

Runtime Monitoring and Risk-Adaptive Control for LLM Agents

Deepak Kumar Gour¹ Dr. Sushma Agrawal²

¹Department of Computer Science and Engineering, Jyoti Vidhyapeeth Women's University, Jaipur, Rajasthan, India

²Department of Computer Science and Engineering, Jyoti Vidhyapeeth Women's University, Jaipur, Rajasthan, India

Abstract - Tool-enabled large language model agents can retrieve content, call external tools, and initiate actions. These capabilities also create exposure to prompt injection, unsafe tool use, data leakage, and poorly bounded autonomy. This paper documents AgentGuard, a CPU-only prototype that combines a lexical Secure Runtime Monitoring Protocol (SRMP), a six-factor Adaptive Agent Risk Scoring model (AARS), and a Risk-Adaptive Scheduling State Machine (RASS). The study analyses archived simulation records rather than a new deployment experiment. In the primary 200-event simulation, the combined pipeline produced a precision of 0.905, recall of 0.792, F1 of 0.844, and specificity of 0.974 against SRMP-linked reference labels. Across ten fixed-seed runs, mean F1 was 0.791 (SD 0.051; run-level bootstrap 95% CI 0.760-0.819). The lexical layer was vulnerable to evasion: detection fell from 0.76 on original attack prompts to 0.12 after synonym substitution and combined perturbations, while recall on a 64-prompt internal benchmark was 0.529. Because some labels are generated from SRMP output, the evaluation is not independent. The findings are therefore an internal characterization of an inspectable prototype, not evidence of operational detection performance. The paper identifies the validation needed before deployment-oriented claims can be made.

Key Words: LLM agents; runtime monitoring; prompt injection; risk scoring; adaptive control; agent security; simulation evaluation

1. INTRODUCTION

Large language model (LLM) agents extend the model beyond conversation. They can retrieve documents, inspect external content, call tools, and carry a task through several decisions. Each added capability creates another point at which untrusted input may influence the system. OWASP treats prompt injection as a major risk for LLM applications and separates direct instructions from indirect instructions hidden in documents, web pages, files, or tool responses [1]. Once an agent has permission to act, a successful injection can affect an external system rather than merely alter generated text.

Recent benchmarks show that this is a practical security problem. HarmBench provides a common basis for adversarial red-teaming across harmful behaviours [2]. AgentHarm shifts the focus from single-turn chatbot responses to malicious tasks completed through several agent steps [3]. InjecAgent studies indirect injection in agents connected to tools [4], whereas Agent Security Bench includes prompt injection, memory poisoning, and mixed attacks [5]. These studies evaluate live models and tool

environments. Their numerical results cannot be transferred directly to a small, synthetic, CPU-only prototype.

AgentGuard was developed to explore that lightweight setting. SRMP first checks event text against a compact, versioned rule set. AARS combines the resulting security evidence with

infrastructure, model, agent, retrieval-augmented generation (RAG), and user-history signals. RASS then converts the composite score into a control state and a recommended response. The architecture reflects a broader security principle: monitoring works best alongside least privilege, traceable decisions, and human oversight [1], [6], [7], not as a stand-alone text filter.

Studying has two aims. The first is to document how SRMP, AARS, and RASS interact, including the points at which the prototype can recommend intervention. The second is to examine the evidence already produced by the project across its canonical runs, repeated seeds, evasion tests, internal benchmark, and larger synthetic workloads. The comparison with recent monitoring and guardrail research is qualitative because the experimental settings are different. Particular attention is given to three constraints in the present evidence: synthetic telemetry, overlap between prompts and lexical rules, and labels that are partly derived from SRMP output.



Figure 1. AgentGuard monitoring and control flow.

2. MATERIALS AND METHOD

A. Prototype Architecture

Fig. 1 shows the path followed by an event. The event record contains the prompt, simulated resource telemetry, model and token information, tool calls, the number of RAG sources, and user context. SRMP checks the prompt against 23 substring rules covering prompt injection, jailbreaks, system-prompt extraction, data leakage, unsafe tool instructions, excessive agency, suspicious tool requests, and exposure of sensitive data. Its output is a set of security flags with associated severities.

