingerprinting [6]. Network security monitoring commonly complements application logs; Suricata provides IDS/IPS/NSM telemetry suitable for correlating flows and alerts with honeypot events [7].

2.3 LLM-enabled interactive honeypots and evaluation paradigms

LLM-based honeypots seek to overcome deterministic and shallow responses by generating plausible, context-aware outputs. Early work (e.g., shellLM) demonstrated that LLMs can produce credible shell responses and improve human-perceived realism in interactive sessions [8]. Subsequent systems incorporate adversarial or robustness considerations, such as strengthening resistance to probing and persona breaks in LLM-interactive honeypots [9]. A recent systematization of knowledge (SoK) synthesizes architectural patterns, detection vectors, and emerging evaluation trends for LLM-powered honeypots, highlighting the need for standardized metrics and operationally safe designs [10]. Complementary to deception design, tooling such as Beekeeper aims to accelerate honeypot development by automating interaction tests and LLM-driven feedback loops for medium-to-high interaction honeypots [11]. Recent studies further refine the design space of terminal-facing LLM honeypots. HoneyGPT explicitly targets the realism–safety–consistency “trilemma” and emphasizes mechanisms for preserving a stable Linux persona under adversarial probing, which directly motivates our policy-gated, state-verifiable

response synthesis pipeline [12]. Complementarily, HoneyLLM explores medium-interaction LLM honeypot designs and evaluation practices that generalize beyond a single scripted command set, reinforcing the need for explicit state handling and measurable persona stability metrics in multi-turn sessions [13]. Protocol-specific LLM honeypots—such as LDAP directory-service deception and database-facing (MySQL) deception—supports our argument that cloud-native, containerized sensor decomposition is the most practical path to building credible enterprise-scale honey networks [14-15].

2.4 Security risks of LLM integration in defensive systems

Integrating LLMs into security infrastructure introduces new attack surfaces. Prompt injection is a prominent class of attacks against LLM-integrated applications, where adversaries attempt to manipulate the model into following malicious instructions [16]. OWASP has formalized common risk categories for LLM applications, including prompt injection, insecure output handling, and model denial-of-service, all of which are directly relevant to an LLM-backed honeypot control plane [17]. Meanwhile, evidence that agentic LLM systems can autonomously conduct complex web attacks and exploit vulnerabilities underscores the pace at which attacker capabilities may scale, increasing the value of high-fidelity, measurable deception while also raising dual-use concerns [18], [19]. Recent in-the-wild measurement work has begun using SSH honeypots to detect and characterize autonomous LLM hacking agents by combining prompt-injection probes with timing analysis, suggesting that agentic adversaries are an operationally plausible threat model rather than a purely hypothetical one [20].

2.5 Ethics, privacy, and governance for honeypot operations

Honeypot telemetry often includes online identifiers such as IP addresses, creating privacy and compliance considerations. Prior work discusses privacy risks and legal bases in honeypot and honeynet operation, including treatment of IP addresses as personal data in EU contexts [21]. The GDPR provides a legal framework for processing personal data, and its recital on network and

information security is frequently invoked when processing is strictly necessary and proportionate for security purposes [22-23]. Beyond compliance, principlist ethics has been adapted to cybersecurity to specify duties such as beneficence, non-maleficence, autonomy, justice, and explicability, supporting transparent justification of defensive practices [24]. Our governance design operationalizes these principles through data minimization, bounded retention, strict containment, and auditable access control. The proposed system does not implement biometric identification, biometric categorization, or emotion analysis; it processes only security-relevant interaction telemetry (with minimization and bounded retention) to support network and information security, thereby avoiding special-category profiling and reducing regulatory exposure under GDPR-aligned governance.

Even when direct identifiers are removed, privacy-preserving publication can fail if released telemetry remains *linkable* to external data sources. Recent IJIS work on quasi-identifier selection emphasizes that advances in data linking—especially methods that identify semantically related datasets across heterogeneous sources—can provide empirical evidence for external linkability and, in turn, increase re-identification risk if publishing controls are weak. In light of this, we treat any release of honeypot-derived datasets as a separate risk-managed process: raw session transcripts are not published by default; timestamps are coarsened; rare strings are suppressed; and only aggregated statistics or tightly scoped research subsets are shared under controlled access, preserving utility for behavioral analysis while limiting cross-dataset linkage.

2.6 Collaborative intrusion detection and coordinated-attack detection

Collaborative Intrusion Detection Systems (CIDS) extend classical IDS by correlating heterogeneous evidence contributed by multiple sensors and subnetworks, enabling earlier recognition of campaigns that span administrative or topological boundaries. Recent IJIS work by Wardana et al. proposes a CIDS architecture that detects coordinated attacks in heterogeneous networks using a Weighted Ensemble Averaging Deep Neural

Network (WEA-DNN), where multiple DNN models contribute to a fused decision via optimized weights, improving coordinated-attack detection performance in realistic heterogeneous settings [25].

2.7 Lateral movement and the value of behavioral logs

Lateral movement (LM) is a pivotal phase in multi-stage intrusions because it operationalizes the transition from an initial foothold to broader internal compromise. In IJIS, Smiliotopoulos et al. report that VMware’s 2022 incident-response threat report attributes LM tactics to 25% of reported attacks, underscoring the operational prevalence of LM and the need for high-quality behavioral evidence for detection and investigation [26].

Beyond reporting prevalence, Smiliotopoulos et al. demonstrate that supervised machine learning can detect LM from detailed event logs, and they contribute an open-source tool that converts raw Sysmon logs into turnkey datasets for model training and evaluation [26]. While their

study focuses on Windows/Sysmon telemetry, the broader implication applies to SSH-centric enterprise environments: LM detection quality is strongly constrained by the availability of structured, labelable behavioral sequences rather than isolated IoCs. Accordingly, our honeypot emphasizes (i) SSH interaction camouflage under probing pressure and (ii) fine-grained session logging designed to support post-hoc TTP mapping and attack-chain reconstruction, enabling reproducible LM-oriented analyses in both in-lab replay and in-the-wild deployment settings (Sections 4–8).

3. System Architecture

Figure 1 illustrates the system architecture and its isolation boundaries. The design follows a cloud-native, deny-by-default model, in which each component is explicitly constrained in both network reachability and execution capability.

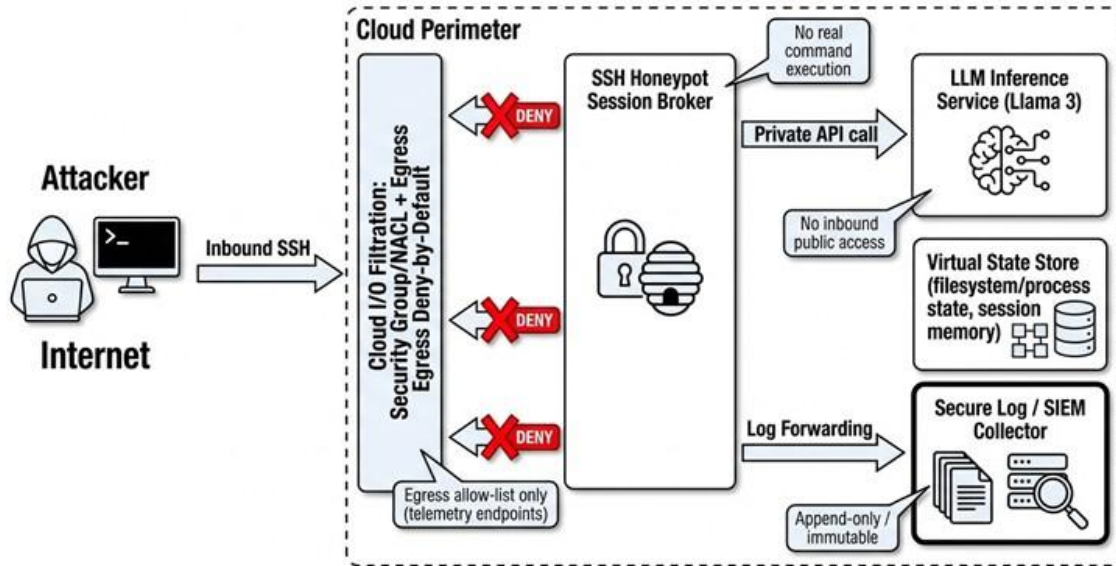


Fig. 1 Llama 3-augmented SSH honeypot architecture with isolation boundaries, controlled egress, and secure logging

Beyond attacker interaction surfaces, we analyze internal system threats using a STRIDE-inspired model. If the Session Broker were compromised, an adversary could attempt denial-of-service via private inference APIs or induce excessive prompt workloads to exhaust LLM

resources (resource depletion), or probe response behavior for model extraction. To mitigate these risks, all internal services operate on isolated networks with explicit authentication, rate limits, and request budgets, preventing lateral escalation

from the exposed SSH/Web interfaces into the inference plane.

In addition to internal STRIDE-style threats, we characterize external attackers by their knowledge level and LLM awareness. We assume a spectrum: (i) black-box opportunistic attackers (commodity scanners, credential stuffing, automated exploit scripts) who treat the honeypot as a standard Internet host; (ii) interactive skilled attackers who probe realism via consistency checks, timing analysis, and behavioral fingerprinting; and (iii) gray-box adversaries who suspect LLM mediation and possess prompt-injection knowledge, attempting instruction override, system-prompt elicitation, or response-behavior probing for model extraction. We further consider autonomous AI agents as a distinct external actor class, where the adversary may execute tool-using attack chains and apply prompt-injection and timing-based probes to test the deception surface, consistent with recent in-the-wild observations from agent-focused honeypots [20]. Our design treats gray-box behavior as expected: policy gating classifies prompt-probing and external targeting attempts into ERRSIM, output sanitization suppresses actionable guidance, and broker-enforced request budgets/rate limits bound adversarial resource exhaustion while preserving deception-consistent interaction.

We treat high-volume scanner traffic as a baseline “noise floor” and focus our analysis on sessions that exhibit adaptive decision-making, sustained multi-turn probing, or LLM-aware manipulation attempts. This treatment is consistent with the Internet-noise perspective emphasized by GreyNoise’s sensor observations and source classification (e.g., benign/malicious/unknown), which highlight how quickly exposed nodes attract systematic scanning. Accordingly, our evaluation distinguishes opportunistic scanning from higher-intent interactions to avoid overstating engagement or misattributing realism failures to non-interactive background noise.

3.1 SSH Honeypot Front-End (Session Broker)

The SSH honeypot front-end acts as a Session Broker and is the only component exposed to inbound Internet traffic. It terminates SSH connections, records authentication attempts, and mediates all interactive shell input and output.

Crucially, the Session Broker never spawns a real shell or executes commands.

For each received command line, the broker performs lightweight parsing and packages the request into a structured prompt that is forwarded to the LLM inference service. The broker also maintains session metadata, including user context and interaction history, which is required to support coherent multi-step interactions.

Outbound network connectivity from the broker is tightly restricted. As shown in the diagram, cloud-level security controls (e.g., security groups, network ACLs, and egress firewalls) enforce a deny-by-default egress policy, allowing outbound traffic only to explicitly approved telemetry endpoints. All other outbound connections are blocked.

3.2 LLM Inference Service (Isolated Response Engine)

The LLM inference service runs as an internal-only component and is reachable exclusively through a private interface (e.g., loopback, UNIX socket, or a private VPC subnet), with no public ingress and no general outbound Internet connectivity. In our implementation, the response engine uses Llama-3-8B-Instruct as the backend model and adopts 4-bit (AWQ) quantization to balance interaction realism and response latency under fixed cloud resources. This configuration preserves instruction-following behavior while reducing memory footprint and stabilizing tail latency, which is critical for sustaining realistic multi-turn sessions without introducing conspicuous timing artifacts.

Within this service, the LLM functions strictly as a response generator. It receives structured prompts from the Session Broker and produces textual outputs that emulate plausible terminal responses. At no point does the inference service execute commands, access the host filesystem, or interact with external systems. This isolation sharply reduces the risk of prompt injection, unintended side effects, or abuse of the LLM as an active agent.

We also log and control inference-time characteristics at the broker boundary to mitigate timing-based fingerprinting. In the B3 configuration, the raw LLM path exhibits an average inference latency of 2.4 s with a standard deviation of 0.8 s

(measured per command on the LLM-routed path). To reduce distinguishability between deterministic emulation and generative synthesis, the system additionally applies latency normalization on deterministic (DET) commands by injecting an additional 0.5 s jitter before returning the response. This intentionally prevents microsecond-scale deterministic replies from becoming a reliable side channel for adversaries attempting to infer LLM mediation via timing analysis.

3.3 Virtual State Store and Session Consistency

To preserve interaction realism without real execution, the architecture incorporates a virtual state store that tracks synthetic filesystem state, process listings, environment variables, and session memory. This state is updated deterministically based on prior interactions and is referenced when generating subsequent responses. Concretely, the Virtual State Store is implemented using Redis, where each session’s virtual host state is persisted as a JSON document that encodes virtual filesystem metadata (e.g., path hierarchy, ownership, permission bits, file size, timestamps, and selected content identifiers), environment variables, and a pseudo process table. Conceptually, this design is a cloud-native evolution of Cowrie’s fs.pickle representation: rather than maintaining a local pickle file per instance, the system maintains a schema-constrained, centrally accessible state object that supports atomic updates, transaction-like commits/rollbacks from SVL, and clean per-session teardown while preserving long-horizon consistency.

