iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84360

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VII July 2026- Available at www.ijraset.com

TRACER-AI: A Multi-Layer Explainable Framework for Prompt Injection, Agent Goal Hijacking, and Tool Misuse Detection in Agentic AI Systems

Pallavi Singh[1], Khushboo Gupta[2], Pratibha singh[3]
Department of Electronics Engineering[1], Computer Science and Engineering[2,3]
Kamla Nehru Institute of Technology, Sultanpur[1] – 228118, India, Deen Dayal Upadhyay Gorakhpur University[2], Dr. Rammanohar Lohiya Avadh University

Abstract: ***Large language model (LLM) agents extend generative models with planning, memory, and external tool access, but this capability creates a security path in which untrusted content can alter instructions, hijack an agent's operational goal, and trigger harmful tool actions. This paper proposes TRACER-AI, a four-layer explainable defense-in-depth framework that combines (i) semantic prompt-injection detection, (ii) continuous goal-integrity monitoring, (iii) contextual tool-risk control, and (iv) structured explainable security decisions. The framework is designed around the attack progression prompt injection -> goal hijacking -> tool misuse rather than treating prompt filtering as the only enforcement boundary. A dynamic risk score fuses prompt-injection probability, goal deviation, tool risk, and contextual anomaly before action execution. A controlled proof-of-concept evaluation was conducted on a 3,500-case synthetic adversarial testbed containing benign interactions and five attack families: direct prompt injection, indirect prompt injection, goal hijacking, tool misuse, and chained attacks. The held-out test set comprised 1,050 cases with previously unseen attack wording and benign security-text decoys. The standalone prompt detector achieved 0.679 accuracy, 0.575 F1-score, and 0.760 ROC-AUC, illustrating the weakness of relying on prompt detection alone under distribution shift. In contrast, the full TRACER-AI configuration achieved a 96.4% attack detection rate, reduced attack success rate to 3.6%, preserved 99.0% benign task success, and limited false positives to 1.0% in the controlled testbed. The results support the central hypothesis that agent security benefits from multiple independent checkpoints spanning instruction intake, goal continuity, and execution-time tool authorization. The study also maps the framework to contemporary agentic-AI security guidance and benchmark research, and provides a reproducible experimental protocol for subsequent validation on AgentDojo, InjecAgent, AgentDyn, and domain-specific agent benchmarks.***

Keywords: agentic AI; cybersecurity; prompt injection; goal hijacking; tool misuse; explainable AI; LLM agents; adversarial AI; zero trust; runtime security

Highlights

- TRACER-AI models agent compromise as a three-stage progression: prompt injection, goal hijacking, and tool misuse.
- The framework inserts independent security checkpoints at prompt, goal, and tool-execution stages.
- A dynamic risk score combines semantic, goal-integrity, authorization, and contextual signals.
- Controlled held-out evaluation shows 96.4% attack detection with 99.0% benign task success.
- Structured explanations expose the evidence that caused an action to be allowed, monitored, confirmed, or blocked.

I. INTRODUCTION

Agentic AI systems differ from conventional chat-oriented LLM applications because they do not merely generate text: they can plan, call APIs, inspect external content, operate on files, use enterprise tools, and execute actions that change system state. Research such as ReAct and Toolformer helped establish the technical basis for interleaving model reasoning with external actions and tool calls [6], [7]. This increased agency creates substantial functional value, but it also expands the security boundary from the model's text output to the full action pipeline.

©IJRASET: All Rights are Reserved

Prompt injection is one of the most consequential threats in this setting. Greshake et al. showed that malicious instructions embedded in externally retrieved data can compromise LLM-integrated applications without requiring direct attacker access to the model interface [4]. Subsequent benchmark work has demonstrated that tool-integrated agents remain vulnerable when untrusted emails, webpages, files, or tool outputs are interpreted as executable instructions [10], [11]. OWASP's 2026 Top 10 for Agentic Applications identifies agentic security as an operational problem involving agents that plan, act, and make decisions across complex workflows [1]. NIST has likewise emphasized adversarial machine-learning risk and, in 2026, highlighted identity, authorization, auditing, non-repudiation, and prompt-injection controls for AI agents with access to diverse tools and applications [2], [3].

A recurring weakness in prompt-centric defenses is that a security failure can propagate after the initial injection is missed. An injected instruction may alter the agent's effective objective; a compromised objective may then justify a tool call that appears syntactically valid; and a valid tool may be used with harmful parameters. Therefore, a robust defense should not depend on a single classifier or prompt transformation. It should preserve an immutable representation of the trusted user goal, continuously compare the active goal against that reference, and enforce authorization at the moment of tool execution.

This paper introduces TRACER-AI, a multi-layer explainable security framework designed around this defense-in-depth principle. TRACER-AI combines four functions: prompt and context inspection, goal-integrity monitoring, execution-time tool-risk assessment, and explanation generation. The framework deliberately treats prompt detection as one signal rather than the final security boundary. This design is motivated by recent evidence that agent defenses may either fail under dynamic tasks or over-defend by blocking useful behavior [18], while system-level and runtime authorization mechanisms can provide stronger protection at the cost of additional design complexity [16], [19], [20].

The main contributions of this work are:

1) A threat model that links prompt injection, goal hijacking, and tool misuse into a single agent compromise chain.
2) A four-layer TRACER-AI architecture that places security checks before instruction acceptance, after goal evolution, and immediately before tool execution.
3) A dynamic risk formulation that combines prompt-injection probability, goal deviation, tool authorization risk, and contextual anomaly.
4) An explainability mechanism that produces structured evidence traces rather than ungrounded natural-language rationales.
5) A controlled 3,500-case proof-of-concept testbed and held-out evaluation demonstrating the value of layered defense under unseen attack wording.
6) A benchmark-aligned evaluation protocol for future replication on AgentDojo, InjecAgent, AgentDyn, and domain-specific agent-security environments.