AARS converts these observations into a single risk value. The implemented score is the weighted sum of six normalized factors:

$$R = 0.15 R_{infra} + 0.15 R_{model} + 0.20 R_{agent} + 0.15 R_{rag} + 0.25 R_{security} + 0.10 R_{user} \quad (1)$$

RASS assigns one of four states to the score: ALLOW below 0.30, MONITOR from 0.30 to below 0.60, RESTRICT from 0.60 to below 0.80, and BLOCK at 0.80 or above. A downward transition must also cross a 0.05 hysteresis margin, which reduces rapid movement between adjacent states. The prototype records recommended actions such as logging, sandboxing, throttling, human review, or blocking. These are

simulated control decisions; AgentGuard is not connected to a live model or tool executor.

Many recent guardrails intervene at the boundary between an agent and its tools. ToolSafe, for example, screens individual steps before a tool is invoked [8]. A policy-enforcement design for Model Context Protocol (MCP) agents combines interception with information-flow control and audit logging [9]. AgentGuard is similar in where it seeks to intervene, but different in how it makes the decision: its present detector is lexical, and its events and control actions are simulated rather than drawn from live tool traces.

B. Existing Simulation Corpus

The analysis draws on artifacts preserved in the AgentGuard export tree; no additional experimental runs were generated for this paper. The canonical suite contains seven simulations of 200 events each. Four runs vary the configured attack rate, one tests AARS with equal factor weights, one repeats the main configuration with the same seed, and one repeats it with a second seed. A further ten runs use different fixed seeds at a 30% configured attack rate. Archived outputs are also available for lexical evasion, a static 64-prompt HarmBench-inspired set, post-hoc threshold analysis, and larger synthetic workloads.

Table 1. Canonical simulation suite.

Run	Configured attack rate	Seed	Events	Purpose
EXP-01	0%	42	200	benign workload baseline
EXP-02	10%	42	200	low attack-rate condition
EXP-03	30%	42	200	primary condition
EXP-04	50%	42	200	high attack-rate condition
EXP-05	30%	42	200	equal-weight AARS ablation
EXP-06	30%	42	200	deterministic repeat
EXP-07	30%	123	200	second-seed repeat

Attack events are sampled from a pool of 25 adversarial prompts and benign events from a separate pool of 25 prompts. The generator also changes several telemetry distributions according to the assigned class: attack events receive larger output-token ranges, added latency, more tool calls, and more RAG sources. This design is useful for exercising the complete pipeline. It also means that some non-security variables carry class information by construction, which must be considered when the results are interpreted.

C. Metrics and Interpretation

The archived reports include precision, recall, F1, accuracy, specificity, false-positive rate, ROC-AUC, PR-AUC, risk summaries, and bootstrap intervals. The ten-seed summary uses a run-level percentile bootstrap with 1,000 resamples and alpha 0.05. Two additional analyses are retained in the project: a threshold sweep from 0.01 to 0.99 based on Youden's J, and a severity-weighted calculation that assigns greater cost to system-prompt extraction, unsafe tool instructions, data leakage, and jailbreak events.

These measures describe the stored simulations; they are not estimates from an independent deployment dataset. In the

canonical composite evaluation, a positive reference event is derived from security_flags, while AARS uses SRMP output in its security component. The reference label and the prediction pathway are therefore linked. For that reason, this paper reports what the pipeline recorded within the simulation instead of presenting the values as external detector accuracy. The threshold sweep is also treated as a diagnostic exercise. A deployable threshold would need representative validation data, independent labels, and an explicit account of the relative cost of each error [10]-[13].

D. Literature Comparison Protocol

The related-work review uses peer-reviewed benchmark studies, guidance from standards and government bodies, and recent preprints where mature publications are not yet available. NIST's Generative AI Profile organizes risk management around GOVERN, MAP, MEASURE, and MANAGE [6]. Its draft guidance on benchmark evaluation also stresses clear objectives, documented data provenance, versioning, uncertainty, and qualified claims [14]. ProbGuard and AgentSentinel illustrate more recent attempts to estimate risk and intervene while agents are running [15], [16]. They provide useful design references, although their performance figures are not directly comparable with AgentGuard's synthetic results.

3. Results And Discussion

A. Canonical and Repeated-Seed Results

The main stored run, EXP-03, contains 200 events. The project's security-flag logic marks 48 of them as threats. Against those labels, the pipeline produced 38 true positives, 4 false positives, 10 false negatives, and 148 true negatives. Table II gives the corresponding composite measures. The F1 confidence interval for this run is 0.756-0.916. EXP-06 uses the same configuration and random seed, and reproduces the result exactly, as expected for a deterministic repeat.

