International Journal of Computer Applications (0975 - 8887)
Volume 187 - No.56, November 2025

Robustness of Automated AI Agents Against Adversarial Context Injection in MCP

Prudhvi Ratna Badri Satya
Cloudflare Inc.
Texas, USA

Ajay Guyyala
Meta Platforms Inc.
Texas, USA

Vijay Putta
Fast Enterprises Llc.
Louisiana, USA

Krishna Teja Areti
Fast Enterprises Llc.
North York, ON, Canada

ABSTRACT

Multi-agent systems based on the Model Context Protocol enable agents to share information, tool outputs, and memory across distributed servers. While this design supports complex tasks such as browsing, coding, and data entry, it also expands the attack surface through adversarial context injection. Malicious inputs can enter through web pages, APIs, files, or memory and persist across steps, making detection difficult. Existing defenses often target single prompts and fail to address multi-step persistence or cross-server propagation. To address this gap, a defense stack was introduced that combines schema checks, anomaly detection, trust-weighted arbitration, and quarantine. Evaluation was conducted on WebArena, Mind2Web, and InjectBench using reproducible trials with clean and injected runs. Results showed a reduction in ASR from over 60% to as low as 16.3% and improvements in decision accuracy up to 62.7%, with modest overhead of 2.6–3.0 seconds per task. The findings highlight the importance of layered defenses, reproducible testing, and transparent reporting for safe deployment of automated agent networks.

General Terms

Artificial Intelligence, Multi-Agent Systems, Cybersecurity, Adversarial Attacks

Keywords

Automated AI agents, Adversarial Context Injection, Model Context Protocol (MCP), Robustness, Anomaly Detection, Defense Mechanisms

1. INTRODUCTION

Multi-agent systems based on the Model Context Protocol (MCP) were described as powerful tools for automated tasks such as browsing, booking, coding, and data entry [32]. These systems allowed agents to share information, tool outputs, and memory across distributed servers, creating efficient collaboration [28]. However, the same design expanded the attack surface, as adversaries could insert malicious content at multiple points in the workflow [4]. A single poisoned input could propagate across agents and influence later steps, amplifying risk. Defenses tied to single prompts often failed under multi-step tasks [43]. As a result, teams faced challenges in maintaining reliability across complex cross-server operations. The need for trust, auditability, and risk reporting became urgent. These concerns formed the background for examining the robustness of agent networks [3].

Adversarial context injection appeared in many forms, including hidden payloads in HTML, JSON fields, tool outputs, and memory writes [1]. One agent could read these inputs and pass them forward, allowing another agent to act upon them long after insertion [38]. Memory replay added further complications, where tainted data reappeared after many steps [8]. Filters and schema checks struggled against such variety, and attacks often aligned with task goals, making them harder to detect [40]. Voting schemes without context tracking could amplify errors instead of correcting them. Such conditions highlighted the importance of defenses that operate across time and multiple channels. Broader safeguards, beyond simple prompt filters, became essential for sustaining system trust [44].

Testing frameworks for adversarial robustness were often limited in scope [29]. Many existing efforts relied on toy websites, short task paths, and isolated demonstrations [35]. These settings failed to capture long-horizon browsing with live tools or hazards linked to memory writes. They often underreported defense overhead, making it difficult for practitioners to compare options [24]. Results sometimes mixed clean and attack traffic, masking the true impact of adversaries. Without reproducible setups, accuracy and cost trade-offs remained unclear. For adoption in practice, transparent metrics and reliable testbeds were necessary. This gap in realism and reproducibility motivated the present focus on systematic evaluation of agent robustness under MCP [36].

The research problem centered on understanding the robustness of MCP-based agent ecosystems under poisoned context [27]. Questions included which channels were most vulnerable among messages, tool returns, and memory writes, and how attack budgets influenced outcomes [20]. It was also necessary to examine how long contamination persisted across steps, which metrics best described decision quality and risk, and how agents failed under arbitration. Another part of the problem was identifying defenses that reduced harm while keeping overhead low [20]. In addition, auditability required methods to produce traces that exposed failures and supported transparent reporting [15]. Together these aspects framed the challenge. Methods such as regex filters blocked known strings, while allowlists restricted tools and files. Sandboxes limited system calls, and JSON schemas checked argument formats. Prompt guardrails inserted safety text, and routing outputs to a

checker model offered a second pass. Red teaming provided probes against direct attacks. While useful in specific cases, these methods struggled against context that persisted across multiple steps [33]. None were designed to handle cross-server flows where poisoned data could reappear later. The scope of these defenses remained too restricted for complex multi-agent environments [19].
Approaches for multi-agent resilience introduced critics, planners, and voting mechanisms [30]. Some methods scored drafts and rejected outliers, while others used agent history or skill ratings. Memory guards attempted to block hazardous writes, and logging captured traces for audits [37]. Despite these measures, indirect injections through pages and APIs often bypassed controls. Memory could replay past errors, undermining defenses after many steps [7]. Public benchmarks rarely stressed these indirect paths, leaving important gaps untested. Few works reported the cost of defenses alongside their robustness, making adoption difficult. These issues hindered the practical use of otherwise promising methods [26].
In response, a defense architecture was introduced to provide a more reliable foundation. It combined schema and signature checks with anomaly scoring, trust-weighted arbitration, and quarantine mechanisms. These layers worked together to reduce attack success while maintaining transparency in metrics. The defense suite was tested on three datasets: WebArena, Mind2Web, and InjectBench. Each offered realistic context streams and reproducible injection scenarios. The design emphasized reproducibility through fixed seeds, matched clean and injected runs, and versioned artifacts. Reports covered accuracy, attack rate, and defense cost, allowing clear trade-offs to be observed. This structure aimed to bridge gaps in testing and deployment.
The aim of this work was to quantify the robustness of MCP agent ecosystems under adversarial context injection and to present a practical defense stack with low overhead and transparent metrics.