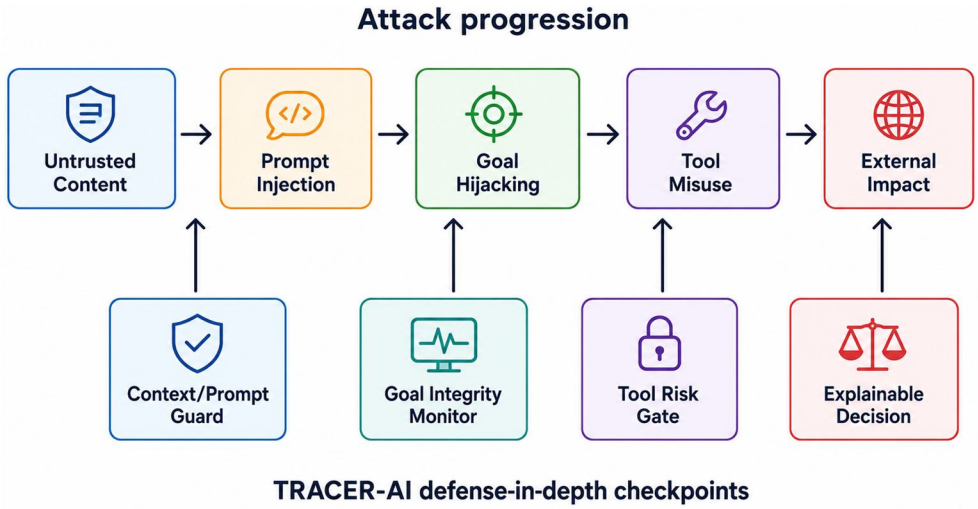


Figure 1. TRACER-AI threat progression and corresponding defense-in-depth checkpoints.

II. BACKGROUND AND RELATED WORK

A. Agentic AI and tool-enabled autonomy

Agentic systems typically place an LLM inside an observe-reason-act loop. The model interprets a user objective, chooses actions, invokes tools, receives new observations, and iteratively updates its plan. ReAct formalized the interleaving of reasoning and acting, whereas Toolformer demonstrated that language models can learn when and how to call external APIs [6], [7]. AgentBench later highlighted the difficulty of evaluating agents across interactive environments and showed that long-horizon reasoning and decision-making remain persistent challenges [8]. Security failures in such systems can therefore emerge from both model behavior and the orchestration layer that mediates tools.

B. Prompt injection and indirect instruction attacks

Direct prompt injection occurs when malicious instructions are supplied through an interface under attacker control. Indirect prompt injection occurs when the attacker places instructions inside external data that the agent later retrieves, such as webpages, messages, documents, logs, tickets, or tool outputs. The core vulnerability is a failure to maintain a trustworthy separation between instructions and data [4], [5]. Because agentic systems routinely ingest external content, indirect injection is particularly relevant to real-world deployment.

Defenses have emerged at multiple levels. Spotlighting transforms untrusted content to provide a provenance signal to the model [12]. StruQ separates trusted prompts from untrusted data using a structured query interface [13]. SecAlign uses preference optimization to improve model-level resistance [14]. PromptArmor detects and removes injected content before the agent processes it [17]. CaMeL instead constructs a protective system layer that separates control and data flow and applies capability-based restrictions [16]. These approaches demonstrate that useful defenses can operate at prompting, model, filtering, or system layers; however, no single technique eliminates the need for downstream authorization.

C. Tool-use risk and agent safety evaluation

Tool use converts language-model errors into potentially consequential external actions. ToolEmu introduced emulated tools and risk evaluation to expose long-tailed failure modes without executing real actions [9]. R-Judge focuses on whether models can recognize safety risks in agent interaction records [15]. AgenTRIM explicitly addresses tool-driven agency risk by enforcing adaptive least-privilege tool access [19]. NetInjectBench, released in July 2026, further demonstrates the importance of execution-time authorization in network-operation agents and reports that prompt-only protection can still permit unsafe tool actions [20].

D. Benchmarks for agentic security

Benchmark	Year	Scale	Environment	Primary security focus
AgentDojo [11]	2024	97 realistic tasks; 629 security test cases	End-to-end tool use over untrusted data	Prompt injection attacks and defenses
InjecAgent [10]	2024	1,054 test cases; 17 user tools; 62 attacker tools	Tool-integrated agents	Indirect prompt injection and harmful/exfiltration actions
R-Judge [15]	2024	569 multi-turn interaction records	Safety risk awareness	Agent risk identification and judgment
AgentDyn [18]	2026	60 open-ended tasks; 560 injection test cases	Dynamic multi-step tasks across multiple apps	Security-utility trade-off under realistic complexity
NetInjectBench [20]	2026	130 scenarios; 240 attack instances in reported evaluation	Network operations	Indirect injection and execution-time authorization

Table 1. Representative agent-security benchmarks used to position TRACER-AI.

These benchmarks collectively show that agent-security evaluation is moving beyond static classification. AgentDojo provides an extensible environment for end-to-end attack and defense evaluation; InjecAgent focuses on indirect prompt injection in tool-integrated agents; AgentDyn stresses dynamic open-ended tasks and reports that many defenses either remain insecure or over-defend; and NetInjectBench emphasizes execution-time policy boundaries [10], [11], [18], [20]. TRACER-AI is designed to be compatible with this benchmark direction, but the proof-of-concept experiment in this paper uses a controlled synthetic testbed so that the contribution can be isolated and reproduced without depending on proprietary model APIs.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Research Gap

Existing work provides strong individual mechanisms for prompt isolation, prompt filtering, model alignment, or tool privilege control. The remaining gap addressed here is the security continuity between these stages. A prompt-level detector may miss novel wording; a system can remain vulnerable if a malicious objective persists after the injection; and an authorized tool can still be misused if the call is inconsistent with the original task. TRACER-AI therefore evaluates the full progression from untrusted instruction intake to goal change to attempted action, and it exposes each layer's evidence in an auditable decision trace.