Table 2. Primary synthetic-run results.

Metric	EXP-03 primary result	Interpretation
Precision	0.905	38 of 42 predicted positives matched the stored label
Recall	0.792	38 of 48 stored positives crossed the decision band
F1	0.844	harmonic balance of stored precision and recall
Specificity	0.974	148 of 152 stored negatives were below the decision band
False-positive rate	0.026	4 of 152 stored negatives were predicted positive
ROC-AUC	0.9996	not independently valid because of label coupling
Mean AARS score	0.362	composite risk in the primary synthetic workload

Results vary more clearly when the seed changes. Across ten seeds, F1 averages 0.791 with an SD of 0.051 and a run-level bootstrap 95% interval of 0.760-0.819. Individual values range from 0.674 to 0.844, showing that the fixed synthetic setup is still sensitive to which prompts and event attributes are sampled. Fig. 2 summarizes the canonical attack-rate runs, and Fig. 3 shows the ten-seed variation at 30%. Mean composite risk rises with the configured attack rate, an expected result because prompt selection and telemetry are both generated conditionally.

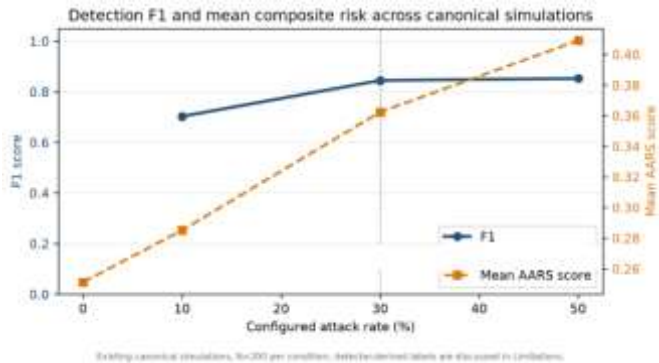


Figure 2. Existing canonical results by configured attack rate.

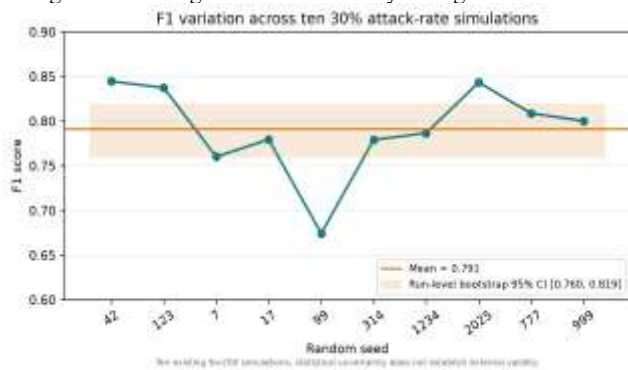


Figure 3. F1 Variation over ten stored random seeds.

The equal-weight run is more informative than the static-signature baselines because it changes one part of AARS while leaving the event set unchanged. Replacing the 0.25 security weight with one-sixth lowers F1 from 0.844 to 0.364 in the main condition. The pipeline is therefore highly sensitive to the security factor. This does not prove that 0.25 is an optimal weight, since SRMP flags also contribute to the construction of the reference labels.

B. Threshold and Severity Analyses

The archived Youden analysis compares the current threshold of 0.61 with a post-hoc maximum at 0.52. At 0.61, the stored values are $J=0.788$ and $F1=0.881$; at 0.52, they are $J=0.977$ and $F1=0.989$. Fig. 4 shows why the lower value looks preferable for this particular audit log. It is not sufficient evidence for changing a production threshold. The calculation uses mutable dashboard records, relies on labels derived from security flags, and has no independent validation/test split. In the archived severity-weight analysis, weighted F1 is 0.909, compared with an unweighted F1 of 0.837 on the same records. Weighting can be reasonable when missing a high-consequence event is more costly than missing a lower-severity event [12], [13]. The selected weights nevertheless express a policy choice. Domain owners should set them, and any reported result should retain the unweighted measures for comparison.