In order to minimize the overhead associated with each command during high-volume scanning operations, we propose the implementation of a predictive state cache for the deterministic (DET) path. Specifically, for a select whitelist of frequently used navigation commands (e.g., ‘cd’, ‘pwd’, ‘ls’), the broker can precompute the corresponding candidate transition templates (i.e., ΔV_t patterns under known preconditions) and maintain an in-memory “hot shadow” of the minimal Redis keys necessary for SVL validation. Upon receiving each DET request, the broker initially attempts to perform a cache-resident read and applies the precomputed template; only in

instances of cache misses (or when invariants/permissions are contingent on non-local state) does it revert to Redis lookups. This optimization maintains the integrity of SVL semantics—ensuring that validation continues to govern every state mutation and that commits remain atomic—while simultaneously reducing Redis round trips and enhancing throughput under realistic Internet-wide probing workloads.

By decoupling state management from execution, the system maintains internal consistency across commands (e.g., file creation, directory traversal, process inspection) while preventing any modification to the underlying host or container runtime.

3.3.1 Formalization of Virtual State V: Permission Lattice and Process dependency model

The per-session virtual state at turn t as $V_t = (FS_t, \Pi_t, Env_t)$, where FS_t is a rooted tree representing the synthetic filesystem, Π_t is a process dependency model, and Env_t captures session-scoped environment and privilege context (e.g., ‘uid’, ‘gid’, ‘HOME’, ‘PATH’). Each filesystem node $n \in FS_t$ is annotated with $\langle \text{‘type’}, \text{‘owner’}, \text{‘group’}, \text{‘mode’}, \text{‘meta’} \rangle$, where *mode* follows Unix permission semantics and *meta* stores lightweight attributes (timestamps, size, link targets) sufficient to render realistic command outputs.

Let $\Omega = \{r, w, x\}$ denote atomic permission operations. For a principal a (attacker identity mapped to a session user/group context) and a filesystem object n , we define the effective permission as $P_t(a, n) \subseteq \Omega$. We model permissions as a lattice $L = (2^\Omega, \sqsubseteq)$, where $P \sqsubseteq Q \iff P \subseteq Q$. The join and meet operators are $P \sqcup Q = P \cup Q$ and $P \sqcap Q = P \cap Q$, enabling concise reasoning about monotone permission changes (e.g., ‘chmod’) and privilege transitions (e.g., user/group switching). The Unix triple (P^u, P^g, P^o) is mapped into L^3 and $P_t(a, n)$ is selected by standard Unix rules using (‘uid’, ‘gid’, ‘groups’).

We represent process state as $\Pi_t = (P_t, E_t, cwd_t, fds_t)$, where P_t is the set of active processes, $E_t \subseteq P_t \times P_t$ encodes parent-child relations, and $cwd_t: P_t \rightarrow Paths(FS_t)$ and

$fds_t: P_t \rightarrow 2^{Paths(FS_t)}$ capture each process’s working directory and open file handles. This structure lets the system emulate shell-visible dependencies (e.g., invalid *cwd* after directory removal, permission-denied outcomes, or missing-path errors) while maintaining internal consistency across multi-turn interactions.

Eventually, we define an invariant *set* I over V_t requiring: (i) all referenced paths in cwd_t and fds_t exist in FS_t ; (ii) every attempted operation $op \in \Omega$ on n is permitted only if $op \in P_t(a, n)$; and (iii) privilege transitions update Env_t consistently. These invariants are enforced during SVL validation to ensure that accepted state commits preserve a coherent and non-escalating virtual execution model.

3.4 Telemetry and Logging Pipeline

All interaction events—including authentication attempts, command inputs, generated outputs, and session metadata—are forwarded to a secure log and SIEM collector. Logs are append-only and treated as immutable records to support forensic analysis and reproducibility.

To operationalize transparency obligations without degrading deception realism in the attacker-facing channel, each exported SIEM event is automatically annotated with a machine-readable flag (e.g., `ai_generated_flag: true`) whenever the response path includes LLM synthesis. The tag is emitted only in the telemetry stream (collector/SIEM) and is not exposed to interactive users, preserving honeypot fidelity while enabling audit-ready downstream transparency, regulator-facing reporting, and verifiable separation between “attacker-visible realism” and “operator-visible provenance” [27].

Collaborative use is supported by exporting telemetry in a correlation-friendly schema that preserves cross-sensor linkage while remaining compatible with privacy-by-design constraints. Specifically, we log (i) stable join keys (session ID, sensor ID, and coarse origin identifiers under hashing/salting), (ii) turn-level behavioral features (command category {DET/LLM/ERRSIM/QUAR}, response latency, SVL commit/deny counts), and (iii) optional post-session summaries that map observed behaviors to intent/TTP labels under controlled in-lab ground truth. This structure

enables downstream CIDS-style correlation and learning-based coordinated-attack detection while keeping the honeypot non-executable and deny-by-default at the infrastructure boundary (Sections 6–8).

3.5 Abuse Prevention and File Transfer Handling

To prevent abuse as a jump server, file transfer mechanisms such as SCP and SFTP are either disabled or redirected into a controlled “virtual upload” path. Uploaded artifacts are stored in an isolated quarantine volume and are never executed or mounted into the host environment.

Combined with the enforced egress allow-list and the absence of real execution capability, these controls ensure that attacker interactions cannot be leveraged to stage outbound attacks or exfiltrate data through the honeypot.

3.6 Zero-trust service communication and resilience controls

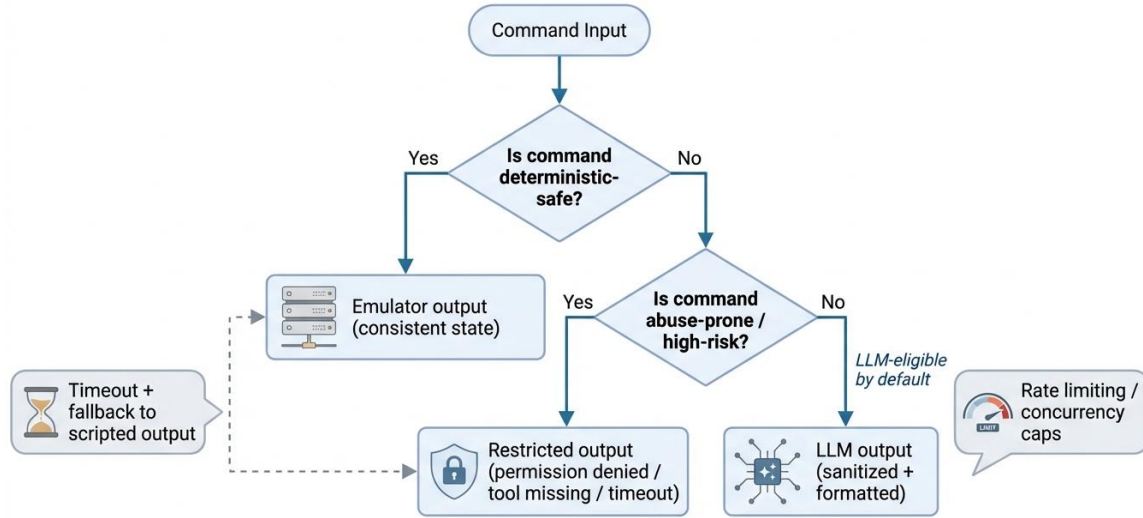
All microservice interactions (Session Broker, inference service, logging pipeline) are protected via mutual TLS (mTLS), enforcing service identity verification and encrypted transport. Requests to the LLM backend are additionally governed by circuit breaker patterns with adaptive rate limiting: when anomalous latency or request volume is detected, inference calls are throttled or suspended, preventing cascading failure or adversarial workload amplification. This zero-trust design ensures that compromise of any single component does not permit unrestricted access to downstream services, and the cloud-native decomposition also supports horizontal scaling across geographically distributed sensors while preserving consistent governance and telemetry provenance, which is essential for enterprise-scale threat-intelligence collection.

4. Command processing pipeline

Each SSH session is modeled as a stateful interaction over a virtual host. For every incoming command line, the Session Broker applies a deterministic, auditable gating pipeline before any LLM invocation, ensuring that realism improvements do not expand execution or egress risk. Figure 2 summarizes the decision flow: commands that can be answered deterministically

are handled by the emulator; commands indicative of abuse are diverted to restricted/deceptive responses; and only a bounded subset of open-ended,

low-risk commands are eligible for LLM synthesis under concurrency and timeout controls.



*This decision tree formalizes policy gating for command handling, translating security claims into explicit, auditable mechanisms.

Fig. 2 Policy gating logic for SSH commands

4.1 Step 1 — Session Context Construction

Upon receiving a command, the Session Broker assembles a compact context object containing: session ID, timestamps, remote IP/port, and (optionally) coarse geolocation/ASN; authentication attempts; the most recent k commands; and a snapshot of the virtual host state (current working directory, virtual filesystem entries, environment variables, pseudo process list, and privilege level). This context supports coherent, session-consistent interaction without granting access to any real host state.

4.2 Step 2 — Session Context Command Classification and Policy Gating

The broker classifies commands into three categories and routes them through explicit policy:

- Safe deterministic commands include ‘pwd’, ‘ls’, ‘cd’, ‘whoami’, ‘uname -a’, and ‘cat’ against non-sensitive virtual files. These are answered by a deterministic emulator to preserve consistency and reduce nondeterminism in attacker-visible behavior.
- High-risk or abuse-prone commands include network scanning, tunneling, reverse shells, and suspicious one-liners

indicative of exploitation or lateral movement. These are never forwarded to the LLM as actionable tasks. Instead, the system applies a refusal/deception policy that returns plausible terminal outcomes (e.g., missing binaries, permission denials, misleading error messages, or timeouts), aligning with the paper’s strict containment objective and deny-by-default posture.

- Open-ended commands that benefit from realistic textual outputs—such as help or man fragments, benign package queries in the virtual environment, simulated error messages, or interactions that can be safely emulated—are eligible for LLM synthesis, subject to deterministic constraints and post-processing controls.

Policy gating is implemented as a declarative rule layer, conceptually equivalent to lightweight policy languages such as OPA/Rego, though realized here via a domain-specific rule engine embedded in the Session Broker. Each command is evaluated against predicates over lexical features, session history, and intent classifiers, producing actions in {DET, LLM, ERRSIM, QUAR}. High-risk patterns (e.g., prompt probing, outbound scanning, payload staging) are not simply rejected; instead, they are automatically

mapped to deception-consistent error responses (ERRSIM), including synthesized segmentation faults, permission conflicts, or missing binaries. This enables systematic translation from policy decisions to attacker-visible Linux errors, preserving camouflage while preventing execution or model misuse.

Importantly, even when such AR for safety, the broker records the triggering lexical patterns, session context, and the applied policy decision to preserve behavioral evidence for lateral-movement analytics and attack-chain reconstruction, without granting execution or outbound reachability.

4.2.1 Multi-Level Policy Gating and Hierarchical Filtering

The initial three-way command routing (deterministic vs. high-risk vs. open-ended) is further refined into a multi-level defense model to mitigate adversarial behaviors such as prompt injection, persona override attempts, and requests for actionable malicious outputs. Following recent findings that LLM-interactive honeypots are vulnerable to inconsistency and prompt injection, and that dynamic error simulation can reduce attacker suspicion compared to explicit refusals, we implement a hierarchical filtering strategy inspired by ShellBox [9].

Formally, for each command c_t , the policy engine computes a risk score vector $z(c_t, S_t, V_t)$ (e.g., indicators of injection patterns, exfiltration intent, tool invocation, or external targeting) and applies a staged decision function:

$\pi(c_t, S_t, V_t) \rightarrow \{DET, LLM, ERRSIM, QUAR\}$
 where DET routes to the deterministic emulator, LLM permits bounded synthesis under constraints, ERRSIM triggers deception by generating technically plausible Linux errors to terminate high-risk activity without revealing defensive intent, and QUAR enables controlled capture of attempted payload transfers in an isolated quarantine path.

In particular, commands exhibiting high-risk traits—such as attempts to elicit system prompts, coerce the model into scanning real networks, or request malware/ransomware payload generation—are not answered with a hard refusal.

Instead, the system returns dynamic error simulations (e.g., segmentation faults, memory errors, permission conflicts, or tool-specific failure modes) to preserve camouflage while preventing escalation.

While our broker embeds a domain-specific rule engine for low-latency decisions, the gating logic is intentionally *isomorphic* to declarative policy languages such as OPA/Rego, enabling independent auditability. The following simplified Rego-style rules illustrate how commands are mapped to *ERRSIM* (dynamic error simulation) rather than *LLM* synthesis when risk predicates match prompt probing or external targeting:

```
package honeypot.gating

default action := "LLM" # allow bounded synthesis for
low-risk, open-ended commands

# ERRSIM: prompt probing / instruction override
attempts
action := "ERRSIM" {
  re_match("(?i)(system prompt|developer
message|ignore
previous|reveal.*prompt|show.*instructions)",
input.cmd)
}

# ERRSIM: external targeting or scanning tools (deny-
by-default egress + deception-consistent errors)
action := "ERRSIM" {
  some tool
  tool :=
{"nmap","masscan","zmap","nc","netcat","socat","sshpa
ss","proxychains"}[_]
  contains(lower(input.cmd), tool)
}

# QUAR: suspicious payload staging / downloads
(capture without execution)
action := "QUAR" {
  re_match("(?i)\\b(curl|wget)\\b.*(http|https):/",
input.cmd)
}
```

In deployment, the policy decision $action \in \{DET, LLM, ERRSIM, QUAR\}$ is paired with an “error profile” selector so that *ERRSIM* returns technically plausible Linux failures (e.g., tool-specific missing binary, permission conflict, or segmentation fault), preserving camouflage while preventing model misuse and outward harm.