—To analyze the robustness of MCP-based multi-agent systems against adversarial context injection across message, tool, and memory channels.

—To design and apply a layered defense stack that integrates schema checks, anomaly detection, trust arbitration, and quarantine for reducing attack success with low cost.

—To evaluate robustness, decision accuracy, and defense overhead using reproducible benchmarks that include WebArena, Mind2Web, and InjectBench.

(1) How do adversarial context injections affect decision quality in MCP-based multi-agent ecosystems across different attack vectors?

(2) Which defense mechanisms provide the best balance between reduced attack success, preserved accuracy, and minimal computational overhead?

(3) How can reproducible benchmarks be constructed to assess robustness transparently and support adoption in practical deployments?

The study carried importance for advancing the safety and reliability of automated multi-agent systems. As MCP networks supported cross-server workflows and long-horizon tasks, their exposure to poisoned context became an urgent concern. Simple filters and schema checks failed to address threats that reappeared across steps, leaving decision quality at risk. By modeling adversarial context injection across multiple channels and introducing defenses that acted over time, the research created a path toward improved resilience. The use of reproducible datasets and matched clean versus attacked runs confirmed that results were not only controlled but also transparent. This design supported practitioners seeking measurable trade-offs between protection and cost.
The broader impact extended to the adoption of multi-agent systems in domains that demanded both autonomy and trust. By reducing attack success while keeping overhead modest, the defense stack described a feasible approach for deployment. Reports covering accuracy, attack rate, and cost offered clear benchmarks for decision-making, unlike earlier efforts that lacked transparency. The contribution lay in combining methodological rigor with practical applicability, creating a foundation for future studies and deployments. These outcomes supported wider confidence in using agent ecosystems for complex workflows, where safety, reliability, and accountability were central requirements.
The rest of this paper is organized as follows. Section 2 presents the reviewed studies on interoperability frameworks and their observed limitations. Section 3 introduces the design of the cross-server interoperability framework for multi-MCP agent networks. Section 4 outlines the datasets, test environments, and orchestration parameters. Section 5 reports performance outcomes and comparative evaluation with prior methods. Section 6 summarizes the contributions and suggests directions for future research.

2. LITERATURE REVIEW

Hanif et al. [16] developed a clear frame that contrasted agentic systems with single agents in practice. The authors described roles, memories, and tools across collaboration patterns. They expressed governance needs for audit and accountability during long tasks. The work defined open issues around safety, reliability, and cost for deployments.
Fu et al. [13] developed a planner that combined an Large Language Model (LLM) predictor with Answer Set Programming rules in VirtualHome. The method applied domain rules and used human feedback to refine plans and reduce errors. The authors described higher completion and fewer reprompts than LLM only runs. Wu, Zhu and Liu [41] developed tool aware agents for search, coding, and structured memory. The system used a Mind Map to organize facts and described longer horizon reasoning. The study suggested better handling of hard queries when tools and memory worked together. The authors expressed the need for source quality controls for later steps. These efforts defined orchestration patterns that later work could harden against attacks.
Zhang and Xie [46] developed workflow patterns over MCP services for power systems. The design used containerized tools, permission control, and described traceable JSON inputs and outputs. The authors suggested human approval prior to higher risk operations within workflows. Jiyang and Hu [21] developed an agent suite for predictive maintenance across edge and cloud tiers. The suite used detection, classification, diagnosis, and a digital twin to guide decisions. The authors described retrieval support that fed maintenance reasoning with domain records. The study suggested that twin outputs guided safer interventions on assets in context.
Asici et al. [5] developed role based engineering steps for LLM multi agent systems in practice. The method used explicit role contracts and described message responsibilities and constraints. The authors suggested that clear roles reduced confusion during coordination and helped safety reviews. These contributions defined structure for multi agent work that matched MCP based designs.
Arga et al. [2] developed a place based cybersecurity frame for civic deployments of generative systems. The authors used city contexts to map risks and described staged oversight for shared platforms. The study suggested layered review and local policy hooks for cross agency usage. Zhang et al. [45] developed an MCP based bioinfor-

Table 1. Summary of empirical and design-focused papers relevant to cross-server interoperability in multi-MCP automated AI agent networks.