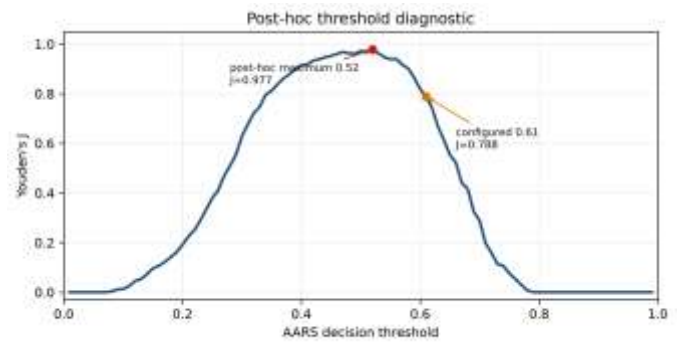


Figure 4. Stored post-hoc Youden's J diagnostic.

C. Robustness, Benchmark Transfer, and Baselines

Lexical evasion produces the clearest negative result. SRMP detects 0.76 of the original 25 attack prompts. Detection falls to 0.12 after synonym substitution and after the combined transformation; leet-speak, inserted whitespace, and Unicode homoglyphs produce a rate of 0.16. Case changes and added prefix or suffix text cause no recorded decline. This pattern is typical of literal substring rules: a small surface change can remove the trigger while leaving the harmful intention intact.

The internal HarmBench-inspired set contains 64 fixed prompts, of which 34 are labelled harmful and 30 benign. On this set, SRMP records precision 1.000, recall 0.529, and F1 0.692. Recall varies sharply by category, from 0.167 for system-prompt extraction to 1.000 for unsafe-tool prompts. Fig. 5 places these category results beside the evasion test. The set is a small internal collection inspired by HarmBench and AdvBench; it is not an official HarmBench evaluation.

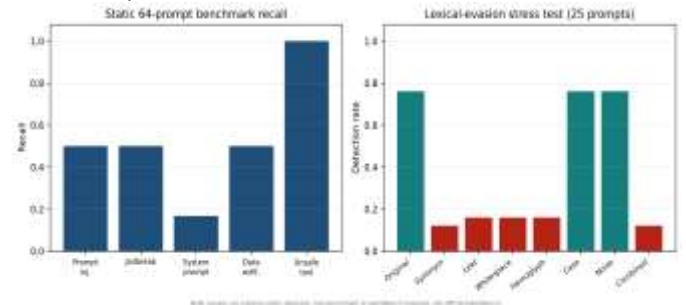


Figure 5. Internal benchmark and lexical-evasion results.

One archived comparator is named Pattern Proxy A, although it is a manual written pattern proxy and not a deployed Llama Guard model. On the same 64 prompts, the proxy records $F1=0.903$, the keyword filter 0.828, the hand-weighted TF-IDF-like baseline 0.342, and SRMP 0.692. These values are implementation checks, not model-comparison results. The static-signature and security-only baselines reach $F1=1.000$ in the canonical setting because they closely reproduce the logic used to define a positive label. Their perfect scores are therefore circular rather than evidence of superior detection.

Table 3. Internal baseline comparison.

System in stored comparison	F1	What it actually represents
SRMP AgentGuard	0.692	lexical rules on the internal static benchmark
"Pattern Proxy A"	0.903	manual pattern proxy, not Llama Guard
Keyword filter	0.828	manually selected keywords
"Weighted-Term Proxy"	0.342	manual weighted-term proxy, not trained TF-IDF

D. Large-Volume Simulation

The large-volume archive reports $F1=0.952$ for $N=2,000$ and $F1=0.966$ for $N=20,000$ at a configured attack rate of 50%. The reported $F1$ interval narrows from 0.059 at $N=200$ to 0.020 at $N=2,000$ and 0.006 at $N=20,000$.

The narrower interval reflects improved sampling precision as more synthetic events are drawn. It does not add external validity because the larger runs retain the same prompt pools, label construction, and conditional telemetry distributions.

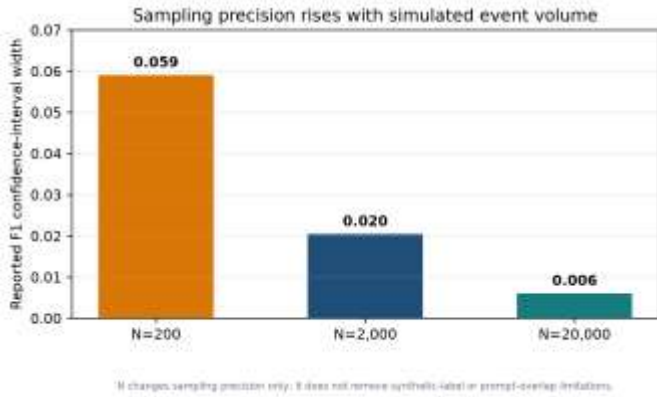


Figure 6. Reported confidence-interval width at increasing simulated event volumes.