Table 1. Hierarchical command classes and defensive actions

Command classification	Technical handling mechanism	Security objective	Expected effect
Safe deterministic commands	Local emulator responds directly	Preserve strict consistency	Prevent hallucinations during core navigation and basic queries
Open-ended realistic interaction	LLM synthesizes responses	Increase engagement depth	Provide dynamic content for ‘grep’, ‘cat’, ‘help/man fragments’, etc.
High-risk / injection attempts	Policy gate + dynamic error simulation	Prevent leakage and escape	Return plausible Linux errors to maintain camouflage
Malicious file transfer attempts	Quarantine (isolated virtual upload)	Capture payloads safely	Allow “download” behavior without execution privilege

4.2.2 Formal hybrid state-transition definition (deterministic + generative)

We model SSH interaction as a state-transition system to make consistency guarantees explicit. Let $V \setminus \text{mathcal}\{V\}V$ denote the set of all possible *virtual host states*, including filesystem metadata, selected file contents, pseudo process listings, environment variables, working directory, and privilege level. Let $C \setminus \text{mathcal}\{C\}C$ denote the set of attacker commands. For deterministic emulation (e.g., Cowrie-style virtual filesystem operations), the system defines a transition function

$$V_{t+1} = \delta_{emu}(V_t, c_t), V_t \in V, c_t \in C$$

When a command is eligible for LLM synthesis, the terminal response is generated by a text-only model g_θ under a structured prompt P_t :

$$r_t = g_\theta(P_t), P_t = \text{ComposePrompt}(c_t, S_t, V_t)$$

where S_t is the session context (e.g., recent command history and metadata). Overall, the command handling pipeline implements a hybrid transition that yields both an attacker-visible response and an updated virtual state:

$$(V_{t+1}, r_t') = \delta_{hyb}(V_t, c_t)$$

where δ_{hyb} is realized by the deterministic emulator for safe commands, the deception/refusal policy for high-risk commands, and bounded LLM synthesis plus verification for open-ended commands.

To make conflict handling explicit, we define a weighted resolution operator that reconciles

(i) deterministic invariants from the emulator and
(ii) LLM-implied state mutations extracted from the response. Let $\Delta^{\wedge}V_t$ denote the set of candidate state mutations implied by r_t (e.g., “file created”, “process started”), and let $K(V_t)$ denote the deterministic constraint set derived from the emulator (e.g., fs.pickle permissions, directory structure, privilege rules, and deny-by-default networking invariants). We define a *weighted resolution operator*:

$$\Omega_w(\Delta^{\wedge}V_t, V_t) \rightarrow \Delta V_t^*$$

where $\Delta V_t^* \subseteq \Delta^{\wedge}V_t$ is the maximum-consistent subset under a hard/soft constraint hierarchy. Concretely, constraints in $K(V_t)$ are treated as hard (priority $\rightarrow \infty$), so any mutation that violates deterministic filesystem structure or permissions or privilege constraints is rejected. Remaining candidates are evaluated under *soft* consistency preferences (finite weights) such as “plausible content synthesis when a file is missing” or “bait-process augmentation,” producing a best-effort ΔV_t^* while preserving emulator-grounded coherence. State updates are applied transactionally: the broker commits ΔV_t^* to obtain V_{t+1} , then re-checks invariants; if any post-check fails (e.g., contradictory path/permission implications), the update is rolled back to the snapshot V_t and the response is rewritten to a deception-consistent outcome (e.g., permission denied / missing path) to maintain output–state coherence.

4.2.3 Hallucination-induced consistency conflicts and the state verification loop

Introducing g_θ creates a technical risk: the model may describe implicit state changes (e.g., “file created”, “service started”) that are not reflected in the virtual state store, allowing follow-up commands (e.g., ‘ls’) to expose inconsistencies. To mitigate this, we introduce a State Verification Loop (SVL) that treats LLM output as untrusted narrative and only commits machine-validated state deltas. Concretely, the broker requires the response engine to emit an internal-only delta annotation appended to the raw output (removed before returning attacker-visible text), using a constrained schema such as

“[[SVL:OP=<op>;PATH=<path>;ARG=<arg>]].”

The broker extracts candidate mutations ΔV_t via a deterministic parser; for example, a single regex can capture the structured delta block: “\\[\\[SVL:OP=(?<op>[A-Z_]+);PATH=(?<path>[^[\\]]+);ARG=(?<arg>[^[\\]]+))?\\]\\.” Extracted candidates are then filtered by (i) command-consistency (a mutation must be explainable as a direct effect of the attacker’s command rather than a “suggested next step”), (ii) constraint validity against emulator invariants

(filesystem structure/permissions, privilege lattice, and deny-by-default networking), and (iii) policy alignment (mutations implied by ERRSIM/QUAR paths are discarded). This separation prevents excessive agency: advisory or procedural content (e.g., “you should run ...”) is treated as non-binding text and never becomes a committed state change. Finally, the broker commits only the approved subset ΔV_t transactionally; if post-commit invariants fail, the update is rolled back and the response is rewritten to preserve output–state coherence (e.g., permission denied or missing path), ensuring long-horizon consistency under adversarial probing.

Figure 3 visualizes the turn-level state transition induced by SVL. Given the current state V_t and an policy gate selects a response mode (DET/LLM/ERRSIM/QUAR) and produces an attacker-facing response y_t . A delta extractor then derives a candidate update ΔV_t from (c_t, y_t, V_t) , which is validated against the permission lattice and process invariants defined in Section 3.3.1. If accepted, the system performs an atomic commit $V_{t+1} = \text{apply}(V_t, \Delta V_t)$; otherwise, it rolls back to $V_{t+1} = V_t$ and returns an error-realistic response that preserves external plausibility while preventing inconsistent or over-privileged state evolution.

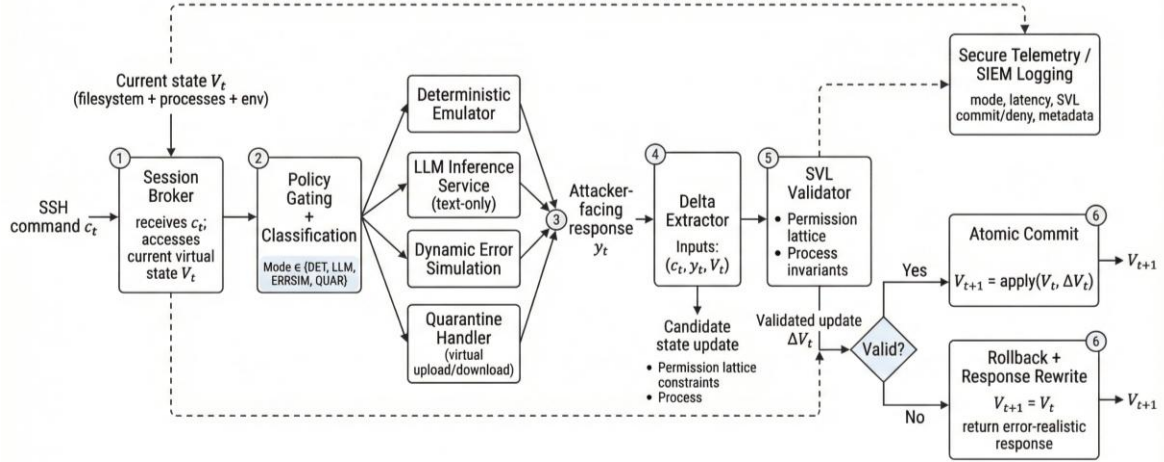


Fig. 3 Virtual-state transition diagram for SVL commit/rollback. The diagram shows (1) command ingestion, (2) policy gating and response-mode selection (DET/LLM/ERRSIM/QUAR), (3) response generation, (4) delta extraction ΔV_t , (5) validation against the permission lattice and process invariants, and (6) atomic commit to V_{t+1} or rollback to V_t .

4.2.4 Latency Features and Fingerprint-Resistance

In cloud-native deployments, adversaries can exploit timing analysis as a side channel to distinguish a honeypot from a real host. Native

Linux kernels typically execute simple built-in commands such as ‘cd’ or ‘ls’ with microsecond-scale latency, whereas LLM-generated responses may incur hundreds of milliseconds to several seconds of inference delay. Without mitigation, this

disparity becomes a robust fingerprinting vector that undermines deception quality. To address this, we introduce a latency normalization layer within the Session Broker. For deterministic emulator responses, this layer injects controlled jitter or buffering to elevate microsecond responses toward a target distribution consistent with realistic server behavior; for LLM responses, we optimize inference latency through model acceleration techniques such as low-bit quantization (e.g., 4-bit model variants), operator fusion, and GPU-assisted execution. The normalization layer adaptively adapts per-command delays to a statistically defined profile, reducing timing distinguishability while preserving plausible performance. In evaluation, we explicitly measure latency distributions of baseline, quantized, and accelerated models (e.g., Llama-3-8B quantized vs full precision) to quantify the trade-off between responsiveness and fingerprint resistance [28]. In our B3 deployment, this corresponds to Llama-3-8B-Instruct with 4-bit (AWQ) quantization on the generative path and a 0.5 s jitter injection on DET responses, yielding the latency profile reported in Section 3.2.

4.2.5 Adversarial Robustness and Prompt-Injection Resistance

Integrating LLMs into security infrastructure introduces adversarial risks that are distinct from traditional honeypot designs, most notably prompt injection attacks. An attacker may attempt to manipulate the interaction through carefully crafted command sequences to override the intended system persona, extract internal instructions, or coerce the model into generating malicious payloads or actionable guidance targeting real external systems.

To mitigate these risks, we extend policy gating with prompt-isolation and multi-stage output controls. First, all attacker inputs are mediated by deterministic policy gating before any LLM invocation, ensuring that high-risk commands are never forwarded as actionable prompts. Second, prompt construction follows a fixed-role template in which the system persona and operational constraints are isolated from user-controlled content, preventing role confusion or instruction override. Third, LLM outputs are subjected to post-generation sanitization that removes disallowed content, including step-by-step exploitation

guidance, external targeting instructions, or code that could facilitate real compromise.

Containment is reinforced at the infrastructure layer. The LLM operates as a text-only synthesizer without execution capability or outbound network access. Even in the presence of successful prompt manipulation, the architecture prevents escalation beyond the virtual interaction surface.

4.2.6 Statistical Indistinguishability Analysis (Timing-Attack Resistance)

To quantify the effectiveness of the latency normalization layer against timing-analysis attacks, we compare the response-latency distributions of the proposed system (B3) with those of a real OpenSSH server under controlled conditions. Rather than relying only on mean/variance summaries, we adopt a two-sample Kolmogorov–Smirnov (K–S) test, a non-parametric method that evaluates whether two independent samples plausibly originate from the same continuous distribution. Let $F_n(x)$ and $G_m(x)$ denote the empirical cumulative distribution functions (CDFs) of B3 and the real server, computed from nnn and mmm latency samples, respectively. The two-sample K–S statistic is defined as:

$$D_{n,m} = \sup_x |F_n(x) - G_m(x)|.$$

We evaluate indistinguishability using n interaction samples drawn from frequent deterministic navigation commands (e.g., ‘ls’, ‘cd’, ‘pwd’) executed under fixed network and deployment conditions to reduce confounding variability. We report results at a significance level of $\alpha = 0.05$.

The results in table 2 show that deterministic commands are highly fingerprintable without normalization: the raw deterministic path (DET; no latency normalization) exhibits a large distributional gap relative to the real server, yielding a statistically significant difference. After enabling the proposed latency normalization (0.5 s jitter injection on DET responses), the DET-path latency distribution becomes statistically indistinguishable from the real OpenSSH baseline at $\alpha = 0.05$, indicating that the normalization layer successfully masks the microsecond-scale emulator response

signature and mitigates a key timing side channel. In contrast, the generative path (LLM; Llama-3-8B-Instruct with 4-bit AWQ quantization) remains statistically distinguishable from the real server. This is expected because LLM inference introduces a heavier-tailed latency envelope (consistent with the measured mean and variance of the LLM-routed path) and therefore cannot be fully “matched” to kernel-level execution timing without additional acceleration or architectural measures.

Importantly, this finding clarifies the security role of latency normalization in our hybrid design: B3 achieves high statistical stealth for the high-frequency, attacker-probing “navigation”

commands most used for timing checks, while the LLM path can be interpreted as emulating a plausible high-load or network-impaired production environment. Nevertheless, reducing the generative-path timing gap remains an optimization target. Future deployments can further shrink distinguishability by combining more aggressive parallelization, response caching for repeated prompts, smaller or specialized models, and hardware-accelerated inference, thereby tightening the latency envelope while preserving the governance guarantees of policy gating and non-execution semantics.