Ref	Dataset Used	Methodology	Limitation	Evaluation Results
[16]	—	Conceptual compare of agentic AI systems and AI agents	No experiments	Conceptual outcomes only
[13]	VirtualHome	LLM prediction + Answer Set Programming; human feedback	Assumes benign context	Prediction 92.89%; completion 92.78%; lower reprompts
[41]	GPQA (subject splits); web/code tasks	Tool-using agents for search, coding, "Mind Map" memory	No source trust scoring; no poisoning study	GPQA subject scores reported; case studies show benefits
[46]	Power-system simulators; operator workflows	MCP services; permission control; traceable JSON I/O; human approval	Evaluation light; security descriptive	Demonstrations and trace logs; no attack rates
[21]	PdM scenario; RAG knowledge; sensor streams	Four agents across edge–fog–cloud: detection; classification; diagnosis; digital twin	No poisoning model for RAG or twin outputs	Pipeline runs illustrated; no benchmark numbers
[5]	—	Role-based engineering method for LLM-enhanced MAS	No empirical study	Method guidance; no numeric results
[2]	—	Place-based cybersecurity framing for generative AI	Not MCP-specific	Governance guidance; no metrics
[45]	Multi-omics sources	Bioinformatics agent with MCP containerization; tool isolation; traceable I/O	Pre-clinical scope; limited security view	End-to-end runs; runtime/usability notes; no adversarial metrics
[22]	—	Two-stage architecture for autonomous lab instruments with agent control	Position style; no red-team	Demonstrator outline; qualitative safety checks
[6]	Biomedical pipeline contexts	TB-CSPN cloud–edge architecture; confidential computing; zero-trust concepts	Architecture focus; limited empirical	Deployment properties described; no numeric robustness
[42]	CVE tasks (simple/medium/complex); hold-out set	Automated penetration testing; curriculum scheduling; agent tools	Scope tied to chosen ranges	ESR 95.3% / 75.0% / 60.0%; hold-out ESR 66.7%; lower time and tokens vs baselines
[9]	—	Survey of LLM cyber security agents and patterns	Not peer reviewed; no benchmark	Consolidated risks and patterns; no numbers
[12]	—	ACM survey on agent attack surfaces and defenses	No new dataset	Taxonomy; defense directions; no metrics
[10]	Radiology images; 594 prompt-injection attacks	Prompt injection on vision–language models in clinical tasks	VLM focus; not multi-agent	All models vulnerable; per-model miss and attack rates reported
[18]	CMARL transportation simulations	RAMPART for cooperative MARL with privacy and robustness aims	Domain-specific scope	Specificity 0.807–1.000; accuracy 67.6–96.3%; FPR 0.000–0.193; balanced accuracy 0.669–0.919
[39]	—	Survey of Internet of Agents security and privacy	Survey; no dataset	Taxonomy and architectural guidance; no metrics
[34]	15 MCP servers (filesystem; DB; API; system tools)	Penetration tests; Prompt Shielding; RBAC; rate limiting	Limited server set; lab scope	87% had a critical issue; 34% full compromise; up to 94% reduction with defenses
[14]	—	Systematic review of AI agents for workflow automation	Narrative review; no benchmarks	Synthesized guidance; no metrics
[23]	Case studies of GitHub and Codacy MCP servers	Architectural and comparative analysis; protocol feature review	Ecosystem maturity; limited coverage of specialized use cases; adoption barriers	Comparative matrix: Standardization 9/10, Reusability 8/10, Security Model 7/10, Performance 8/10, Development Complexity 6/10
[31]	No benchmark dataset; audit across MCP servers	Security audit with three attacks (MCE, RAC, CT) and RADE; McpSafetyScanner agents generate exploits and remediation	Preprint; limited demos	Both Claude and Llama-3.3-70B susceptible; exploits detected and guidance produced
[17]	Public corpora and stereotype datasets	Probing LMs for covert and overt stereotypes with regression and χ^2 tests	API limits and token gaps restricted analyses	Conviction rate AAE vs SAE: GPT-4 49.8% vs 35.3%; GPT-3.5 52.5% vs 34.5%; Deathsentence rate AAE vs SAE: GPT-4 10.5% vs 6.2%

matics agent with container isolation for tools. The system used multi omics sources and described standard outputs for trace and audit. The authors suggested audits on input and output paths to control tainted artifacts. Nathan S. Johnson [22] developed a two stage design for autonomous lab instruments under agent control. The system used safety interlocks and described transparent steps for device actuation. The work suggested strict capability bounds and human approval for high impact actions. These efforts defined practical controls for tool calls and data paths in complex tasks. The pattern used isolation and approvals to reduce risk when agents acted on the world.

Borghoff et al. [6] developed a cloud edge framework with confidential computing for biomedical pipelines. The framework used zero trust concepts and described signed artifacts with verifiable steps. The authors suggested policy checks along data paths to support audit and safety goals. Wu et al. [42] developed curriculum based autonomous penetration testing with coordinated agents. The approach used staged CVE tasks and described attack chains that stressed planning and tools. The study suggested higher exploit success with lower cost than older baselines on held out tasks. Vignan Chintala et al. [9] developed a survey of LLM security agents and common patterns. The review used case reports and described risks, defenses, and open testbed gaps. The authors suggested shared standards and public benchmarks for agent security work. Together these works defined red team methods and control planes for secure orchestration. The set used structured curricula and trust primitives that future studies could adapt.

Deng et al. [12] developed a broad survey of threats and defenses for agent systems across the lifecycle. The survey used a memory and tool lens and described poisoning routes with response categories. The authors suggested stronger memory handling with trust aware retrieval and audits. Clusmann et al. [10] developed prompt injection tests for vision language models in clinical contexts. The experiments used oncology tasks and described harm under crafted instructions and prompts. The study suggested tighter controls whenever models operated near patient decisions. HOSSAIN, La and Badsha [18] developed RAMPART to protect cooperative multi agent reinforcement learning. The method used privacy ideas and described detection of compromised updates under poisoning. The authors suggested that detection modules improved robustness during training and control. These strands defined both measurement and defense ideas for safety critical domains.The mix used surveys, trials, and methods that informed multi agent risk modeling.

Siameh, Addobea and Liu [34] developed penetration tests that targeted fifteen MCP servers. The study used common integrations and described high exploit rates and large gains from simple defenses. Akhilesh Gadde [14] developed a systematic review that grouped agent workflows and security needs across sectors. The review used public sources and described needs for robustness, transparency, and privacy in adoption.