E. Positioning Against Recent Work

Table IV sets out the basis for comparison with recent work. Prompt-injection and agent-security benchmarks show why benign-prompt testing alone is inadequate [2]-[5]. ToolSafe, TraceSafe, and related systems examine sequences of tool calls [8], [17], while ProbGuard and AgentSentinel use learned or probabilistic evidence from execution traces [15], [16]. AgentGuard pursues the same general goal of layered runtime monitoring, but its present scope is narrower: the first-stage detector is lexical, interventions are simulated, and evaluation remains internal.

Table 4. Conceptual positioning against recent work.

Work	Main setting	Relation to AgentGuard	Why numerical comparison is inappropriate
HarmBench [2]	model red teaming	benchmark-design reference	live target models and behavior-based judge
AgentHarm [3]	multi-step harmful agent tasks	motivates runtime controls	measures completed harmful tasks, not lexical rules
InjecAgent [4]	indirect injection with tools	close threat-model reference	evaluates real agents and tools
ASB [5]	agent attacks and defenses	coverage and utility-security reference	broad multi-model benchmark
ToolSafe [8]	learned tool-call guardrails	control-boundary analogue	learned guardrail and live trajectories
ProbGuard [15]	probabilistic trace monitoring	risk-monitoring analogue	learned DTMC and different environments
AgentSentinel [16]	computer-use defense	end-to-end interception analogue	computer-use benchmark and model-based audit

The comparison also makes the next design requirement clear. Current systems increasingly combine semantic models, trace reasoning, program analysis, access control, and enforceable

policy [8], [9], [15]-[19]. A lexical monitor remains useful as a transparent and inexpensive first check, but it cannot supply broad semantic coverage on its own. AgentGuard's evasion results provide direct evidence of that limitation.

F. Limitations and Threats to Validity

Several limitations affect the interpretation of these results. Most importantly, the canonical labels are not independent: SRMP produces security_flags, those flags help define the reference class, and AARS consumes the same security evidence. In addition, attack status changes simulated latency, tool counts, output tokens, and RAG-source counts, so non-security variables contain class information by design. The resource and latency fields are generated values rather than wall-clock measurements around a live LLM, tool, or network request. The prompt pools are small, English-only, and close to the vocabulary used in the rules. Finally, the evasion set does not independently establish semantic preservation, and neither the internal benchmark nor the synthetic traces use a locked external test set with expert-adjudicated labels.

These weaknesses are consistent with current evaluation guidance. NIST recommends explicit objectives, traceable data provenance, version control, uncertainty reporting, and careful qualification of benchmark claims [14]. OWASP and NIST likewise favour defence in depth, least privilege, monitoring, and human review [1], [6], [7]. A stronger evaluation would use independently labelled external traces, verify that adversarial transformations preserve meaning, select thresholds on held-out data, measure real tool-execution overhead, and include public task-complete benchmarks such as InjecAgent or AgentHarm.

4. Conclusion

AgentGuard shows how lexical monitoring, composite risk scoring, and adaptive control can be assembled in a small, inspectable runtime prototype. In the archived simulations, the main run records $F1=0.844$ and the ten-seed mean is 0.791. The equal-weight ablation also shows how strongly the decision depends on the security factor. At the same time, the evidence sets firm limits on the claim: lexical transformations sharply reduce detection, recall on the internal benchmark is modest, labels overlap with SRMP output, and latency is simulated rather than measured.

The present contribution is therefore an auditable research baseline, not a deployment-ready security claim. AgentGuard preserves event-level records and exposes each stage of its control logic, making it possible to investigate failure rather than treating the monitor as a black box. The next phase should introduce independently adjudicated labels, public agent benchmarks, semantic and trace-based monitors alongside the lexical rules, measured enforcement overhead, and an explicit analysis of how false interventions affect task completion. That evidence is needed before the prototype can be assessed as an operational runtime defence.

Acknowledgment

The author thanks Jyoti Vidhyapeeth Woman's University for its academic support. The study uses existing artifacts from the AgentGuard research prototype (<https://agent-guard-plan.replit.app/>) and involves neither human participants nor a new deployment experiment.