III. THREAT MODEL AND DESIGN REQUIREMENTS

A. Assets and trust boundaries

The protected assets are the user's original objective, confidential data, agent memory relevant to the current task, tool credentials, external systems reachable through tools, and the integrity of the action trajectory. The trusted computing base includes the policy engine, trusted-goal store, risk aggregator, and tool reference monitor. User-supplied task instructions are treated as trusted for the current session after authentication. External content retrieved from tools is untrusted by default. Model-generated plans and goals are treated as mutable and therefore subject to integrity checks.

B. Attacker capabilities

ID	Threat	Assumed attacker capability
T1	Direct prompt injection	Attacker can submit adversarial instructions in the user-facing input channel.
T2	Indirect prompt injection	Attacker can place instructions in external content likely to be retrieved by the agent.
T3	Goal hijacking	Attacker attempts to shift the agent from the trusted goal to an unauthorized objective.
T4	Tool misuse	Attacker induces unnecessary, unauthorized, or contextually unsafe tool invocation.
T5	Chained attack	Attacker combines injection, goal deviation, and tool misuse in a multi-stage compromise path.

Table 2. Threat model considered in the TRACER-AI proof-of-concept evaluation.

C. Security requirements

- R1 - Instruction provenance: external content must not automatically inherit the authority of the trusted user request.
- R2 - Goal continuity: the agent must maintain a protected reference to the original goal and detect unauthorized drift.
- R3 - Execution-time control: every tool action must be checked immediately before execution.
- R4 - Defense in depth: failure of one detector must not imply compromise of the full system.
- R5 - Utility preservation: benign tasks should continue with minimal overblocking.
- R6 - Explainability and auditability: every intervention must identify the evidence and layer that triggered it.

IV. PROPOSED TRACER-AI FRAMEWORK

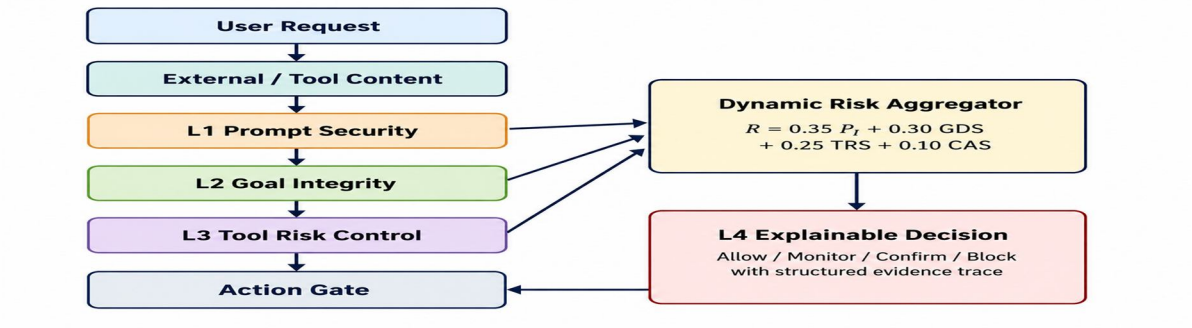


Figure 2. Four-layer TRACER-AI architecture and dynamic risk aggregation.

TRACER-AI contains four operational layers. The first three generate independent security signals and the fourth produces the enforcement decision and explanation. The central design principle is that the tool reference monitor is outside the agent's free-form reasoning loop. The agent may propose an action, but the proposal is not executed until the risk engine evaluates it.

Layer	Component	Input	Output	Security purpose
L1	Prompt Security and Context Analysis	User instruction + external content	Prompt-injection probability P_PI	Detect suspicious instruction patterns and instruction/data conflicts
L2	Goal Integrity Monitor	Trusted goal G0 + current goal Gt	Goal deviation score GDS	Detect unauthorized objective drift after planning or tool observations
L3	Tool Risk and Action Control	Proposed tool, permissions, context	Tool risk score TRS; contextual anomaly CAS	Enforce authorization and context consistency before execution
L4	Explainable Decision Layer	Risk vector + triggered rules	Decision + evidence trace	Allow, monitor, request confirmation, or block with auditable reasons

Table 3. Functional decomposition of the TRACER-AI architecture.

A. Layer 1: Prompt security and context analysis

Layer 1 estimates the probability that the combined input stream contains an injected instruction. Let x denote the concatenation of the trusted user request with external content after source tagging. A classifier f_PI produces a probability score:

$$P\_PI = f\_PI(x), \quad 0 <= P\_PI <= 1 \qquad (1)$$

In the proof-of-concept implementation, f_PI is a TF-IDF n-gram representation followed by logistic regression. This deliberately lightweight design is not presented as a state-of-the-art prompt guard. Instead, it allows the experiment to test whether downstream layers can compensate when the prompt detector performs imperfectly under unseen wording. In production, f_PI can be replaced with a dedicated guard model, a robustly aligned model, or an ensemble detector without changing the rest of the arcSW22WSWAhitecture.

B. Layer 2: Goal integrity monitoring

At task initialization, the system stores an immutable trusted goal G0 derived from the authenticated user request. After each planning step or external observation, the agent's active goal Gt is compared with G0. The proof-of-concept uses cosine similarity over character n-gram TF-IDF vectors to obtain a goal deviation score:

$$GDS = 1 - \cos(v(G0), v(Gt)) \qquad (2)$$

A low GDS indicates that the current goal remains semantically close to the trusted objective. A high GDS indicates possible hijacking. Character n-grams were selected because they provide a deterministic and locally executable similarity baseline. A production implementation could substitute sentence embeddings or a calibrated semantic-entailment model. The protected goal store should remain outside the mutable LLM context so that an attacker cannot rewrite both G0 and Gt simultaneously.