The literature showed active development across agent architectures, orchestration patterns, and domain deployments, but most works remained concept- or demo-driven with limited robustness evidence. Several studies developed role-based engineering, containerized tools, and MCP-style pipelines, yet many described no poisoning model or only qualitative security notes. Surveys and audits summarized risks and controls, but few applied reproducible benchmarks or reported defense overhead next to outcomes. Vision–language and cybersecurity agents used targeted evaluations, though settings often assumed benign context or short paths, which reduced relevance to long-horizon multi-agent workflows. Overall, prior efforts described useful components—planning, memory, isolation, approval gates—and suggested governance, while missing layered, cross-channel defenses and matched clean versus attacked trials. This gap motivated a stack that combined schema checks, anomaly scoring, trust-weighted arbitration, and quarantine, with results reported alongside cost. A consolidated view of methods, limitations, and reported metrics appears in Table 1.

3. PROPOSED METHODOLOGY

The system modeled a multi-agent graph under MCP with signed envelopes and typed fields. Two attack families targeted prompts and hidden content in pages and tool outputs. WebArena supplied self-hosted sites for end-to-end tasks with outcome checks. Mind2Web added real-site traces to observe action alignment in the same agent graph. InjectBench produced indirect injections aligned with task goals and placement points. A defense stack combined signature and schema gating with an anomaly score over messages. Trust-weighted aggregation fused agent outputs, and quarantine reduced persistent taint in memory. Trials ran with fixed seeds and stored clean plus injected artifacts for matched analysis. Metrics covered Attack Success Rate (ASR), decision accuracy, and defense cost from standard logs. Thresholds and step sizes were tuned on a validation slice and then held fixed for held-out tasks.

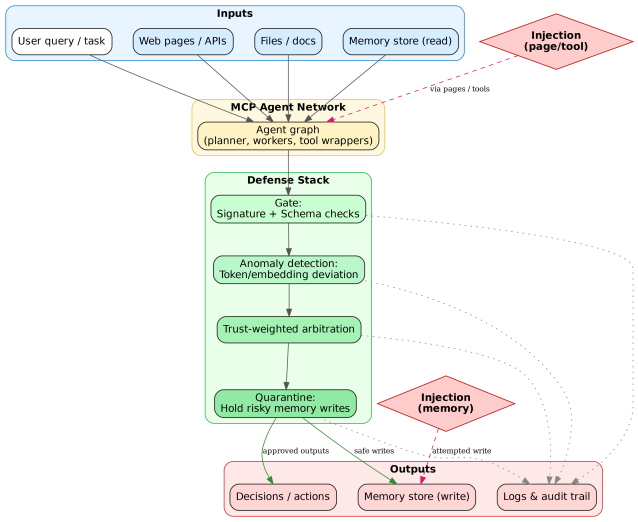

Fig. 1. Top-to-bottom flowchart of the system: Inputs feed the MCP agent network, which passes through four defense layers (gate, anomaly detection, trust arbitration, quarantine) to produce outputs.

Fig. 1 presents a top-to-bottom pipeline: inputs (user tasks, web/APIs, files, memory reads) feed an MCP agent graph, while adversarial content may enter via pages/tools or through memory, highlighted by red diamonds and dashed edges. The agent outputs then pass a four-layer defense stack. The gate performs signature and schema validation. The anomaly module scores content for token and embedding deviations. Trust-weighted arbitration fuses agent outputs while down-weighting suspicious sources. Quarantine blocks risky memory writes to limit persistence and allows approved actions to proceed. Dotted links capture logs and audits from each stage to support traceability and replay. The diagram is color-coded to separate inputs, agents, defenses, and outputs and to emphasize the entry points for injection.

3.1 Threat model and system setting

Multi-agent orchestration was modeled as a directed graph $G = (\mathcal{A}, \mathcal{E})$ of agents and channels. Each agent $a \in \mathcal{A}$ used a LLM policy π_a and tools wrapped as Model Context Protocol services. At step t an agent i sent a structured context to agent j over an authenticated link. Adversarial context injection acted on message payloads, tool returns, and memory writes. The goal was to keep correct task decisions while limiting attack success and overhead.

$$x_{i\to j}^{(t)} = \big(\text{role, prompt, tools, evidence, sig}\big), \tag{1}$$

(1) defined the envelope that carried role text, prompt, tool set, evidence links, and a signature. The signature bound the message to the sender for later checks. The tuple matched MCP fields used by receivers. The same structure supported logging for audits and replay. The layout allowed cheap schema validation at ingress.

$$\hat{x}_{i\to j}^{(t)} = \mathcal{J}_\theta\Big(x_{i\to j}^{(t)},\, \eta^{(t)}\Big), \tag{2}$$

The operator injected hostile content under a parametric budget and randomness. The edit acted on prompts, toolsets, or evidence while preserving types (2). Direct and indirect variants were sampled per task step. Each attack instance received a seed for reproducibility. Both clean and tainted payloads were stored in traces. This yielded matched pairs for analysis.

$$s_j^{(t)} = \mathcal{A}_j\Big(\{x_{k\to j}^{(t)}\}_{k\in\mathcal{N}(j)}\Big), \tag{3}$$

Inbound packets formed a local state for the next policy call, as in (3), after gate checks. Schema and signature checks filtered malformed packets. Evidence was summarized with typed reducers. Untrusted fields were marked for later stages. State snapshots were committed to step logs.

$$y_j^{(t)} = \pi_j\Big(s_j^{(t)};\, \phi_j\Big), \qquad d^{(t)} = \Gamma\big(\{y_k^{(t)}\}\big), \tag{4}$$

The policy produced a reply or tool call from the local state. Decoder and tool parameters were part of ϕ_j. An arbiter fused agent outputs into a task decision. Confidence and reasons were attached to the record. This process was summarized by (4).