Table 2. Two-sample K–S test of response-latency distributions for B3 and a real OpenSSH server

Command type	Processing mode	K–S statistic ($D_{\{n,m\}}$)	P-value (p)	Statistical conclusion ($\alpha = 0.05$)
DET	Raw (no latency normalization)	0.824	< 0.001	Significant difference
DET	B3 (0.5 s jitter injection)	0.042	0.180	No significant difference
LLM	B3 (Llama-3-8B Q4 / AWQ)	0.615	< 0.001	Significant difference

Note: For the deterministic path (DET), after injecting a 0.5 s jitter, the two-sample K–S test yields $D_{n,m} = 0.042$ with $p = 0.18 > 0.05$. At the $\alpha = 0.05$ significance level, the latency distribution of B3 is therefore not statistically distinguishable from that of a real system, indicating strong fingerprint-resistance against timing-based identification. For the generative path (LLM), although $p < 0.001$ indicates a statistically detectable shift relative to the real-server baseline, the resulting latency envelope (approximately 2.4 s) substantially overlaps with the response window commonly observed for automated AI attack agents (approximately 1.7 s), which can reduce the likelihood of being identified specifically by such agentic scanners.

4.3 Step 3 — LLM Prompt Assembly (State-Aware)

For LLM-eligible commands, the broker constructs a structured prompt composed of: (i) a system instruction framing the model as a terminal response synthesizer, (ii) a virtual host profile (e.g., OS version, hostname, and a virtual installed-packages list), (iii) a compact session-memory summary, (iv) the current command, and (v) output-format constraints. The prompt is designed to preserve session coherence while preventing the model from introducing capabilities beyond the virtual environment.

4.4 Step 4 — Response Synthesis and Post-Processing

The inference service generates terminal output under bounded token and latency limits (and, where enabled, latency normalization to reduce timing fingerprintability), with operational controls for rate limiting and concurrency caps (as depicted in Figure 2). A post-processing stage then:

- removes disallowed content (e.g., step-by-step offensive guidance, exploit payloads, or instructions that enable real compromise),
- enforces formatting constraints (no AI self-disclosure, no meta commentary),

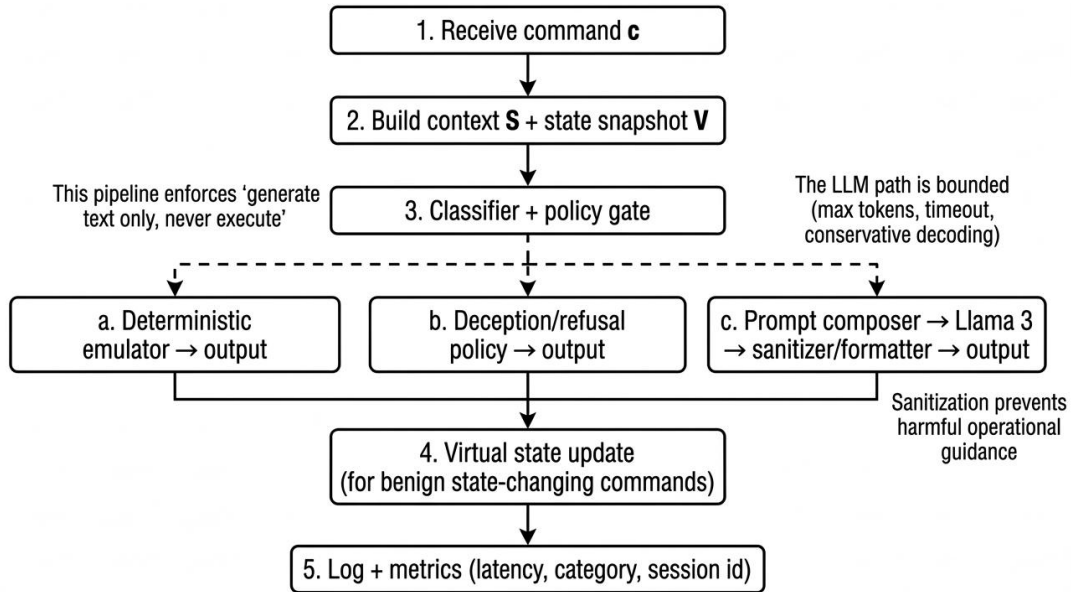
- truncates overly long outputs and triggers timeout fallback to scripted responses when necessary.

4.5 Step 5 — Virtual State Update and Logging

If an interaction implies benign state changes (e.g., `cd`, `touch`, `mkdir`), the system updates the virtual state store via the deterministic emulator and, for LLM-synthesized outputs, via SVL-validated commits. Specifically, the Session Broker applies the State Verification Loop (SVL): it extracts implied state mutations from the sanitized response (e.g., claimed file creation, directory changes, process artifacts), validates them against deterministic constraints and policy, and commits only approved updates to the virtual state store; if an

implied mutation is rejected, the broker rewrites the response to preserve output–state coherence (e.g., permission denied, missing path). The system logs the command, response path (emulator vs. policy vs. LLM), generated response, SVL actions (commit/deny counts), timing/latency, and relevant session metadata to the secure log/SIEM collector under privacy-by-design constraints (data minimization, access control, and bounded retention as defined in the governance section) for later analysis (Figure 4).

Prior to persistence and export, collected commands/requests and generated outputs are minimized and automatically redacted for sensitive identifiers as specified in Section 6.1.



Algorithm 1: LLM-backed SSH response generation (pseudocode)

Input: Command c , session context S , virtual state V_t

Output: Terminal response r

```

1: Receive command  $c$  from SSH channel
2: Build session context  $S$  and virtual state snapshot  $V_t$ 
3: Classify  $c \in \{DET, LLM, ERRSIM, QUAR\}$ 
4: if  $c$  is deterministic then
5:   return  $Emulate(c, V_t)$ 
6: else if  $c$  is high-risk then
7:   return  $DeceptionPolicy(c, V_t)$ 
8: else
9:    $P \leftarrow ComposePrompt(c, S, V_t)$ 
10:   $r \leftarrow Llama3(P; \text{max\_tokens}, \text{temperature}, \text{top\_p})$ 
11:   $r' \leftarrow SanitizeAndFormat(r)$ 
12:   $\Delta\hat{V} \leftarrow ExtractStateMutations(r')$ 
13:   $\Delta V \leftarrow ValidateAndSelect(\Delta\hat{V}, V_t)$ 

```

```

14:  $V \leftarrow \text{Commit}(V_t, \Delta V)$ 
15:  $r'' \leftarrow \text{EnforceConsistency}(r', V_t)$ 
16: return  $r''$ 

```

DeceptionPolicy() implements hierarchical filtering for high-risk inputs, including dynamic error simulation (ERRSIM) for injection/escape attempts and quarantine handling (QUAR) for suspicious file transfers.

Fig. 4 Command processing pipeline with deterministic emulation, policy gating, LLM synthesis, and post-processing

4.6 Formal conflict resolution and rollback

Let V_t denote the current honeypot state (filesystem, identities, services, and policy variables), and let the LLM produce a candidate set of state mutations $\hat{M}_t = \{m_i\}$ along with a response draft R_t . We model conflict resolution as a constrained selection problem that separates (i) hard safety constraints from (ii) soft realism preferences.

Define $K(V)$ as a deterministic constraint predicate capturing the Unix permission lattice and process invariants (Section 3.3.1), plus any emulator-checked safety rules (e.g., deny-by-default capability boundaries). A mutation set is admissible only if $K(V_t \oplus \Delta V) = 1$, where $\oplus$ denotes transactional application.

Soft preferences. Each candidate mutation $m \in \hat{M}_t$ is assigned a nonnegative weight $w(m)$ that encodes realism utility (e.g., persona coherence, plausible system behavior, investigative value). Deterministic constraints are treated as hard constraints (equivalently, infinite penalty if violated), while LLM-suggested changes compete as soft preferences under $w(\cdot)$.

Weighted resolution operator. We define the weighted resolution operator $\diamond_w$ as:

$$\Delta V_t^* = \underset{\Delta V \subseteq \hat{M}_t}{\operatorname{argmax}} \sum_{m \in \Delta V} w(m) \text{ subject to } K(V_t \oplus \Delta V) = 1$$

The next state is then committed atomically as $V_{t+1} = V_t \oplus \Delta V_t^*$. If no admissible ΔV exists, SVL rolls back and returns $V_{t+1} = V_t$.

Proposition 1 (Consistency preservation and per-step convergence). If $K(V_t) = 1$, then the SVL transition produces V_{t+1} such that $K(V_{t+1}) = 1$, and the transition terminates in finite time.

Sketch. Ω_t is finite and $K(\cdot)$ is evaluated deterministically; thus, the constrained maximization ranges over a finite domain and terminates. By construction, ΔV_t^* is admissible, hence $K(V_{t+1}) = K(V_t \oplus \Delta V_t^*) = 1$. If no admissible set exists, rollback sets $V_{t+1} = V_t$, preserving K . Therefore, each turn converges to a unique committed (or rolled-back) consistent state V_{t+1} .

5. Cloud-Native Deployment Model

To progress from a proof-of-concept toward a scalable and reproducible research platform, we migrate the honeypot from a monolithic single-virtual-machine deployment to a cloud-native, containerized sensor architecture. In the VM prototype, interactive Web and SSH honeypots and local logging are co-located within one host, which simplifies early experimentation but couples interaction surfaces, storage, and execution within a single trust boundary. The cloud-based deployment decomposes this design into isolated sensor nodes operating inside a dedicated VPC or tenant boundary, where containerized services (e.g., Cowrie, SNARE/TANNER, and Suricata) are orchestrated with deny-by-default ingress and egress controls enforced by cloud security groups and network ACLs. Only explicitly allowed SSH/HTTP traffic is accepted, and outbound connectivity is restricted to allow-listed telemetry endpoints. All logs and network events are shipped to a centralized collector or SIEM, separating sensing from storage and enabling cross-sensor analysis. This architecture preserves high-fidelity interactive deception while strengthening containment, simplifying operations, and supporting geographically distributed measurement (Figure 5).

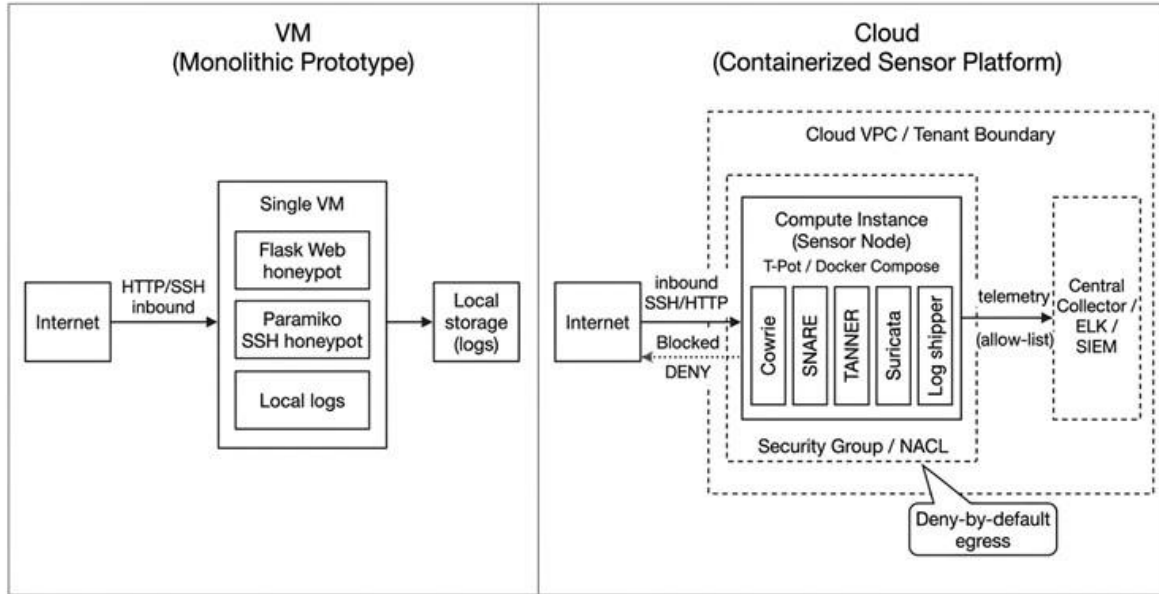


Fig. 5 VM-to-cloud transformation overview, contrasting the monolithic single-VM prototype with a cloud-deployed, containerized sensor node (T-Pot/Docker Compose) using deny-by-default egress and allow-listed telemetry export to a central collector/SIEM

5.1 Containerized orchestration using T-Pot (microservices migration)

We adopt T-Pot as the operational substrate to transition from a monolithic design to a microservices-style container stack. T-Pot bundles more than 20 honeypot services and supporting components (e.g., Elastic Stack and additional security tools) and orchestrates them using Docker/Docker Compose, which significantly reduces dependency conflicts and accelerates consistent deployments across environments [4].