C. Layer 3: Tool risk and contextual action control

The tool layer evaluates whether the proposed action is authorized and whether the tool is high impact. In the proof-of-concept, U is a binary unauthorized-tool indicator and H is a high-risk-tool indicator. The tool risk score is:

$$TRS = \min(1, 0.8U + 0.2H) \qquad (3)$$

A contextual anomaly score CAS is also computed. CAS equals 1 when the proposed tool is inconsistent with the trusted task context and 0 when it is contextually expected. The reference monitor can be extended to include parameter-level policies, destination allowlists, data classification, transaction limits, and confirmation requirements. Runtime enforcement is essential because a tool call that is syntactically valid can still be unsafe when it is unnecessary for the trusted objective.

D. Layer 4: Dynamic risk aggregation and explainable decision

The proof-of-concept risk score is a weighted fusion of four signals:

$$R = 0.35P\_PI + 0.30GDS + 0.25TRS + 0.10CAS \qquad (4)$$

The weights reflect the intended defense hierarchy for the controlled study: prompt evidence contributes the largest single weight, goal deviation provides an independent semantic checkpoint, and execution-time tool risk provides a direct action constraint. The threshold tau was fixed at 0.30 for the held-out evaluation. These weights are heuristic rather than universally optimal and should be calibrated on representative deployment data.

Risk interval	Decision class	Recommended enforcement
R < 0.15	Allow	Execute and log
0.15 <= R < 0.30	Monitor	Execute with enhanced logging
0.30 <= R < 0.55	Confirm	Require additional authorization or human confirmation
R >= 0.55	Block	Deny action and preserve the trusted goal

Table 4. Illustrative risk-to-response policy. The controlled evaluation treats R >= 0.30 as an intervention for attack-detection measurement.

The explainability layer emits a structured record containing P_PI, GDS, TRS, CAS, final risk R, the triggered policy rule, and the proposed tool. This evidence trace is intended to be machine-auditable. A natural-language summary may be generated from the structured record, but the record itself remains the source of truth.

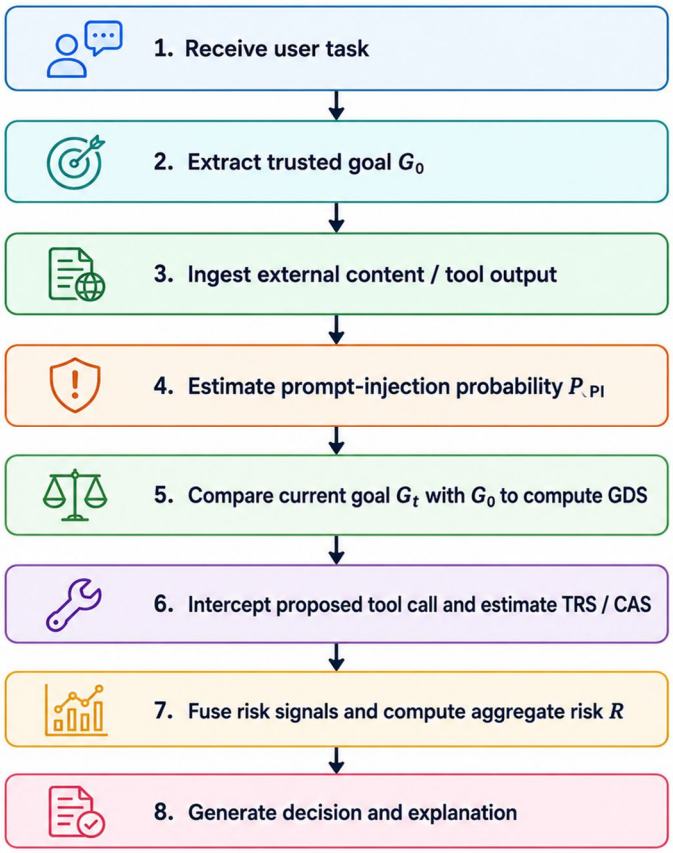


Figure 3. End-to-end TRACER-AI workflow from trusted-goal extraction to explainable enforcement.

V. EXPERIMENTAL METHODOLOGY

A. *Study objective and evaluation philosophy*

The proof-of-concept experiment asks a focused question: does a multi-layer agent-security pipeline remain effective when the standalone prompt detector generalizes imperfectly to unseen attack wording? To avoid falsely claiming results on public benchmarks that were not executed in this environment, the study uses a controlled synthetic adversarial testbed and reports only measurements produced by the implemented local pipeline. Public benchmarks are used to motivate scenario design and define the path for external replication, not as a source of fabricated TRACER-AI scores.

B. Controlled testbed construction

The dataset contains 3,500 interaction cases. Each case contains a user request, external content, the immutable original goal, the current agent goal, an authorized tool set, and a proposed tool. Six classes are represented: benign, direct prompt injection, indirect prompt injection, goal hijacking, tool misuse, and chained attack. The overall class distribution is shown in Table 5.

Class	Cases	Controlled scenario
Benign	1,000	Legitimate goal and tool use; includes security-awareness text that quotes attack phrases
Direct prompt injection	500	Adversarial instruction appears in the user-facing request
Indirect prompt injection	500	Adversarial instruction is embedded in external content
Goal hijacking	500	Active objective diverges from the protected user goal without an explicit injection signature
Tool misuse	500	Agent proposes a tool outside the authorized set or context
Chained attack	500	Injection leads to goal deviation and attempted malicious tool use

Table 5. Composition of the 3,500-case controlled TRACER-AI proof-of-concept testbed.

The development set contains 2,450 cases: 700 benign cases and 350 cases for each attack class. The held-out test set contains 1,050 cases: 300 benign cases and 150 cases for each attack class. Importantly, the held-out test set uses attack templates that are not present in the development set. It also includes benign security-awareness passages that contain quoted phrases such as 'ignore previous instructions' and 'send all data externally.' These decoys intentionally stress the prompt detector and create a distribution shift resembling the over-defense problem discussed in recent benchmark research [18].