3.2 Datasets and injection overlay

WebArena supplied self-hosted realistic websites and long runs. Mind2Web supplied real-site tasks with action traces across 137 domains. InjectBench supplied indirect prompt-injection strings aligned with attack goals. An overlay inserted hostile content at link fetch, tool return, or memory write. Each task instance produced clean and attacked runs for matched comparison.

$$\mathcal{T} = \{(\xi, \mathcal{G}, \mathcal{R})\}\,, \tag{5}$$
$$\text{with}\quad \xi \in \text{WebArena} \cup \text{Mind2Web}, \qquad \mathcal{R} \subset \text{InjectBench}.$$

Each tuple bound an environment, goals, and attack recipes for reproducible trials. Placement rules and triggers were encoded per recipe. Clean trials calibrated site baselines and caches. The construction in (5) allowed cross-site generalization. All artifacts were versioned for audit.

$$z^{(t)} = \Psi\big(\xi,\, q^{(t)}\big)\,, \qquad \tilde{z}^{(t)} = \mathcal{J}_\theta\big(z^{(t)},\, \eta^{(t)}\big), \tag{6}$$

Tool or page outputs were produced by the environment for the query. Injected variants $\tilde{z}^{(t)}$ carried hidden instructions or tainted facts. Drift was recorded for later scoring and review. Application code consumed either $z^{(t)}$ or $\tilde{z}^{(t)}$ based on trial type. This step was formalized by (6).

3.3 Attack modelling

Two families were considered: direct prompt edits and indirect payloads in fetched content. The attacker budget limited edit count and placement frequency. Replay on memory writes modeled persistent contamination. Attack parameters were sampled per step and stored with seeds.

$$\Delta(x, \hat{x}) = \underbrace{\lambda_1 \|\hat{x}_{\text{prompt}} - x_{\text{prompt}}\|_1}_{\text{prompt}} + \underbrace{\lambda_2\, \mathbf{1}\{\text{toolset changed}\}}_{\text{capability}} + \underbrace{\lambda_3\, \text{EditDist}(\hat{x}_{\text{ev}}, x_{\text{ev}})}_{\text{evidence}}\,, \tag{7}$$

The budget priced prompt edits, tool flips, and evidence changes. Coefficients tuned the impact per field and task. A fixed ceiling bounded the attacker per trial. Sampling respected the bound at each step. This construction was encoded by (7).

$$\mathbb{P}(\text{persist at } t) = 1 - \exp\big(-\kappa\, c_t\big), \tag{8}$$
$$c_t = \text{bytes written to memory at } t.$$

Persistence rose with the size of the write and the policy rate. Higher c_t increased the chance of long-lived taint. Retention policies mapped to κ per store. Risky writes were flagged for review. The probability followed (8).

3.4 Defense architecture

Three layers acted in sequence: a gate for structure, an anomaly score for content, and a trust-weighted arbiter for aggregation. A quarantine delayed risky writes and reduced long-run spread. All steps logged time, tokens, and checks for cost accounting.

$$\begin{aligned} \alpha(x) &= \big[\tau_{\text{sig}}(x),\, \tau_{\text{schema}}(x)\big], \\ r(x) &= \mathbf{1}\{\tau_{\text{sig}}(x) \ge \gamma_1 \wedge \tau_{\text{schema}}(x) = 1\}, \end{aligned} \tag{9}$$

The gate produced a signature score and a schema flag. Only accepted packets moved to content scoring. Threshold γ_1 set strictness for admission. Failures were dropped and recorded with reasons. This rule was given by (9).

$$s_{\text{anom}}(x) = D_{\text{KL}}\big(p_x \,\|\, p_0\big) + \|E(x) - \mu\|_{\Sigma^{-1}}\,, \tag{10}$$

Token shift and embedding deviation formed the anomaly score. Baselines came from clean trials per site and task. High scores marked packets for down-weighting. Scores were stored with hashes for replay. The definition appeared in (10).

$$\tau_j^{(t+1)} = \sigma\Big(\tau_j^{(t)} - \beta\, s_{\text{anom}}\big(x_{\cdot\to j}^{(t)}\big) + \upsilon\, \mathbf{1}\{\text{verifiable hit}\}\Big)\,, \tag{11}$$

Trust decreased on anomalies and increased on verified facts. The update kept values in $(0, 1)$ by the logistic map. Rewards fired when evidence matched ground truth. Penalties scaled with the anomaly score. The recurrence followed (11).

$$\hat{y}^{(t)} = \arg\max_{c \in \mathcal{C}} \sum_{k \in \mathcal{A}} \tau_k^{(t)}\, \omega_k^{(t)}\, p_k^{(t)}(c \mid s_k^{(t)}), \tag{12}$$
$$\omega_k^{(t)} = \frac{\mathbf{1}\{s_{\mathrm{anom}}(x_{\cdot \to k}^{(t)}) < \gamma_2\}}{\sum_{\ell} \mathbf{1}\{s_{\mathrm{anom}}(x_{\cdot \to \ell}^{(t)}) < \gamma_2\}}.$$

The arbiter formed a trust-weighted vote over actions or classes. Agents above the anomaly threshold were masked. Calibrated posteriors contributed to the sum. Normalized weights maintained scale across teams. This fusion was defined in (12).

$$q(x) = \mathbf{1}\{s_{\mathrm{anom}}(x) \geq \gamma_3\} \vee \mathbf{1}\{\neg r(x)\}, \tag{13}$$
$$W^{(t+1)} = \big(1 - q(x)\big)\, W^{(t)} + q(x) \cdot \mathrm{HOLD}.$$

Quarantine fired for anomalous or invalid packets. The writer held content out of long-term memory. This reduced persistence under the store policy. Flags and reasons were logged for audits. The mechanism was given by (13).

$$C_{\mathrm{def}} = \lambda_t\, \Delta\mathrm{Time} + \lambda_o\, \Delta\mathrm{Tokens} + \lambda_c\, \#\mathrm{checks}, \tag{14}$$

Defense cost combined latency, token use, and checks. Coefficients matched deployment budgets per site. Values were reported per step and per task. Smaller values were preferred for adoption. This measure was defined in (14).