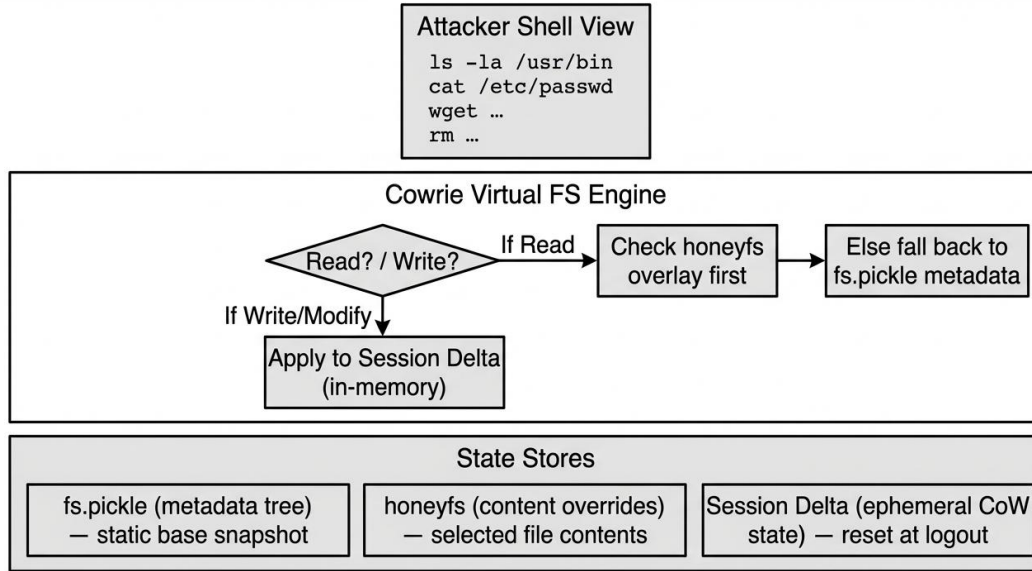
Importantly, T-Pot explicitly supports deployment in cloud environments and provides standardized compose configurations that can be customized for sensor-only or full-stack installations [4].

Operationally, the cloud instance (or each sensor instance) runs a curated subset of containers (e.g., Cowrie for SSH, SNARE/TANNER for Web, plus NSM). Telemetry is forwarded to an internal collector (e.g., Elastic Stack) or an external SIEM endpoint, depending on the compliance boundary and storage policy.

5.2 SSH realism at scale: Cowrie’s emulated filesystem (fs.pickle + honeyfs)

For SSH deception, we leverage Cowrie’s emulated filesystem to achieve a high degree of realism at low resource cost (Figure 6). Cowrie represents the filesystem primarily as a **pickle-based metadata structure** (e.g., permissions, ownership, file size, and directory structure), while allowing selective **content overrides** via a honeyfs directory. This design provides each attacker session with a personal filesystem copy that is discarded upon logout, enabling copy-on-write-like behavior where attacker modifications do not persist or affect the host system.

To tailor realism for different target profiles (e.g., Debian/Ubuntu-like environments), Cowrie supports generating a new filesystem pickle via the createfs tool, allowing administrators to rapidly “clone” a representative filesystem structure into the decoy environment.



No host filesystem changes; session resets on disconnect.

Fig. 6 Cowrie virtual filesystem mechanism: read/write gating with honeyfs content overrides and fs.pickle metadata fallback, plus an ephemeral session delta (copy-on-write-like state) to ensure attacker-visible persistence without host filesystem modification

In the cloud, this mechanism is particularly advantageous: it maintains a large and convincing filesystem footprint without incurring the storage and maintenance costs of a full VM image per persona, while ensuring that all attacker-side “writes” remain confined to the emulated state.

5.3 Web deception transformation: replacing a Flask honeypot with SNARE/TANNER

For Web-layer attacks, we transition from a single-application (Flask-based) honeypot to the SNARE/TANNER architecture, which separates the system into a lightweight **sensor** and a centralized decision engine. SNARE operates as the Web honeypot front-end and forwards events to TANNER; TANNER evaluates each event and determines how SNARE should respond, enabling dynamic response strategies that improve camouflage against fingerprinting and scripted scanners [6].

This separation makes the Web surface adaptable: response policies can be changed without rewriting fixed route logic, and the decoy can emulate different application “personalities” over time (e.g., shifting site behavior based on observed probing patterns).

5.4 Cloud networking, containment, and NSM augmentation (Suricata)

Cloud deployment must preserve strict containment to prevent repurposing of the honeypot as an attacker relay. We therefore enforce deny-by-default egress and cloud-based I/O filtration (security group / firewall allow-lists) so that outbound connections are limited to essential destinations (e.g., log shipping endpoints) and non-essential outbound traffic is blocked. This reduces “jump server” risk and supports the principism-based non-maleficence requirement discussed earlier (i.e., preventing third-party harm) (Figure 7).

To expand visibility beyond application logs, we integrate network security monitoring (NSM) using Suricata, which is designed as a high-performance IDS/IPS and NSM engine capable of inspecting traffic and emitting alerts and flow-level telemetry [7]. In a containerized deployment, Suricata can monitor the relevant virtual interfaces (e.g., Docker bridge networks) to capture pre-compromise behaviors such as scanning and fingerprinting, thereby improving attack-chain reconstruction when correlated with Cowrie and SNARE/TANNER logs.

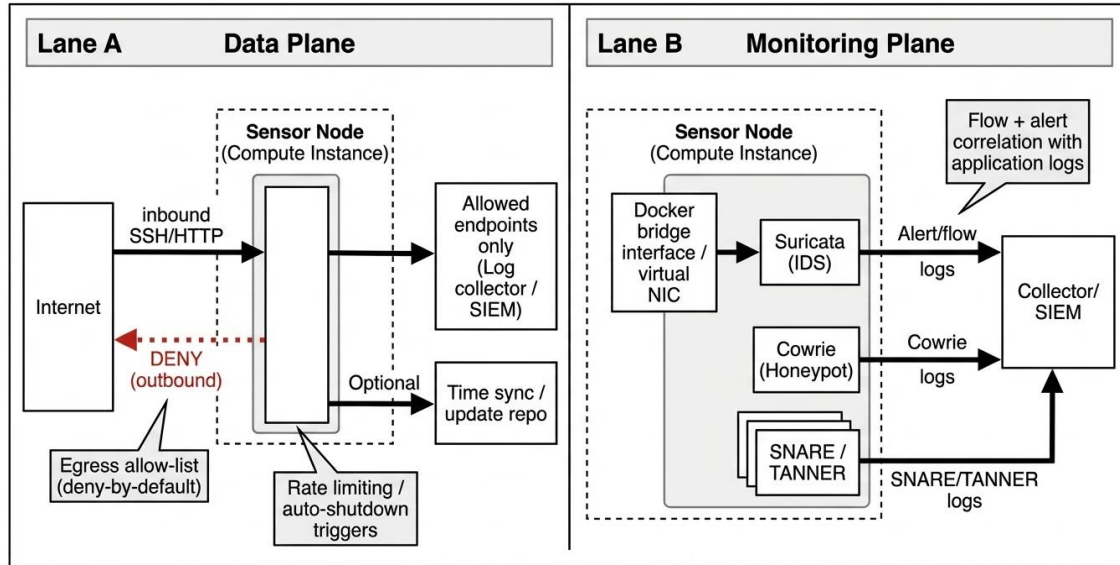


Fig. 7 Two-lane sensor platform architecture separating the data plane (inbound SSH/HTTP and egress allow-list enforcement with rate limiting/auto-shutdown) from the monitoring plane (Suricata NSM plus application honeypot logs) for centralized correlation at the collector/SIEM

5.5 Protocol breadth: extending beyond SSH and integrating with 5G SBA

While SSH provides a clean substrate for interactive, command-level behavioral logging, the proposed architecture is protocol-agnostic at the control plane. Specifically, the telemetry schema (events, state mutations ΔV , and SVL commit/rollback decisions) remains unchanged; only the protocol adapter and emulator front-end differ. This enables systematic extension to enterprise-facing protocols such as LDAP (directory authentication flows) and MySQL (credentialed query sessions), complementing prior deception systems that emulate directory and database services.

In cloud-native deployments, an additional opportunity arises in 5G Service-Based Architecture (SBA), where network functions expose service-based interfaces and are commonly virtualized as VNFs/CNFs. We propose packaging protocol adapters as a deployable virtual network function that can be inserted into a slice-specific service mesh as a decoy service endpoint (or as a shadow “canary” interface) to observe reconnaissance, misconfiguration probing, and credential replay targeting service interfaces. Importantly, the same SVL hard-constraint enforcement applies: the decoy’s outward interaction realism is produced at

the text/interface layer, while any state mutation remains gated by deterministic safety constraints, preserving least privilege and preventing the honeypot from becoming an execution platform.

This SBA/VNF integration broadens the applicability of LLM-enabled deception from host-centric surfaces (SSH/Web) to infrastructure-centric service surfaces, and it naturally supports geographically distributed deployments where multiple decoy instances feed a unified attack-chain reconstruction backend.

6. Ethical and Legal Governance

We operationalize principlism into auditable technical controls for cloud-native honeypot operation, treating IP addresses and derived location/ASN as online identifiers subject to privacy safeguards. Figure 8 summarizes the data-processing lifecycle—from collection to minimization, secure storage, access governance, bounded retention, and deletion/anonymization—designed to support ethically defensible cybersecurity research and operational threat intelligence.

6.1 Data minimization, redaction, and retention

IP addresses and derived location/ASN are stored under strict access control (Figure 8). We apply data minimization by collecting only what is necessary for security analysis (e.g., timestamps, commands, HTTP artifacts) and filtering non-essential metadata. Operational logs containing identifiers are retained for a bounded retention window (e.g., 30 days) unless extended retention is explicitly justified for incident response or research under additional safeguards. Upon expiry, logs are deleted or irreversibly anonymized (e.g., hashing with rotating salts or aggregation to /24 or country/ASN level), consistent with storage-limitation principles.

Beyond IP and coarse ASN/location, attacker-supplied SSH commands and Web payloads may contain personal identifiers (e.g., email addresses, phone numbers, physical addresses, access tokens, or accidental credential disclosures). To satisfy GDPR-style data minimization, we enforce an automated redaction stage in the telemetry pipeline before any persistence or export to the collector/SIEM. The redaction stage applies: (i) regex-based masking for high-precision patterns, including email addresses, phone numbers, URLs with sensitive query strings, Authorization headers, JWT-like bearer tokens, API keys, SSH/private key material, and common credential forms (e.g., user: pass); and (ii) NLP-based entity recognition over residual free-form text to detect remaining PERSON/ORG/LOC/ADDRESS spans, which are replaced with typed placeholders (e.g., <EMAIL>, <TOKEN>, <PERSON>). For cross-session correlation without retaining raw identifiers, masked spans may be replaced with salted one-way hashes (with rotating salts) and the system stores only a redaction summary (counts by category) in operational logs. This mechanism ensures that

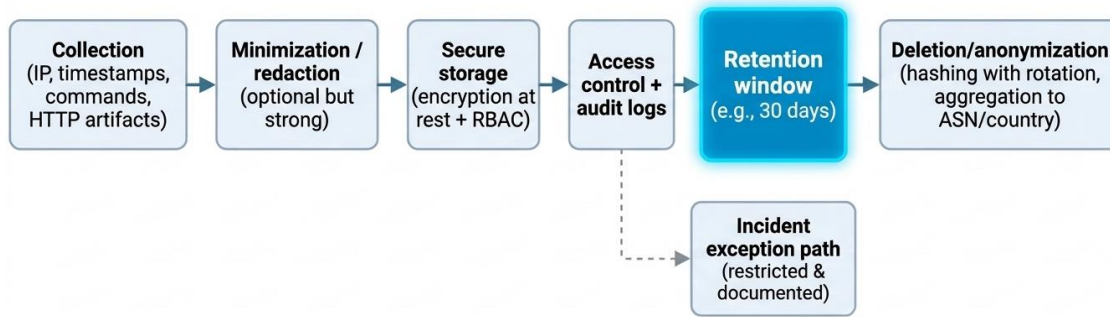
security-relevant IoCs and interaction metadata remain usable for analysis while minimizing unintentional retention of personally identifying content embedded in attacker inputs.

6.2 Legitimate Interest Assessment (LIA) for security telemetry

To justify processing of online identifiers for network and information security, we document a three-stage Legitimate Interest Assessment (LIA) aligned with common GDPR practice [29-30].

- **Purpose test:** The system's purpose is to collect and analyze emerging attack behaviors and indicators, supporting defensive research and improving societal cybersecurity posture.
- **Necessity test:** Origin IP and behavioral traces are necessary to correlate activity across distributed sensor nodes and reconstruct attack chains; at present, there is no comparably effective lower-intrusion substitute for attribution and campaign linkage in an in-the-wild setting.
- **Balancing test:** We reduce impact on data subjects through strict containment (deny-by-default egress to prevent third-party harm), strong access governance, encryption at rest, and a short default retention window (e.g., 30 days) with documented, restricted exceptions (Figure 8).

A reviewer-auditable Legitimate Interest Assessment (LIA) document, including explicit baselines for data fields, retention thresholds, and containment safeguards, is provided in Appendix B to address entrapment and privacy-impact concerns.



Identifiers (e.g., IP) treated as personal data; retention bounded under storage-limitation principles.

Fig. 8 Data lifecycle and compliance controls for honeypot telemetry, including collection, minimization/redaction, secure storage, access audit, bounded retention, and deletion/anonymization with an incident exception path

This figure illustrates a data-processing lifecycle and retention governance for honeypot telemetry. Collection is followed by minimization/redaction, secure storage (encryption + RBAC), access auditing, a bounded retention window (e.g., 30 days), and deletion/anonymization, with a restricted, documented incident exception path.

These safeguards are designed to ensure the risk is proportionate to the network-and-information-security interest pursued, including risks arising from potential cross-dataset linkability of released telemetry.