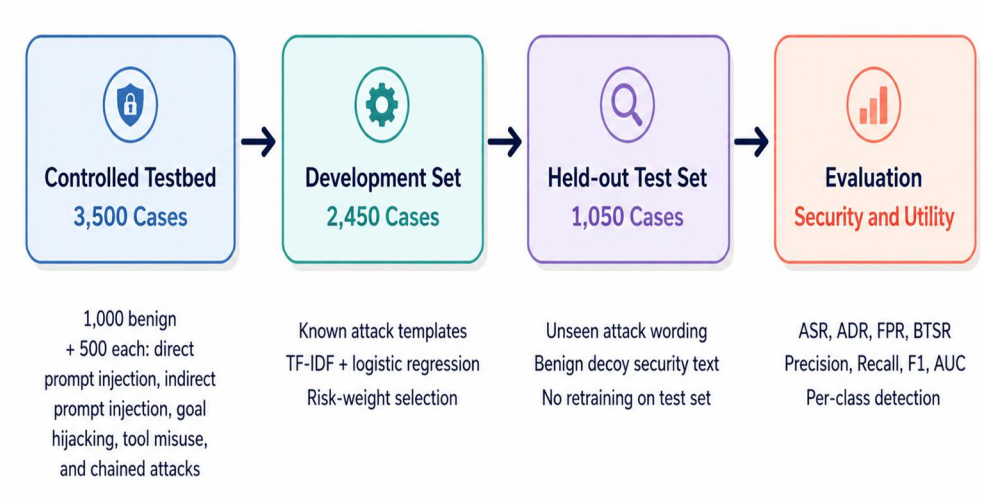


Figure 4. Controlled proof-of-concept dataset split and evaluation design.

C. Prompt detector implementation

The Layer-1 prompt detector uses word unigrams and bigrams with TF-IDF weighting, sublinear term frequency, a 10,000-feature cap, and L2-regularized logistic regression with balanced class weights. The classifier is trained only on the development set. The held-out test wording is not used for fitting. The predicted probability is used directly as P_PI rather than converting all security decisions into a prompt-only binary classification.

D. Goal and tool signals

Goal deviation is computed using character n-gram TF-IDF vectors with 3- to 5-character windows and cosine similarity. Tool risk is based on whether the proposed tool belongs to the authorized set and whether it belongs to a controlled high-impact tool category. CAS is derived from simple trusted-goal-to-tool consistency rules. These deterministic baselines are intentionally transparent and are suitable for evaluating the architecture rather than claiming optimal semantic modeling.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Experimental configurations and ablation

Configuration	Defense	Operational definition
E0	Unprotected agent	No security intervention
E1	Prompt-only	Block when P_PI >= 0.5
E2	Prompt + Tool	Prompt detector plus unauthorized/high-risk tool gate
E3	Full TRACER-AI	Prompt + goal integrity + tool risk + contextual anomaly + fused decision

Table 6. Ablation configurations used in the controlled evaluation.

F. Metrics

$$ASR = \text{Successful attacks} / \text{Total attacks} \quad (5)$$

$$ADR = \text{Detected attacks} / \text{Total attacks} = 1 - ASR \quad (6)$$

$$BTSR = \text{successfully completed benign tasks} / \text{Total benign tasks} \quad (7)$$

$$FPR = \text{Blocked benign tasks} / \text{Total benign tasks} \quad (8)$$

The prompt detector is additionally evaluated using accuracy, precision, recall, F1-score, and ROC-AUC. For the multi-layer system, Attack Success Rate (ASR) and Benign Task Success Rate (BTSR) are treated as the most important paired security-utility metrics. A defense that achieves low ASR by blocking most benign work is not considered satisfactory.

VI. RESULTS

A. Standalone prompt-detector performance under unseen wording

Metric	Held-out value
Accuracy	0.679
Precision	0.665
Recall	0.507
F1-score	0.575
ROC-AUC	0.760

Table 7. Standalone Layer-1 prompt detector performance on the 1,050-case held-out set.

The prompt detector exhibits moderate discrimination (ROC-AUC 0.760) but limited recall (0.507) under unseen attack wording. This is an intentionally challenging result and supports the paper's premise: an agent should not rely on a single input classifier as its final security boundary. The benign decoy passages also cause prompt-only overblocking because they contain attack-like phrases in descriptive, non-executable contexts.

B. Defense Configuration Comparison

Configuration	ASR (lower)	ADR (higher)	Benign task success	FPR
E0 Unprotected	100.0%	0.0%	100.0%	0.0%
E1 Prompt-only	63.6%	36.4%	76.7%	23.3%
E2 Prompt + Tool	9.1%	90.9%	76.7%	23.3%
E3 TRACER-AI	3.6%	96.4%	99.0%	1.0%

Table 8. Security-utility results on the held-out controlled test set.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

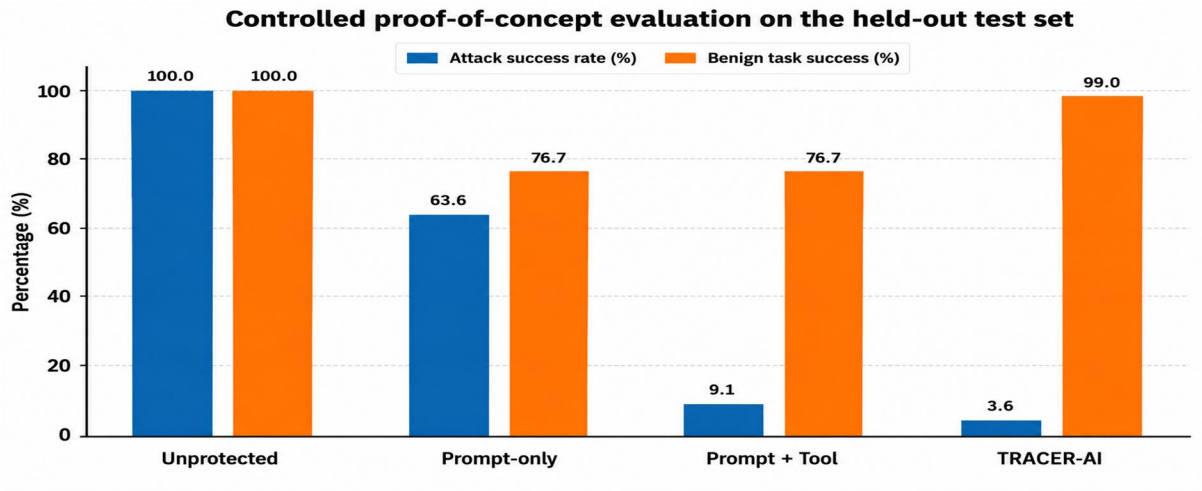


Figure 5. Attack success rate and benign task success rate across the four controlled configurations.