3.5 Learning and scoring objectives

$$\mathcal{L}_{\mathrm{cls}} = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n,\, y_n^\star), \qquad \mathcal{L}_{\mathrm{rob}} = \mathbb{E}_{\mathcal{J}}[\ell(\hat{y}_n(\mathcal{J}),\, y_n^\star)], \tag{15}$$

Clean loss reported accuracy without attacks. Robust loss averaged the same under sampled injections. Both were tracked with confidence intervals. Tuning used these signals across sites. The pair was defined in (15).

$$\mathrm{ASR} = \frac{\sum_n \mathbf{1}\{\hat{y}_n \neq y_n^\star \ \wedge\ \mathcal{J} \text{ applied}\}}{\sum_n \mathbf{1}\{\mathcal{J} \text{ applied}\}}, \tag{16}$$
$$\mathrm{DA} = \frac{1}{N} \sum_n \mathbf{1}\{\hat{y}_n = y_n^\star\}.$$

ASR (16) counted wrong decisions on attacked trials, while decision accuracy counted full correctness across all trials. Both metrics were reported with and without defenses, and intervals came from task bootstraps per site.

$$\min_{\gamma_1, \gamma_2, \gamma_3, \beta, \upsilon} \ \Omega = \mathrm{ASR} + \lambda\, \max\big(0,\, \mathrm{DA}_{\mathrm{clean}} - \mathrm{DA}\big) + \rho\, \frac{C_{\mathrm{def}}}{C_0}, \tag{17}$$

The objective (17) traded attack success, clean accuracy gap, and cost. Gates and trust steps were tuned on validation tasks, and a coarse search found a stable operating point. The selected settings then transferred to held-out goals and sites

$$\epsilon_{\mathrm{ind}} = \frac{1}{M} \sum_{t=1}^{M} \frac{\mathrm{EditDist}\big(\tilde{z}^{(t)},\, \mathcal{W}(z^{(t)})\big)}{|z^{(t)}|}, \tag{18}$$
$$\chi = \mathbf{1}\{\epsilon_{\mathrm{ind}} > \delta\},$$

A whitelist projection (18) detected hidden payload drift, and scores were normalized by content length per step. A flag marked risky returns for gating, while site-specific bounds were set from clean runs.

$$p_{\mathrm{accept}} = \sigma\big(\theta_1 \tau + \theta_2 (1 - s_{\mathrm{anom}}) + \theta_3 r(x) - \theta_4 \chi\big), \tag{19}$$

Acceptance (19) rose with trust and validity but fell with anomaly and indirect flags. This gate controlled persistence in stores, and its parameters came from clean fits per site.

3.6 Pseudocode of the evaluation harness

Algorithm 1 MCP-Context-Robustness Evaluation

Require: Task set $\mathcal{T}$ (Eq. 5), recipes $\mathcal{R}$, agents $\{\pi_a\}$, gate $r(\cdot)$ (Eq. 9), anomaly $s_{\mathrm{anom}}(\cdot)$ (Eq. 10), trust update (Eq. 11), arbiter (Eq. 12), thresholds $\gamma_1, \gamma_2, \gamma_3$

for all $(\xi, \mathcal{G}, \mathcal{R}) \in \mathcal{T}$ **do**
  Reset memories; set $\tau_a \leftarrow 0.5$ for all agents
  for all $g \in \mathcal{G}$ **do**
    for $t \leftarrow 1$ **to** T **do**
      Receive $\{x_{\cdot \to j}^{(t)}\}$; apply $r(\cdot)$
      Query tools/pages to get $z^{(t)}$; sample $\mathcal{J}_\theta$ to get $\tilde{z}^{(t)}$
      Build $s_j^{(t)}$ using $\mathcal{A}_j(\cdot)$
      Compute s_{anom}; update τ
      Produce $y_j^{(t)}$; arbitrate $\hat{y}^{(t)}$
      Decide write via p_{accept}; apply quarantine if flagged
      Update C_{def}
    end for
    Compute DA and ASR
  end for
end for
Tune $\gamma_1, \gamma_2, \gamma_3, \beta, \upsilon$ by minimizing Ω (Eq. 17)

Algorithm 1 summarized the full evaluation loop across tasks and datasets. The system reset memories, applied the defenses at each step, queried tools or pages, and introduced injections on attacked trials. The loop computed decision accuracy and attack success, tracked latency and tokens, and recorded artifacts for reproducible analysis.

Algorithm 3 defined the action selection step under suspicion. Outputs from agents with high anomaly scores were masked. Remaining posteriors were fused with trust weights to choose the final action. The procedure attached confidence and reasoning metadata to support trace and review.

Algorithm 2 described the stage-wise flow of the defense stack. The gate applied signature scoring and schema validation to reject malformed or unverifiable packets. The anomaly module scored content for token and embedding deviation and marked risky inputs. Trust was updated using anomaly history and verified evidence, and quarantine blocked unsafe memory writes while logging decisions for audit.