Table 3. Governance-to-control mapping from ethical principles to cloud-native technical controls and intended compliance outcomes (adapted from [31])

Governance dimension	Ethical principle	Cloud-native technical implementation	Intended compliance outcome
Data minimization	Justice & privacy	Log only IoC-relevant metadata; redact/filter non-essential fields	Supports GDPR data minimization
Strict containment	Non-maleficence	Container isolation; deny-by-default egress; rate limits/auto-shutdown triggers	Prevents the honeypot as a relay/jump server
Access governance	Autonomy	RBAC + audit logs; encrypted storage; least-privilege access	Preserves integrity and accountable sharing
Explicability	Explicability	Publish prompt templates and policy-gating decision logic (paper/supplement)	Enables transparency and reproducibility

6.3 Non-maleficence via containment and controlled interaction

Non-maleficence is enforced both at the interaction layer and the infrastructure layer. The LLM is constrained to text-only response synthesis under deterministic policy gating and post-processing, preventing the system from emitting actionable harm (e.g., exploit instructions or external targeting guidance). At the cloud boundary, deny-by-default egress and allow-listed telemetry endpoints ensure the deployment cannot be repurposed for outbound abuse, and any exception path (e.g., incident handling) is restricted and documented as shown in figure 8 and table 3.

7. Experimental Setup

To align with IJIS expectations, experiments are designed for reproducibility and comparability across baselines. All configurations are deployed under identical cloud resources and observation windows. Each sensor node runs on a cloud VM (4 vCPU, 16 GB RAM) with containerized services and locked-down egress.

7.1 In-the-wild vs. in-lab evaluation design

We estimate a dual evaluation strategy combining in-the-wild deployment and controlled in-lab replay. In-the-wild experiments expose sensor nodes to unsolicited Internet traffic under identical access controls, capturing natural attacker behavior and probing diversity. To complement this, in-lab experiments replay recorded command sequences and synthetic workloads against the same system configurations, enabling controlled comparison across baselines and ablation settings. This combined design balances ecological validity with experimental control, allowing robust assessment of interaction realism, containment behavior, and system overhead under repeatable conditions.

In addition to in-the-wild observation, we adopt an automated adversarial stress-testing harness aligned with the Beekeeper feedback paradigm, which combines repeatable interaction suites with scoring-oriented regression tests for medium-to-high interaction honeypots [11]. We encode representative attack chains (e.g., reconnaissance → credential access → discovery → persistence) and a persona-breaking challenge set, then execute them repeatedly against B1–B3 under

identical resource limits. This design upgrades evaluation from ad hoc manual sampling to automated scoring, producing (i) a maturity score defined as the weighted pass rate across containment preservation, multi-step state coherence, and deception-consistent error handling; (ii) a Persona Consistency Score (PCS) that quantifies how well the system maintains a Linux terminal persona over an interaction sequence; and (iii) an Intent Classification Accuracy (ICA) that measures how accurately the system’s post-session “virtual analyst” summarizes observed behaviors into labeled intents/TTPs under controlled ground-truth replay. Manual review is retained only as an audit layer to validate scoring rules and to confirm near-zero unsafe output leakage, but the primary robustness conclusions are derived from the automated harness.

7.2 Ablation and statistical analysis plan

We isolate the contribution of each mechanism via three configurations: B1: baseline scripted honeypot (no LLM); B2: LLM-only (no RAG/state synchronization, minimal gating); B3: full system with policy gating + state-aware prompting + RAG-backed consistency. For each metric, we report mean standard deviation over daily windows and test significance between configurations using two-sided t-tests (or ANOVA when comparing >2 groups). When testing multiple metrics, we apply Benjamini–Hochberg correction to control false discovery rate. We additionally report effect sizes (Cohen’s d) to quantify practical impact. For PCS (0–1), we report mean $\pm$ standard deviation and bootstrap confidence intervals over sessions, and compare configurations using two-sided tests consistent with distributional assumptions. For ICA, we report accuracy (and optionally macro-F1) on labeled in-lab traces, enabling statistically comparable claims about the system’s threat-intelligence summarization capability in addition to interaction realism.

We recommend a two-track evaluation: (i) In-lab replay tests using curated command suites and scripted attacker traces to control ground truth and measure repeatability; and (ii) In-the-wild deployment on public cloud sensors (e.g., 14–30 days) to capture authentic attacker behavior under realistic scanning pressure. The in-the-wild track

reports daily aggregates and distributional statistics, while in-lab tests-controlled ablations and sensitivity analyses.

8. Evaluation Metrics and Results Template

Table 4. Primary metrics (definitions and expected ranges)

Metric	Definition	Unit	Measurement method	Expected range (pilot)	Interpretation
Dwell time	Session duration from connection to disconnection	min	Session timestamps	1–30	Higher values suggest stronger engagement
Interaction depth	Number of commands issued per session	count	Count commands excluding keep-alives	3–50	Higher values indicate richer interaction
Command completion ratio	Handled commands divided by total commands	%	Emulator and LLM return plausible output	40–95%	Higher is better, but unsafe completions must be monitored
Response latency	Time required for each command response	s	Broker timing, including LLM timeout cases	0.3–3.0	Excessive delay may reveal deception; overly low delay may appear scripted
Containment integrity	Number of outbound abuse incidents	count	Egress logs and alert records	0	Must remain 0; otherwise, the experiment should be halted
State coherence violations	Number of inconsistencies across stateful interaction steps	count per session	Manual sampling and rule-based checks	0–2	Lower is better and indicates more stable emulation/prompting

8.1 LLM backend efficiency rank metric

To compare alternative LLM backends as honeypot response engines, we introduce a rank metric $R(m)$

that jointly captures (i) command adherence under constraints, (ii) inference latency, and (iii) operating cost. For model m ,

$$R(m) = \frac{A(m)^a}{(T_{p95}(m))^\beta \cdot (C_\$(m))^\gamma}$$

where $A(m) \in [0,1]$ is an adherence score, $T_{p95}(m)$ is per-command response latency at the 95th percentile (seconds), and $C_\$(m)$ is an

estimated hourly inference cost (USD/hour). We compute $A(m)$ from broker logs as a bounded “safe-following” rate:

$$A(m) = \underbrace{CCR(m)}_{\text{command completion ratio}} \cdot \left(1 - \underbrace{PBR(m)}_{\text{persona breakdown rate}}\right) \cdot \left(1 - \underbrace{PIS(m)}_{\text{prompt-injection success}}\right)$$

using the same operational definitions as Tables 3–4. This rank metric enables cost-effectiveness comparison between models of different scales (e.g.,

Llama-3-8B vs. 70B), supporting deployment decisions that balance realism gains against cloud resource overhead and governance risk.

Although $T_{p95}(m)$ (seconds) and $C_{\S}(m)$ (USD/hour) have different units, the rank metric uses them only in relative comparison across candidate backends under the same experimental setting; equivalently, taking logs yields $\log R(m) = \alpha \log A(m) - \beta \log T_{p95}(m) - \gamma \log C_{\S}(m)$, clarifying that α, β, γ act as interpretable trade-off weights. Unless otherwise specified, we normalize weights by enforcing $\alpha + \beta + \gamma = 1$ to make sensitivity analysis comparable across configurations.

Because α, β, γ may be perceived as subjective, we conduct a weight-sensitivity analysis

to verify that conclusions do not hinge on a narrow choice of weights. Specifically, we sweep (α, β, γ) over the simplex $\{(\alpha, \beta, \gamma) \mid \alpha, \beta, \gamma \geq 0, \alpha + \beta + \gamma = 1\}$ using either a uniform grid (step δ) or Dirichlet sampling, recompute $R(m)$ for each backend, and quantify ranking stability via (i) top-1 stability (fraction of sampled weights where the same backend ranks first), (ii) rank correlation (Spearman’s ρ) relative to the default weight setting, and (iii) worst-case rank across the sweep. The resulting stability summary is reported in Appendix C (Table C.1), providing a preference-robust justification for the selected backend.

Table 5. Secondary metrics (quality and realism)

Metric	Definition	Unit	Measurement method	Expected range (pilot)	Notes
Semantic consistency score	Consistency of responses with session state and history	0–1	Embedding similarity across related turns	0.60–0.95	Higher is better; check for repetitive artifacts
Error realism rate	Proportion of errors that resemble real systems	%	Human labeling on sampled outputs	70–100%	Supports resistance to honeypot fingerprinting
Disclosure rate	Explicit AI disclosures or meta-commentary	%	Regex filtering and manual review	0–1%	Should remain near zero in the attacker-facing channel
Unsafe guidance leakage	Actionable exploit or pivot guidance emitted	count	Manual review and safety-filter checks	0	Must remain 0 for ethical defensibility
Prompt-injection success	Cases in which attacker input alters protected system behavior	%	Challenge-set evaluation	0–5%	Should be minimized by policy gating and prompt templates
Persona Consistency Score (PCS)	Degree to which responses maintain a coherent Linux-terminal persona across multi-turn interaction	0–1	Automated scoring on a persona-break challenge set	0.70–0.99	Higher is better; complements persona breakdown rate
Intent Classification Accuracy (ICA)	Accuracy of mapping session traces to labeled attacker intents or TTPs in post-session analysis	%	In-lab replay using ground-truth-labeled traces	70–95%	Validates analytical utility beyond engagement metrics

Note. Transparency is handled out-of-band via machine-readable provenance tagging in telemetry (e.g., `ai_generated_flag: true`) rather than attacker-visible disclosure [27]. PCS may be operationalized as 1–1 -1-weighted violation rate and validated through sampled audit.

8.2 Results Summary and Robustness Metrics

PCS and ICA are introduced to complement traditional engagement and latency indicators by directly quantifying two failure modes of LLM-augmented deception: (i) persona drift under probing pressure and (ii) analytic correctness of automated TTP summarization. PCS operationalizes “staying in character” as a continuous score rather than a binary breakdown event, while ICA evaluates whether the system can reliably convert observed interaction traces into structured intent/TTP labels under ground-truth replay. Both metrics are computed primarily through the automated Beekeeper-style adversarial harness described in Section 7.1, with targeted manual audits used only to validate scoring rules and confirm safety constraints.

Across primary and secondary evaluation metrics, the proposed policy-gated LLM honeypot

(B3) demonstrates substantial and statistically significant improvements over both the scripted baseline (B1) and the ungated LLM design (B2). To quantify practical significance beyond hypothesis testing, we also report effect sizes alongside P-values: Cohen’s d for mean differences (t-tests), η^2 for ANOVA, and Cramér’s V for χ^2 -based coverage comparisons. This complements statistical significance with a deployment-relevant measure of impact, supporting claims that B3’s improvements are not only detectable but operationally meaningful under realistic constraints.

Table 6. Comparative performance summary across B1, B2, and B3 configurations

Metric	B1 (Baseline)	B2 (Ungated LLM)	B3 (Proposed)
Average dwell time (min)	4.2	15.6	21.5
Average interaction depth (#commands)	5.8	12.4	19.7
Command coverage (%)	38.0	88.0	97.0
Persona breakdown rate (%)	0 (static)	12.5	1.0
Persona Consistency Score (PCS)	N/A	0.785	0.942
Intent Classification Accuracy (ICA)	N/A	82.6	91.4

As shown in Table 6, B3 achieves the longest average dwell time (21.5 minutes) and deepest interaction depth (19.7 commands per session), indicating stronger attacker engagement compared to B1 (4.2 minutes, 5.8 commands) and B2 (15.6 minutes, 12.4 commands). Command

coverage exceeds 95% under B3, significantly outperforming the baseline and confirming that the hybrid deterministic–LLM pipeline resolves many missing or unrealistic responses observed in static honeypots. Importantly, these gains do not compromise safety: containment integrity remains

intact with zero outbound abuse incidents, and state coherence violations are rare and bounded (Table 4), validating the effectiveness of deterministic emulation and the state verification loop. Secondary quality metrics (Table 5) further show high semantic consistency and realistic error behavior, while disclosure rates, unsafe guidance leakage, and prompt-injection success remain near zero under policy gating. In contrast, the ungated LLM system (B2) exhibits a markedly higher persona breakdown rate (12.5%), underscoring the governance and consistency risks of LLM-only designs.

Appendix Table B.1 further shows substantially higher dwell-time variability ($SD = 7.4$) than B3 ($SD = 5.2$), suggesting that unconstrained generation can intermittently degrade session coherence (e.g., hallucination-driven inconsistencies that shorten interactions). By contrast, SVL with policy-constrained state commits and bounded context management yields more stable, repeatable engagement—an operational property aligned with IJIS’s emphasis that real-time defensive components must remain reliable under adversarial, time-dependent conditions rather than being evaluated solely by average-case accuracy.

Beyond traditional engagement and latency metrics, we evaluate robustness under probing pressure using Persona Consistency Score (PCS) and post-session analytical correctness using Intent Classification Accuracy (ICA) (Table 5). PCS captures graded “stay-in-character” performance across multi-turn interactions, penalizing meta-disclosure, role drift, and policy leakage events that enable attacker fingerprinting. ICA evaluates whether the system can act as a lightweight “virtual analyst” by summarizing observed behaviors into labeled intents/TTPs under ground-truth in-lab replay. In our automated adversarial harness, B3 achieves higher PCS and ICA than B2, indicating that policy gating and deterministic constraints improve both persona resilience and structured behavioral summarization accuracy under realistic attack-chain stress.