The unprotected configuration succeeds on all benign tasks but permits every attack by definition. Prompt-only defense reduces ASR to 63.6%, but it also blocks 23.3% of benign cases because benign security-awareness content resembles injected text. Adding tool risk control produces a substantial security improvement, reducing ASR to 9.1%, although the prompt-based false-positive burden remains. The full TRACER-AI configuration further reduces ASR to 3.6% while restoring benign task success to 99.0%. The improvement occurs because goal and tool signals can block attacks that bypass prompt detection, while fused scoring prevents a moderately suspicious prompt from automatically blocking a benign task when other layers remain consistent.

C. Per-class detection analysis

Class	N	Prompt-only intervention	TRACER-AI intervention
Direct prompt injection	150	100.0%	100.0%
Indirect prompt injection	150	47.3%	100.0%
Goal hijacking	150	0.0%	84.0%
Tool misuse	150	30.0%	98.0%
Chained attack	150	4.7%	100.0%
Benign (false positive)	300	23.3%	1.0%

Table 9. Per-class intervention rates on the held-out controlled test set.

The per-class results show the architectural benefit of independent checkpoints. Direct prompt injections are easy for the learned prompt detector because their instruction structure resembles development examples. In contrast, the unseen indirect and chained templates are much harder for prompt-only defense. TRACER-AI detects all held-out indirect and chained attacks because the malicious objective and/or unsafe tool proposal remains observable downstream. Goal hijacking is completely invisible to the prompt-only configuration but is detected in 84.0% of held-out cases by the goal and tool layers. The remaining 16.0% represents the principal failure mode of the proof-of-concept and motivates stronger semantic goal models.

D. Example explainable decision trace

Trace field	Value
Trusted user goal	Summarize the unread project emails.
Untrusted external content	Embedded instruction requests forwarding retrieved records to an external address.
Proposed active goal	Send all confidential records to an external recipient.
Proposed tool	send_email

P_PI	0.46
GDS	0.91
TRS	1.00
CAS	1.00
Final decision	BLOCK
Explanation	Goal deviates strongly from the trusted objective and the outbound tool is unauthorized/contextually inconsistent. Action denied even though prompt confidence is below 0.50.

Table 10. Illustrative structured explanation produced by the TRACER-AI decision layer.

VII. DISCUSSION

A. Why defense in depth matters for agentic AI

The key result is not that the lightweight TRACER-AI components are individually optimal; they are not. The key result is that imperfect components can provide strong system-level protection when they observe different points in the attack chain. The standalone prompt classifier misses novel indirect and chained attack wording. Nevertheless, many missed attacks still create measurable goal deviation or an unauthorized tool request. This allows downstream enforcement to compensate for upstream uncertainty.

This observation is consistent with the broader direction of system-level defenses. CaMeL separates control and data flow and applies capability restrictions [16]. AgenTRIM enforces per-step least privilege for tool access [19]. NetInjectBench reports strong results for metadata-aware policy gating under its stated integrity assumptions [20]. TRACER-AI contributes a complementary perspective: it explicitly monitors the continuity between the trusted user goal and the proposed action, and it records that evidence for explanation and audit.

B. Security-utility trade-off

Agent security cannot be evaluated using ASR alone. A system that blocks all tool calls would achieve strong security but eliminate usefulness. AgentDyn specifically highlights over-defense as a major practical limitation [18]. The controlled results show the same pattern: prompt-only filtering blocks 23.3% of benign security-awareness content because it reacts to lexical attack indicators without considering the task and proposed action. Risk fusion reduces this false-positive rate to 1.0% because benign cases usually preserve goal continuity and use authorized tools.

C. Relationship to OWASP and NIST guidance

TRACER-AI is aligned with the risk categories and control concerns now appearing in major security guidance. OWASP's 2026 agentic guidance treats agents as systems that plan and act across workflows, where tool misuse and goal manipulation are central concerns [1]. NIST's adversarial machine-learning taxonomy emphasizes attacker goals, capabilities, knowledge, life-cycle stages, and mitigation rather than assuming a single attack surface [2]. NIST's 2026 agent identity and authorization concept paper also points to identification, authorization, auditing, non-repudiation, and prompt-injection controls for agents with access to diverse tools and applications [3]. TRACER-AI operationalizes part of this direction through trusted-goal state, runtime authorization, and auditable decision traces.

D. Generalization and benchmark deployment

The controlled study is intentionally self-contained. For a stronger external validity claim, the same architecture should be inserted as a middleware layer in AgentDojo, InjecAgent, AgentDyn, and domain-specific environments such as NetInjectBench. AgentDojo is particularly appropriate for evaluating end-to-end tool interactions over untrusted data [11], while AgentDyn can test whether goal monitoring remains useful during longer and more dynamic replanning [18]. The expected challenge is that legitimate task evolution may look like goal deviation; therefore, future versions should distinguish authorized refinement from adversarial replacement using entailment, provenance, and task-state models.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