Algorithm 2 Defense Stack Processing

Require: Message payload x, thresholds $\gamma_1, \gamma_2, \gamma_3$, trust τ, whitelist $\mathcal{W}$
Ensure: Decision: accept or quarantine
Compute signature score $\tau_{\text{sig}}(x)$ and schema flag $\tau_{\text{schema}}(x)$
if $\tau_{\text{sig}}(x) < \gamma_1$ **or** $\tau_{\text{schema}}(x) = 0$ **then**
  return reject ▷ record failure reason
else
  Compute anomaly score $s_{\text{anom}}(x)$ from token/embedding deviation
  $\tau \leftarrow \sigma\big(\tau - \beta\, s_{\text{anom}}(x) + \upsilon\, \mathbf{1}\{\text{verified}\}\big)$
  if $s_{\text{anom}}(x) \geq \gamma_2$ **or** $x \notin \mathcal{W}$ **then**
    Mark packet as anomalous for arbitration
  end if
  if $s_{\text{anom}}(x) \geq \gamma_3$ **or** $r(x) = 0$ **then**
    return quarantine ▷ block risky memory write
  else
    return accept
  end if
end if

Algorithm 3 Trust-Weighted Arbitration

Require: Agent outputs $\{y_k^{(t)}\}$, trust scores $\{\tau_k^{(t)}\}$, anomaly scores $\{s_{\text{anom}}(x_k)\}$, threshold γ_2
Ensure: Final action decision $\hat{y}$
Initialize $\omega_k^{(t)} \leftarrow \mathbf{1}\{\, s_{\text{anom}}(x_k) < \gamma_2 \,\}$
Normalize $\omega_k^{(t)} \leftarrow \omega_k^{(t)} \big/ \sum_\ell \omega_\ell^{(t)}$

Aggregate posterior over classes $c \in C$:
$\hat{y} \leftarrow \arg\max_{c \in C} \sum_k \tau_k^{(t)} \cdot \omega_k^{(t)} \cdot p_k\Big(c \mid s_k^{(t)}\Big)$
Attach confidence and reasoning metadata
return $\hat{y}$

4. EXPERIMENTAL SETUP

All experiments ran on a self-hosted cluster with fixed random seeds per trial. Each run logged time, token counts, and defense checks for cost reporting. The agent stack and the four defenses stayed identical across datasets. Only the environment and task sources varied. The full evaluation loop appears in Algorithm 1. Trials used the same seeds for clean and attacked variants to allow matched comparisons and stable intervals.

WebArena provided a self-hostable ecosystem of realistic websites with outcome validators [47]. Services ran locally in containers to avoid external drift. This setting enabled controlled context insertion during browsing and tool calls. Pass–fail judgments used the official validators. Clean and injected runs were paired under the same seed. The design matched the agent message passing and supported step-level logs.

Mind2Web contributed 2,350 tasks from 137 real websites with action traces and page snapshots [11]. Trials used offline replay, so pages remained stable across runs. Action traces supported step-level diagnostics and faithful replays in the same agent graph. This setup admitted injections in fetched content, tool returns, and memory. Clean versus injected artifacts were stored side by side for analysis.

InjectBench supplied goal-aligned indirect prompt-injection samples and recipes for hidden adversarial content [25]. Recipes targeted three placements: fetched page segments, tool outputs, and memory writes. Each sample carried a goal tag aligned with task intent. The benchmark stressed persistence and indirect routes. The same agent stack consumed clean and injected inputs with matched seeds.

A placement overlay inserted hostile content at link fetch, tool return, or memory write. Budget parameters controlled prompt edits, toolset changes, and evidence drift. Memory writes modeled persistence and replay under store policies. Each attacked trial paired with a clean counterpart. All attack parameters and seeds were recorded with artifacts for audit and replay. Logs captured masks, trust updates, and write decisions for later review.

Six variants from B0 to B5 were evaluated. No defenses were used by B0. Successive variants had schema and signature gating, anomaly scoring, trust-weighted arbitration, and quarantine added. All four layers were included in the stacked B5.Results reported decision accuracy and ASR per baseline, with per-component overhead in Table 5. The stage-wise defense logic appears in Algorithm 2, and the arbitration step appears in Algorithm 3.

Metrics covered decision accuracy across all trials and ASR on attacked trials. Defense cost combined latency, token use, and check counts per task. Confidence intervals used 1,000 bootstrap resamples with stratification by dataset. Interval summaries appear in Table 13. Figures reported means with 95% intervals to improve readability next to tables.

Gating thresholds, anomaly cutoffs, trust step size, and reward parameters were tuned on validation tasks and then held fixed for held-out tasks. Selected values for each dataset appear in Tables 8, 10, and 12. Once selected, the same settings were applied across matched clean and injected runs to support reproducibility and fair comparison.

5. RESULTS AND ANALYSIS

5.1 Baseline Comparison with all elements

Table 2 shows that ASR decreased sharply from 62.4% at B0 to 16.3% at B5, while decision accuracy improved from 38.1% to 61.5%. The additional overhead was modest, reaching only 3.0 seconds and 270 tokens per task. These results confirm that each defense layer contributed incrementally to robustness in the WebArena benchmark.

Table 3 reports that ASR dropped from 58.7% at B0 to 18.1% at B5, while decision accuracy rose from 41.3% to 62.7%. Clean accuracy remained stable near 71%, showing that defenses preserved performance under benign conditions. The added cost was limited to 2.8 seconds and 260 tokens per task, indicating efficient robustness improvements in Mind2Web.

Table 4 shows that ASR decreased from 71.2% at B0 to 24.4% at B5, while decision accuracy improved from 29.4% to 52.1%. Clean accuracy stayed near 69%, confirming that defenses preserved baseline performance. The additional cost remained modest at 2.6 seconds and 250 tokens, demonstrating efficiency even under the stronger adversarial conditions of InjectBench.