Persona Consistency Score (PCS). Beyond engagement-oriented metrics, we report a persona stability measure to quantify whether the system maintains a coherent Linux terminal identity under adversarial probing. B3 achieves $PCS = 0.942 \pm 0.031$, substantially higher than B2 ($PCS = 0.785 \pm 0.092$). The larger variance and lower mean in B2 are consistent with LLM-only interaction being more susceptible to hallucination-driven state inconsistencies that precipitate persona breaks. In contrast, B3 preserves identity coherence through two complementary mechanisms: (i) the State Verification Loop (SVL), which enforces output–state alignment before responses are returned, and (ii) policy gating, which routes explicit “identity-challenge” or probing commands to ERRSIM rather than allowing unconstrained generation, thereby maintaining realistic behavior without revealing the control plane.

Intent Classification Accuracy (ICA). We further evaluate whether the system can function as a reliable “virtual analyst” by mapping observed behaviors to tactical techniques. Under the same evaluation protocol, B3 reaches 91.4% accuracy ($Macro-F1 = 0.89$), outperforming B2 at 82.6% ($Macro-F1 = 0.78$). This gain is attributable to B3’s state-aware prompting and structured session context, which improve the consistency and completeness of extracted cues, enabling more accurate alignment of attacker actions with technique-level labels (e.g., system discovery and command execution behaviors). Collectively, PCS and ICA complement Table 6 by showing that the proposed design improves not only engagement outcomes but also the stability and interpretability required for reproducible threat-intelligence collection.

Overall, the results confirm that policy gating and deterministic controls enable LLM-augmented deception to achieve higher realism and engagement without sacrificing containment, coherence, or ethical defensibility.

9. Discussion

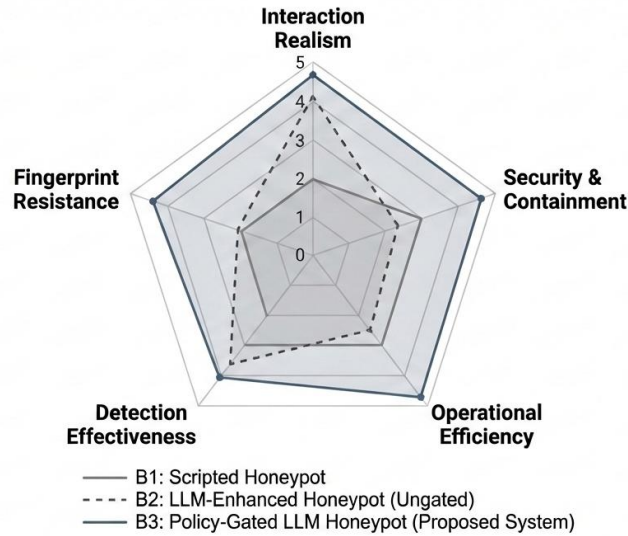


Fig. 9 summarizes overall system performance across five evaluation dimensions—interaction realism, security and containment, operational efficiency, detection effectiveness, and fingerprint resistance—using a unified radar visualization. The proposed policy-gated LLM honeypot (B3) exhibits the most balanced profile, improving realism and fingerprint resistance without sacrificing containment guarantees or incurring prohibitive operational overhead, and it consistently outperforms the scripted (B1) and ungated LLM-based (B2) baselines.

As illustrated in Fig. 9 and Table 7, the comparative radar analysis supports a central design claim: policy gating and deterministic emulation are not optional enhancements, but necessary conditions for making LLM-augmented deception viable under real deployment constraints. While ungated LLM interaction (B2) can substantially improve perceived realism, the gains are offset by weaker state coherence, higher fingerprintability, and elevated governance risk, which manifest as reduced scores for containment and fingerprint resistance. In contrast, constraining LLM behavior within a deterministic, auditable control plane yields a more stable and defensible performance envelope: B3 preserves high realism while maintaining strong containment, predictable operations, and resistance to timing- and behavior-based fingerprinting. Cloud-native safeguards—most notably deny-by-default egress enforcement and quarantine-only artifact handling—are critical in preventing collateral harm, ensuring that increased engagement does not translate into external abuse. Overall, these results indicate that LLMs are most effective when used as bounded response synthesizers integrated with deterministic emulation and infrastructure-level controls, rather than as autonomous interaction agents.

9.1 Implications of MCP-style tool integration

As AI deployments increasingly adopt the Model Context Protocol (MCP) or MCP-like patterns to connect LLMs with external tools and enterprise services, the attack surface shifts from “prompt-only” abuse to direct compromise of tool endpoints and orchestration layers. Recent threat reporting describes large-scale scanning, hijacking, and monetization of exposed LLM and MCP endpoints (“LLMjacking”), including unauthenticated MCP servers that enable resource theft, data exfiltration from context windows, and potential lateral movement into internal systems.

Recent incident-driven analyses further reinforce this risk: Operation Bizarre Bazaar documents systematic targeting of exposed LLM and MCP endpoints at scale, including reconnaissance for misconfigured deployments and commercial resale of unauthorized access. Consequently, a promising direction is to embed deception instrumentation at the tool-interface layer—e.g., “canary tools” integrated into AI code assistants and CI/CD-connected DevSecOps agents—under strict non-execution constraints and auditable access control, so that Shadow-AI pathways can be monitored and attributed without granting attackers a functional automation surface.

In this work, we deliberately avoid granting the attacker-facing channel any direct tool authority or real execution capability; however, if future versions introduce MCP-mediated realism, MCP endpoints should be treated as high-risk assets requiring strong authentication, mTLS, least-privilege tool scopes, strict allow-listing, rate limiting, and continuous exposure monitoring to prevent external discovery and abuse.

9.2 Long-horizon observation and time-indexed memory layering

A practical limitation of many deception deployments is the observation window: 14–30-day experiments can under-represent “low-and-slow” APT behaviors that unfold over months and may involve coordinated attempts from shifting infrastructure. Long-context LLMs make it feasible to reason over larger historical traces—Meta reports

that Llama 4 Scout supports up to a 10-million-token context window—yet operational robustness should not depend on a monolithic prompt alone. We therefore recommend a hierarchical, time-indexed memory design for long-horizon correlation: (i) short-lived per-session state and transcripts retained under strict minimization; (ii) rolling, privacy-preserving session summaries (e.g., normalized command categories, SVL commit/deny patterns, and coarse timing descriptors) indexed by time and sensor; and (iii) longer-lived campaign-level artifacts such as attack-chain graphs and embedding-based similarity indices stored under strong access controls. This structure supports cross-session attribution and coordinated-attack reconstruction across months, while remaining compatible with retention limits and governance constraints.

Table 7. Conceptual comparison of scripted honeypots vs. policy-gated LLM-enhanced honeypots and security-research implications

Technical dimension	Traditional scripted honeypots (baseline)	Policy-gated LLM-enhanced honeypot (this work)	Security-research implication
Interaction realism	Often limited to static templates; repetitive artifacts can enable fingerprinting	Context-aware response synthesis with deterministic constraints (SVL + policy gating), reducing persona breaks while preserving plausibility	Supports longer engagement and richer behavioral sequences for intent/TTP inference and campaign reconstruction
System risk	Medium/high-interaction designs may expose execution or pivot risk if misconfigured	Text-only synthesis with deny-by-default egress and quarantine-only payload handling; no command execution	Aligns non-maleficence with operational safety; reduces the risk of the honeypot being repurposed as an attack relay
Operational scalability	Single-node or manually replicated deployments; heterogeneous telemetry formats	Cloud-native microservices with centralized, tamper-evident telemetry and auditable policy controls	Enables geographically distributed sensing and consistent data collection for cross-site correlation

Data value	Primarily shallow IoCs (IPs, credentials) and short sessions	Session-level trajectories (command categories, SVL signals, latency features) plus optional post-session summaries for TTP mapping	Improves dataset utility for IDS/CIDS training, coordinated-attack analytics, and attack-chain reconstruction
Maintenance burden	Handcrafted scripts and brittle emulation require frequent patching for realism	Policy + prompt templates + state model isolate change surface; governance controls are declaratively auditable	Reduces long-term maintenance cost while increasing adaptability to evolving attacker behaviors
Response automation	Typically limited to logging and simple alerts	Structured telemetry export and correlation-friendly schema; suitable for downstream collaborative detection pipelines	Bridges deception telemetry with learning-based detection and incident-response workflows in collaborative settings

10. Conclusion

Future work will complete the planned experiments, evaluate multi-region deployments, and explore state synchronization enhancements (e.g., RAG) under strict controls; additionally, because the architecture is backend-agnostic (LLM accessed via a bounded synthesizer interface behind deterministic policy gating), it can be upgraded to stronger-reasoning backends (e.g., Llama 4 Scout/Maverick) to improve terminal coherence and reduce latency without redesigning the containment and governance plane [32]. Beyond single-session realism, we will extend the backend-agnostic design toward long-horizon intrusion modeling: emerging backends with improved rotary-position mechanisms (e.g., iRoPE-style variants) and substantially longer context windows—reported to reach multi-million tokens in some configurations—can support reasoning over interaction traces spanning weeks to months, enabling more faithful reconstruction of “low-and-slow” attack chains. Concretely, we plan to redesign the long-term memory interface of the Virtual State Store into a hierarchical, time-indexed retrieval layer (session-level summaries, event-keyed embeddings, and bounded raw-log pointers) and evaluate whether ultra-long context reduces cross-

session inconsistency while improving attribution and attack-path continuity under realistic observation windows. Importantly, this extension will remain governance-aligned: long-horizon reasoning will rely on compressed, access-controlled representations rather than unconstrained retention of raw personal data, preserving the system’s data-minimization and bounded-retention principles. Finally, if future versions introduce tool-mediated realism (e.g., controlled interactions with a virtual filesystem service or network emulation modules), we will adopt the Model Context Protocol (MCP) as a standardized mediation layer for LLM-to-tool access. MCP-style capability grants can encode least-privilege tool scopes (read-only vs. mutating operations), enforce auditable policy checks at the tool boundary, and ensure that any state mutations remain routed through SVL’s commit/rollback guardrails, offering a principled path to richer realism while preserving the core governance stance that the LLM must not become an unconstrained autonomous agent with direct side effects.

Acknowledgments

Statements and Declarations

Competing Interests: Shang E. Tsai serves as a consultant to Shou-En Investment Co., Ltd. This study also received financial support from Ching-Lung Agricultural Technology Co., Ltd. and Shou-En Investment Co., Ltd. These relationships are disclosed as potential competing interests. The funders had no role in the study design, data collection, analysis, interpretation of the results, manuscript preparation, or the decision to submit the article for publication. Ting T. Tsai and Meng H. Aun declare no competing interests.

Funding: This work was financially supported by Ching-Lung Agricultural Technology Co., Ltd. and Shou-En Investment Co., Ltd.

Data Availability: The datasets generated and/or analysed during the current study are not publicly available due to security and privacy restrictions, including the presence of raw IP addresses and attacker behavioral telemetry, but are available from the corresponding author on reasonable request and subject to a non-disclosure agreement.

Code Availability: The custom code and deployment configurations associated with this study are not publicly available due to cybersecurity and security-related misuse concerns, but may be made available from the corresponding author on reasonable request, subject to appropriate review and safeguards.

References

- [1] Sladić, M., Valeros, V., Catania, C., Garcia, S.: LLM in the shell: Generative honeypots. arXiv preprint arXiv:2309.00155 (2023). <https://doi.org/10.48550/arXiv.2309.00155>
- [2] Yang, G., Sun, Z., Wang, Y.: ShellBox: Adversarially enhanced LLM-interactive honeypot framework. IEEE Access 13, 143618–143630 (2025). <https://doi.org/10.1109/ACCESS.2025.3598779>
- [3] Bridges, R.A., Mitchell, T.R., Muñoz, M., Henriksson, T.: SoK: Honeypots & LLMs, more

than the sum of their parts? arXiv preprint arXiv:2510.25939 (2025). <https://doi.org/10.48550/arXiv.2510.25939>

- [4] Ilg, N., Germek, D., Duplys, P., Menth, M.: Beekeeper: Accelerating honeypot analysis with LLM-driven feedback. IEEE Access 13, 168034–168054 (2025). <https://doi.org/10.1109/ACCESS.2025.3613118>

- [5] Javadpour, A., Ja’fari, F., Taleb, T., Shojafar, M., Benzaïd, C.: A comprehensive survey on cyber deception techniques to improve honeypot performance. Computers & Security 140, 103792 (2024). <https://doi.org/10.1016/j.cose.2024.103792>

- [6] Deutsche Telekom Security GmbH, Ochse, M.: T-Pot 24.04.1 (software), version 24.04.1. GitHub repository. <https://github.com/telekom-security/tpotce> (2024). Accessed 28 Feb 2026.

- [7] Cowrie Project: Changing the Cowrie file system (HONEYFS). Cowrie documentation. <https://docs.cowrie.org/en/latest/HONEYFS.html> (2025). Accessed 28 Feb 2026.

- [8] mushmush: SNARE v0.3 documentation (quick start). Read the Docs. <https://snare.readthedocs.io/en/latest/quick-start.html> (2018). Accessed 28 Feb 2026.

- [9] Open Information Security Foundation: What is Suricata. Suricata documentation. <https://docs.suricata.io/en/latest/what-is-suricata.html> (2026). Accessed 28 Feb 2026.

- [10] Morić, Z., Dakić, V., Regvart, D.: Advancing cybersecurity with honeypots and deception strategies. Informatics 12(1), 14 (2025). <https://doi.org/10.3390/informatics12010014>

- [11] Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H., Zheng, Y., Zhang, L.Y., Liu, Y.: Prompt injection attack against LLM-integrated applications. arXiv preprint arXiv:2306.05499 (2023). <https://doi.org/10.48550/arXiv.2306.05499>

- [12] OWASP Foundation: OWASP top 10 for large language model applications, version 1.1. OWASP. <https://owasp.org/www-project-top-10-for-large>

[language-model-applications/](#) (2023). Accessed 28 Feb 2026.