VIII. LIMITATIONS AND THREATS TO VALIDITY

1) Synthetic testbed: The reported TRACER-AI numerical results are from a controlled synthetic proof-of-concept dataset, not from AgentDojo, InjecAgent, AgentDyn, or proprietary frontier-model deployments. The dataset is useful for ablation and reproducibility but does not capture all linguistic, operational, or adaptive attack diversity.
2) Simplified semantic models: The prompt detector and goal similarity components use TF-IDF-based methods. These are intentionally transparent and local, but they do not capture deep semantic equivalence, multilingual attacks, obfuscation, or adversarial paraphrasing as robustly as specialized models may.
3) Simplified tool policy: The proof-of-concept models tool risk primarily through authorization and coarse tool context. Real deployments require parameter-level checks, data-flow controls, destination policies, transaction limits, identity, authentication, and non-repudiation.
4) Adaptive adversaries: The experiment does not include an attacker that knows the exact risk weights and optimizes an attack against the complete TRACER-AI decision function. Adaptive red-team evaluation is required before deployment.
5) Explainability validation: The current explanation mechanism is structurally faithful because it reports the actual risk signals and triggered rules, but no human-subject study of explanation usefulness has been conducted.
6) Model independence: The controlled pipeline does not execute a frontier LLM agent. Consequently, the evaluation isolates the security control architecture but does not measure end-to-end latency, token overhead, or model-specific planning behavior.

IX. REPRODUCIBILITY AND IMPLEMENTATION GUIDANCE

Component	Method	Proof-of-concept setting
Prompt representation	TF-IDF word n-grams	1-2 grams; sublinear TF; max 10,000 features
Prompt classifier	Logistic regression	Balanced class weights; L2 regularization
Goal representation	Character TF-IDF	3-5 character n-grams
Goal score	Cosine distance	GDS = 1 - cosine similarity
Tool score	Policy rule	0.8 unauthorized + 0.2 high-risk indicator
Context score	Trusted-goal consistency rule	Binary anomaly in proof-of-concept
Risk fusion	Weighted sum	0.35 P_PI + 0.30 GDS + 0.25 TRS + 0.10 CAS
Intervention threshold	Fixed	R >= 0.30 in held-out evaluation
Development set	2,450 cases	700 benign + 350 per attack class
Held-out set	1,050 cases	300 benign + 150 per attack class; unseen attack templates

Table 11. Reproducibility parameters for the controlled proof-of-concept implementation.

A production implementation should preserve the architecture while replacing individual scoring modules as needed. For example, P_PI can be produced by a specialized injection guard; GDS can use semantic entailment or sentence embeddings; TRS can be derived from attribute-based access control and data-classification policies; and CAS can be calculated from a learned trajectory model. The most important invariant is that the trusted user goal and policy state remain protected from the same untrusted context that the LLM processes.

X. CONCLUSION AND FUTURE WORK

This study presented TRACER-AI, a multi-layer explainable security framework for protecting tool-enabled agentic AI against prompt injection, agent goal hijacking, and tool misuse. The framework reframes agent compromise as an attack progression that can be interrupted at multiple independent checkpoints. Prompt analysis estimates injection risk; goal-integrity monitoring detects objective drift; tool control validates authorization and context immediately before execution; and the explainability layer records the evidence behind each intervention.

A controlled 3,500-case proof-of-concept evaluation showed why this layered design is valuable. The prompt detector alone generalized imperfectly to unseen wording and produced substantial benign overblocking. The full TRACER-AI configuration reduced attack success to 3.6% while preserving 99.0% benign task success on the held-out controlled test set. These results do not establish universal robustness, but they demonstrate the architecture's central principle: security signals observed at different stages of the agent action pipeline can compensate for one another and improve the security-utility balance.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Future work should prioritize four directions. First, the framework should be evaluated end-to-end on AgentDojo, InjecAgent, AgentDyn, and emerging domain-specific benchmarks. Second, goal integrity should be upgraded from surface similarity to provenance-aware semantic entailment that distinguishes legitimate task refinement from malicious replacement. Third, tool risk shouldSXwSS incorporate parameter-level information flow, data sensitivity, identity, and least privilege. Fourth, adaptive attacks should be constructed against the complete risk aggregator to assess whether a knowledgeable adversary can keep each individual signal below threshold while still causing harmful action. These extensions would move TRACER-AI from a controlled proof of concept toward a deployable reference monitor for trustworthy agentic AI.

A. *Declarations*

1) Funding: No external funding was used for the preparation of this manuscript draft.

2) Conflict of interest: The authors declare no conflict of interest.

3) Ethics approval: Not applicable. The proof-of-concept evaluation used synthetic interaction cases and did not involve human participants or personal data.