REFERENCES

- [1] OWASP GenAI Security Project, "LLM01:2025 Prompt Injection," OWASP Top 10 for LLM Applications, 2024.
- [2] M. Mazeika et al., "HarmBench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal," in Proc. ICML, 2024.
- [3] M. Andriushchenko et al., "AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents," in Proc. ICLR, 2025.
- [4] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents," in Findings of ACL, 2024.
- [5] H. Zhang et al., "Agent Security Bench: Formalizing and Benchmarking Attacks and Defenses in LLM-Based Agents," in Proc. ICLR, 2025.
- [6] C. Autio et al., Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024.
- [7] OWASP GenAI Security Project, State of Agentic AI Security and Governance, ver. 2.01, 2026.
- [8] Y. Mou et al., "ToolSafe: Enhancing Tool Invocation Safety of LLM-Based Agents via Proactive Step-Level Guardrail and Feedback," arXiv:2601.10156, 2026.
- [9] "Runtime Policy Enforcement for MCP-Based LLM Agents," Electronics, 2026.
- [10] A. H. Müller et al., "Classifier Calibration: A Survey on How to Assess and Improve Predicted Class Probabilities," Machine Learning, vol. 112, pp. 3211–3260, 2023.
- [11] M. H. Fluss, A. Faraggi, and B. Reiser, "Estimation of the Youden Index and Its Associated Cutoff Point," Biometrical Journal, vol. 47, no. 4, pp. 458–472, 2005.
- [12] V. S. Sheng, F. Provost, and P. G. Ipeirotis, "Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers," in Proc. KDD, 2008.
- [13] A. Mitrokotsa and C. Dimitrakakis, "Intrusion Detection Using Cost-Sensitive Classification," arXiv:0807.2043, 2008.
- [14] D. Keller et al., Practices for Automated Benchmark Evaluations of Language Models, NIST AI 800-2, Initial Public Draft, 2026.
- [15] Y. Wang, M. Poskitt, J. Wei, and L. Sun, "ProbGuard: Probabilistic Runtime Monitoring for LLM Agent Safety," arXiv:2508.00500, 2026.
- [16] Y. Hu et al., "AgentSentinel: An End-to-End and Real-Time Security Defense Framework for Computer-Use Agents," arXiv:2509.07764, 2025.
- [17] Y.-S. Chen et al., "TraceSafe: A Systematic Assessment of LLM Guardrails on Multi-Step Tool-Calling Trajectories," arXiv:2604.07223, 2026.
- [18] R. Betser et al., "AgenTRIM: Tool Risk Mitigation for Agentic AI," arXiv:2601.12449, 2026.
- [19] P. Wang and Y. Liu, "AgentArmor: Enforcing Program Analysis on Agent Runtime Trace to Defend Against Prompt Injection," arXiv:2508.01249, 2025.
- [20] P. Wang et al., "The Landscape of Prompt Injection Threats in LLM Agents: From Taxonomy to Analysis," arXiv:2602.10453, 2026.
- [21] "How Vulnerable Are AI Agents to Indirect Prompt Injections? Insights from a Large-Scale Public Competition," arXiv:2603.15714, 2026.
- [22] W. Zhang et al., "Your LLM Agent Can Leak Your Data: Data Exfiltration via Backdoored Tool Use," in Findings of ACL, 2026.
- [23] M. A. Ferrag et al., "From Prompt Injections to Protocol Exploits: Threats in LLM-Powered AI Agent Workflows," ICT Express, 2026.
- [24] B. Rombaut et al., "Watson: A Cognitive Observability Framework for the Reasoning of LLM-Powered Agents," arXiv:2411.03455, 2024.
- [25] S. Goyal et al., "LLMGuard: Guarding Against Unsafe LLM Behavior," in Proc. AAAI, 2024.
- [26] D. Dorn, A. Variengien, C.-R. Segerie, and V. Corruble, "BELLS: A Framework Towards Future Proof Benchmarks for the Evaluation of LLM Safeguards," arXiv:2406.01364, 2024.
- [27] Y. Zheng et al., "AgentSight: System-Level Observability for AI Agents Using eBPF," arXiv:2508.02736, 2025.
- [28] Infocomm Media Development Authority, Model AI Governance Framework for Agentic AI, ver. 1.5, 2026.