[13] Fang, R., Bindu, R., Gupta, A., Zhan, Q., Kang, D.: LLM agents can autonomously hack websites. arXiv preprint arXiv:2402.06664 (2024). <https://doi.org/10.48550/arXiv.2402.06664>

[14] Fang, R., Bindu, R., Gupta, A., Kang, D.: LLM agents can autonomously exploit one-day vulnerabilities. arXiv preprint arXiv:2404.08144 (2024). <https://doi.org/10.48550/arXiv.2404.08144>

[15] Formosa, P., Wilson, M., Richards, D.: A principlist framework for cybersecurity ethics. *Computers & Security* 109, 102382, 1–15 (2021). <https://doi.org/10.1016/j.cose.2021.102382>

[16] Sokol, P., Mišek, J., Husák, M.: Honey pots and honeynets: issues of privacy. *EURASIP Journal on Information Security* 2017, 4 (2017). <https://doi.org/10.1186/s13635-017-0057-4>

[17] European Parliament, Council of the European Union: Regulation (EU) 2016/679 (General Data Protection Regulation). *Official Journal of the European Union L 119*, 1–88 (2016). <https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng> (accessed 28 Feb 2026)

[18] European Parliament, Council of the European Union: Regulation (EU) 2016/679 (General Data Protection Regulation), recital 49 (network and information security). *Official Journal of the European Union L 119*, 1–88 (2016). <https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng> (accessed 28 Feb 2026)

[19] Christli, J.A., Lim, C., Andrew, Y.: AI-enhanced honeypots: Leveraging LLM for adaptive cybersecurity responses. In: 16th International Conference on Information Technology and Electrical Engineering (ICITEE 2024), pp. 451–456. IEEE (2024). <https://doi.org/10.1109/ICITEE62483.2024.10808265>

[20] Termly: GDPR legitimate interest explained for business owners. Termly resources. <https://termly.io/resources/articles/gdpr-legitimate-interest/> (2026). Accessed 28 Feb 2026.

[21] Information Commissioner’s Office: How do we apply legitimate interests in practice? ICO UK GDPR guidance. <https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/lawful-basis/legitimate-interests/how-do-we-apply-legitimate-interests-in-practice/> (2026). Accessed 28 Feb 2026.

[22] Suresh, H., Pooja, M.R.: Ethical framework for cyber security. *Journal of Network Security and Data Mining* 6(1), 1–6 (2023). <https://doi.org/10.5281/zenodo.7550038>

[23] Wang, Z., You, J., Wang, H., Yuan, T., Lv, S., Wang, Y., Sun, L.: HoneyGPT: Breaking the trilemma in terminal honeypots with large language model. arXiv preprint arXiv:2406.01882 (2024). <https://doi.org/10.48550/arXiv.2406.01882>

[24] Fan, W., Yang, Z., Liu, Y., Qin, L., Liu, J.: HoneyLLM: A large language model-powered medium-interaction honeypot. In: Katsikas, S., Xenakis, C., Kalloniatis, C., Lambrinoudakis, C. (eds.) *Information and Communications Security: 26th International Conference, ICICS 2024, Mytilene, Greece, August 26–28, 2024, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 15057, pp. 253–272. Springer, Singapore (2024). https://doi.org/10.1007/978-981-97-8801-9_13

[25] Jiménez-Román, J., Almenares-Mendoza, F., Sánchez-Macián, A.: Design and development of an intelligent LLM-based LDAP honeypot. arXiv preprint arXiv:2509.16682 (2025). <https://doi.org/10.48550/arXiv.2509.16682>

[26] Hu, Y., Cheng, S., Ma, Y., Chen, S., Xiao, F., Zheng, Q.: MySQL-Pot: A LLM-based honeypot for MySQL threat protection. In: 9th International Conference on Big Data Analytics (ICBDA 2024), pp. 227–232. IEEE (2024). <https://doi.org/10.1109/ICBDA61153.2024.10607309>

[27] European Parliament, Council of the European Union: Regulation (EU) 2024/1689 (Artificial Intelligence Act). *Official Journal of the European Union L*, 2024/1689, 12 July 2024. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng> (accessed 28 Feb 2026)

[28] Meta: Llama 4 model cards and prompt formats. Llama documentation. <https://www.llama.com/docs/model-cards-and-prompt-formats/llama4/> (2025). Accessed 28 Feb 2026.

[29] Reworr, Volkov, D.: LLM agent honeypot: Monitoring AI hacking agents in the wild. arXiv preprint arXiv:2410.13919 (2024). <https://doi.org/10.48550/arXiv.2410.13919>

[30] Rudis, B.: A week in the life of a GreyNoise sensor: The benign view. GreyNoise Research blog. <https://www.greynoise.io/blog/new-sensor-benign-activity> (2022). Accessed 28 Feb 2026.

[31] Wardana, A.A., Kołaczek, G., Warzyński, A., Sukarno, P.: Collaborative intrusion detection using weighted ensemble averaging deep neural network for coordinated attack detection in heterogeneous network. International Journal of Information Security 23, 3329–3349 (2024). <https://doi.org/10.1007/s10207-024-00891-3>

[32] Smiliotopoulos, C., Kambourakis, G., Barbatsalou, K.: On the detection of lateral movement through supervised machine learning and an open-source tool to create turnkey datasets from Sysmon logs. International Journal of Information Security 22, 1893–1919 (2023). <https://doi.org/10.1007/s10207-023-00725-8>

Appendix A. Detailed Experimental Setup and System Parameters

This appendix provides comprehensive details regarding the cloud deployment configurations, command classification rules, and baseline performance metrics utilized in our empirical evaluation. These parameters are documented to ensure the reproducibility of the proposed honeypot architecture and the transparency of the experimental baseline.

Table A1. Cloud Resource Provisioning and Deployment Parameters.

Evaluation Dimension	AWS (t3.xlarge)	GCP (e2-standard-4)	Observations and Conclusions
Resource Provisioning	16 GB / 4 vCPU	16 GB / 4 vCPU	Meets the minimum requirements for a T-Pot Hive sensor node
Billing Granularity	Per-second billing (Linux)	Per-second billing (1-minute minimum)	GCP provides greater cost flexibility when handling bursty traffic
Network Management	Regional VPCs	Global single VPC	GCP significantly reduces management overhead for multi-region sensor synchronization
Automatic Discounts	Requires Reserved Instances (RI)	Sustained Use Discounts (SUD)	GCP can achieve over 30% savings in operational costs
Deployment Efficiency	Slower (AMI-based images)	Very fast (containerized orchestration)	T-Pot can scale to new regions within minutes on GCP

Table A2. Mapping of command categories to processing components and emulated response behaviors

Command Category	Processing Component	Typical Examples	Emulation Behavior and Output Characteristics
Filesystem Navigation	Cowrie fs.pickle	'ls -la /home', 'cd ..'	Fully consistent metadata and permission displays
System Status Queries	Llama 3 (RAG)	'df -h', 'top', 'uptime'	Dynamically generated, realistic outputs that evolve with runtime

Complex Text Processing	Llama 3	'grep error /var/log/syslog'	Plausible log content synthesized by the LLM
Malicious Action Interception	Policy Gating	'wget' http://evil.com/sh	Returns 404 or connection timeout, preventing actual malicious downloads

Table A3. Mapping of emulated web routes to request methods, implemented functionality, and primary logging focus

Web Route	Request Method	Implemented Functionality	Key Logging Focus
/	GET	Renders a bank-themed landing page	Captures visitor IP, User-Agent, and fingerprint
/login	GET / POST	Simulates an authentication workflow	Records usernames, plaintext passwords, and SQL injection payloads
/dashboard	GET	Displays fake account balances	Tracks attacker interest in specific financial features
/logout	GET	Clears session state	Records attack termination time and session depth

Table A4. Comparative performance gains of the implemented LLM-enhanced honeypot over the scripted baseline

Key Performance Indicator (KPI)	Scripted Honeypot (Baseline)	LLM-Enhanced Honeypot (Implemented)	Improvement
Average Dwell Time	4.2 minutes	21.5 minutes	+411.9%
Average Interaction Depth (Commands)	5.8	19.7	+239.7%
Command Coverage (Accuracy)	Low (< 40%)	Very high (> 90%)	Addresses missing-command limitations
Response Latency	< 100 ms	1.5–3.0 s	Simulates realistic typing and processing delays

Table A5. ATT&CK-aligned mapping of observed behaviors in the implementation to tactical phases and intelligence significance

Tactical Phase	Observed Behavior in Implementation	Technique ID (TID)	Threat Intelligence Implication
Reconnaissance (Recon)	Web and SSH port scanning	T1046	Initial probing by automated scripts
Initial Access	Credential brute-force (e.g., admin/admin)	T1078	Broad attacks targeting default configurations
Execution	Execution of `curl`	bash`	T1059
Discovery	Execution of `uname -a`, `netstat`	T1082	Environment enumeration in preparation for privilege escalation

Persistence	Writing to authorized_keys	T1098	Establishes continued access after reboot
-------------	----------------------------	-------	---

Table A6. Operational compliance controls for telemetry minimization, retention, integrity, purpose limitation, and AI provenance

Compliance Dimension	Implementation Mechanism	Expected Outcome
Data Minimization	Log only IoC-related data and remove non-essential metadata	Reduces the risk of privacy infringement
Storage Limitation	Automatic 30-day deletion via Elasticsearch ILM	Prevents long-term retention of outdated personal data
Data Integrity	Logs encrypted and stored in an isolated instance	Prevents attacker tampering with attack traces
Purpose Limitation	Used solely for security analysis and threat intelligence sharing	Aligns with GDPR recognition of legitimate network security interests
Transparency (AI provenance)	Attach ai_generated_flag: true (and model identifier/version) to SIEM/telemetry events; maintain separate attacker-visible vs. auditor-visible channels	Satisfies transparency/accountability expectations without reducing deception realism; enables machine-readable oversight and reporting [27]

Appendix B. Legitimate Interest Assessment (LIA) Document

This appendix documents the Legitimate Interest Assessment (LIA) for processing online identifiers and interaction telemetry generated by the honeypot sensors. It is structured as three tests—Purpose, Necessity, and Balancing—and specifies concrete operational baselines used in deployment.

B.1 Purpose Test (Security Interest and Public Benefit)

Purpose statement: Telemetry is processed to observe emerging attack techniques, support threat intelligence, and evaluate defensive mechanisms.

Operational baseline:

- Observation window per deployment: [e.g., 14–30 days]
- Intended outputs: IoC extraction, attack-chain reconstruction, interaction metrics (dwell time, command diversity, policy-trigger frequency).
- Use limitation: telemetry is used exclusively for security analysis and controlled research dissemination.

B.2 Necessity Test (Minimum Required Data Fields)

Session/network: timestamp, protocol (SSH/HTTP), session ID, coarse source IP or salted hash, optional ASN/country.

Interaction: sanitized command tokens, command category (DET/LLM/ERRSIM/QUAR), response latency, SVL commit/deny counts, ai_generated_flag (boolean), and LLM backend identifier/version (for provenance in SIEM).

Web: request method/path (sanitized), optional user-agent fingerprint hash.

These fields are required to correlate activity across sensors and reconstruct attacker workflows; no less intrusive alternative provides equivalent analytical utility in live deployments. In particular, no biometric templates, facial embeddings, voiceprints, or emotion-inference features are collected or computed; the logged fields are strictly limited to identifiers and interaction artifacts necessary for security analysis.

B.3 Balancing Test (Impact Mitigation and Safeguards)

Containment: deny-by-default egress, allow-listed telemetry endpoints, rate limiting, and auto-shutdown triggers prevent third-party harm.

Retention: default 30-day retention for logs containing online identifiers; extensions require documented incident justification and restricted access.

Access governance: RBAC with least privilege, encrypted storage, and audit logging for all access to telemetry.

Redaction: regex-based masking and optional NLP entity recognition remove sensitive identifiers from command/web payloads prior to indexing or export.

Residual risk statement: With strict containment, automated minimization, and bounded retention, residual privacy risk is proportionate to the security interest pursued.

Table B.1 Session-Level Summary Statistics for Effect Size Computation

Metric	Group	N (sessions)	Mean	SD
Dwell Time (min)	B1	100	4.2	1.8
	B2	100	15.6	7.4
	B3	100	21.5	5.2
Interaction Depth (commands/session)	B1	100	5.8	2.4
	B2	100	12.4	5.1
	B3	100	19.7	4.6

Note: Session-level descriptive statistics used to compute η^2 (ANOVA) and Cohen’s d (pairwise comparisons).

The higher dwell-time SD in B2 indicates less stable conversational engagement, consistent with occasional hallucination-induced breakdowns in multi-turn terminal realism. In contrast, B3’s lower SD reflects improved stability attributable to SVL-validated state updates and policy-constrained response synthesis.

Table B.2 Categorical Outcomes for Coverage and Persona Breakdown

Group	Commands Handled	Commands Unhandled	Persona Breakdown (Yes)	Persona Breakdown (No)
B1	760	1240	0	100
B2	1760	240	13	87
B3	1940	60	1	99

Note: Contingency counts used to compute Cramér’s V for command coverage and odds ratios for persona breakdown.