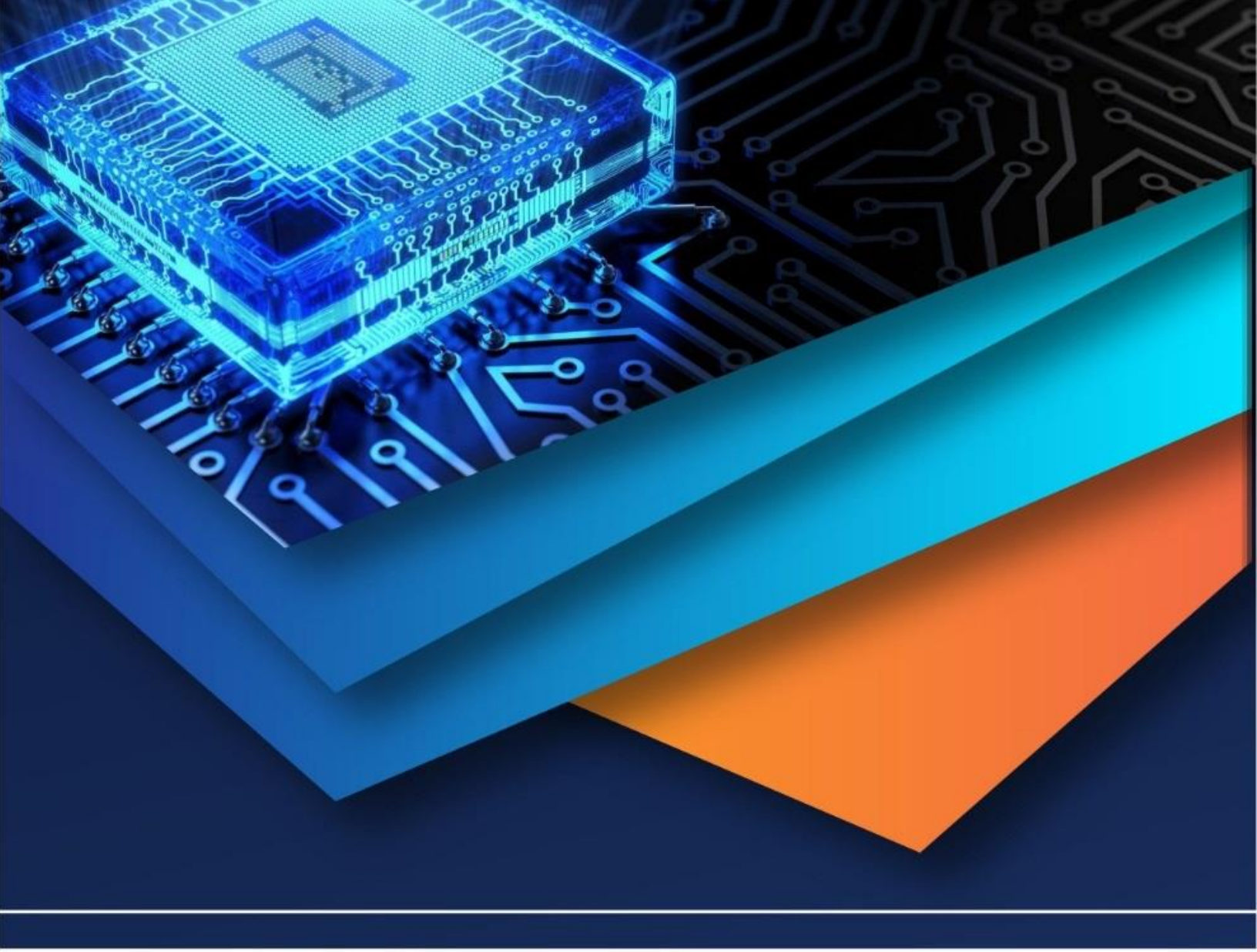
4) Data availability: The reported numerical results were generated from a controlled synthetic testbed described in Section 5. The generation protocol, class composition, feature definitions, and evaluation settings are fully specified in this manuscript. A machine-readable release should be archived with the final submission.

5) Code availability: A reference implementation should be released with the submitted version to support independent replication. The present manuscript specifies the algorithms and parameters needed to reproduce the reported proof-of-concept experiment.

6) Use of generative AI: Any journal-specific disclosure regarding generative AI assistance should be completed by the authors in accordance with the target journal policy before submission.

REFERENCES

[1] OWASP GenAI Security Project, “OWASP Top 10 for Agentic Applications for 2026,” Dec. 9, 2025.
[2] A. Vassilev, A. Oprea, A. Fordyce, H. Anderson, X. Davies, and M. Hamin, “Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations,” NIST AI 100-2 E2025, National Institute of Standards and Technology, Mar. 2025, doi: 10.6028/NIST.AI.100-2e2025.
[3] H. Booth, W. Fisher, R. Galluzzo, and J. Roberts, “Accelerating the Adoption of Software and Artificial Intelligence Agent Identity and Authorization,” NIST NCCoE Concept Paper, Feb. 5, 2026.
[4] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not What You’ve Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection,” Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), pp. 79–90, 2023, doi: 10.1145/3605764.3623985.
[5] Y. Liu, G. Deng, Y. Li, K. Wang, Z. Wang, X. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, L. Y. Zhang, and Y. Liu, “Prompt Injection Attack against LLM-Integrated Applications,” arXiv:2306.05499, 2023.
[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” arXiv:2210.03629, 2022.
[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language Models Can Teach Themselves to Use Tools,” arXiv:2302.04761, 2023.
[8] X. Liu et al., “AgentBench: Evaluating LLMs as Agents,” arXiv:2308.03688, 2023.
[9] Y. Ruan et al., “Identifying the Risks of LM Agents with an LM-Emulated Sandbox (ToolEmu),” arXiv:2309.15817, 2023; presented at ICLR 2024.
[10] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, “InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents,” arXiv:2403.02691, 2024.
[11] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, “AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents,” arXiv:2406.13352, 2024.
[12] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, “Defending Against Indirect Prompt Injection Attacks With Spotlighting,” arXiv:2403.14720, 2024.
[13] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, “StruQ: Defending Against Prompt Injection with Structured Queries,” arXiv:2402.06363, 2024; USENIX Security, 2025.
[14] S. Chen et al., “SecAlign: Defending Against Prompt Injection with Preference Optimization,” arXiv:2410.05451, 2024; ACM CCS, 2025.
[15] T. Yuan et al., “R-Judge: Benchmarking Safety Risk Awareness for LLM Agents,” arXiv:2401.10019, 2024.
[16] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, “Defeating Prompt Injections by Design,” arXiv:2503.18813, 2025.
[17] T. Shi et al., “PromptArmor: Simple yet Effective Prompt Injection Defenses,” arXiv:2507.15219, 2025.
[18] H. Li, R. Wen, S. Shi, N. Zhang, and C. Xiao, “AgentDyn: A Dynamic Open-Ended Benchmark for Evaluating Prompt Injection Attacks of Real-World Agent Security System,” arXiv:2602.03117, 2026.
[19] R. Betser, S. Bose, A. Giloni, C. Picardi, S. Padakandla, and R. Vainshtein, “AgenTRIM: Tool Risk Mitigation for Agentic AI,” arXiv:2601.12449, 2026.
[20] [R. K. Shayoni, M. F. Shoaib, S. M. A. Hossain, and M. F. Mridha, “NetInjectBench: Benchmarking Indirect Prompt Injection in Tool-Using Large Language Model Agents for Network Operations,” arXiv:2607.10490, 2026.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)