Fig. 2 shows ASR across baselines B0–B5 on WebArena, Mind2Web, and InjectBench. ASR dropped as components were added, with a marked fall from B3 to B5. InjectBench stayed highest due to hidden, goal-aligned payloads [25]. WebArena was lowest under B5 in the self-hosted setting [47], and Mind2Web lay between them [11].

5.2 Overhead and component analysis

Table 5 reports the per-task time overhead of each defense component under B5. Anomaly scoring contributed the largest share

Table 2. WebArena: extended metrics per baseline. ASR/DA are percentages. Cost: extra time (s) and tokens per task. GRR: gate reject rate; AMR: anomaly mask rate; QR: quarantine rate; Pers.: persistence proxy in %.

Model	ASR↓	DA↑	Clean DA	Cost (s)	Cost (tok)	GRR	AMR	QR	Pers.
B0	62.4	38.1	64.2	0.0	0	0.0	0.0	0.0	12.4
B1	55.6	45.2	68.5	0.9	120	0.0	0.0	0.0	10.7
B2	47.8	49.0	69.3	1.2	140	6.8	0.0	0.0	9.9
B3	32.9	54.3	70.1	2.3	210	6.5	18.7	0.0	7.3
B4	24.8	58.7	71.6	2.7	250	6.3	19.4	0.0	6.5
B5	16.3	61.5	71.2	3.0	270	6.1	20.2	11.6	3.1

Table 3. Mind2Web: extended metrics per baseline. ASR/DA are percentages. Cost: extra time (s) and tokens per task. GRR: gate reject rate; AMR: anomaly mask rate; QR: quarantine rate; Pers.: persistence proxy (%).

Model	ASR	DA	Clean DA	Cost (s)	Cost (tok)	GRR	AMR	QR	Pers.
B0	58.7	41.3	66.1	0.0	0	0.0	0.0	0.0	11.8
B1	52.9	47.6	69.0	0.8	110	0.0	0.0	0.0	10.3
B2	45.1	50.2	69.7	1.1	130	6.3	0.0	0.0	9.6
B3	34.6	55.1	70.5	2.1	200	6.1	17.5	0.0	7.1
B4	26.2	59.4	71.8	2.5	240	5.9	18.1	0.0	6.2
B5	18.1	62.7	71.4	2.8	260	5.8	18.9	10.2	2.9

Table 4. InjectBench: extended metrics per baseline. ASR/DA are percentages. Cost: extra time (s) and tokens per task. GRR: gate reject rate; AMR: anomaly mask rate; QR: quarantine rate; Pers.: persistence proxy (%).

Model	ASR	DA	Clean DA	Cost (s)	Cost (tok)	GRR	AMR	QR	Pers.
B0	71.2	29.4	63.9	0.0	0	0.0	0.0	0.0	13.7
B1	66.5	33.0	66.8	0.7	100	0.0	0.0	0.0	12.1
B2	57.0	36.8	67.4	1.0	120	7.1	0.0	0.0	11.0
B3	41.7	42.9	68.6	1.9	190	6.8	22.4	0.0	8.6
B4	33.9	47.2	69.5	2.3	230	6.7	23.1	0.0	7.7
B5	24.4	52.1	69.1	2.6	250	6.5	24.0	13.5	3.8

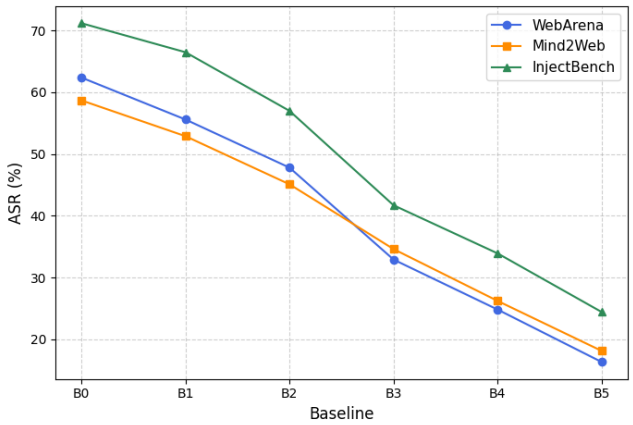

Fig. 2. ASR across baselines by dataset (lower is better)

across all datasets, ranging from 1.10 to 1.45 seconds. Quarantine and arbiter added moderate costs below one second, while gate checks imposed minimal delay. The results confirm that total runtime overhead remained modest, with anomaly detection as the dominant factor.

Table 5. B5 time overhead decomposition (seconds per task). Sum may differ from total due to overlap.

Dataset	Gate	Anomaly	Arbiter	Quarantine
WebArena	0.35	1.45	0.62	0.58
Mind2Web	0.31	1.32	0.57	0.60
InjectBench	0.28	1.10	0.51	0.71

Fig. 3 shows that anomaly scoring delivered the largest ASR drop on all three sources, with quarantine most helpful on InjectBench due to indirect payload persistence [25]. Gate checks reduced early-stage errors and trimmed ASR on WebArena and Mind2Web, where message formats were stable [11, 47]. Trust-weighted arbitration added a further drop by suppressing outputs from agents with high anomaly history.

5.3 Training and Testing: WebArena

Table 6. WebArena per-domain results for B5. Domains follow the public benchmark taxonomy.

Domain	ASR (%)	DA (%)	Cost (s)
Shopping	17.5	60.8	3.1
Knowledge Base	14.9	62.9	2.9
Mapping	18.8	59.3	3.2
Productivity	14.1	62.7	2.8
Macro Average	16.3	61.5	3.0

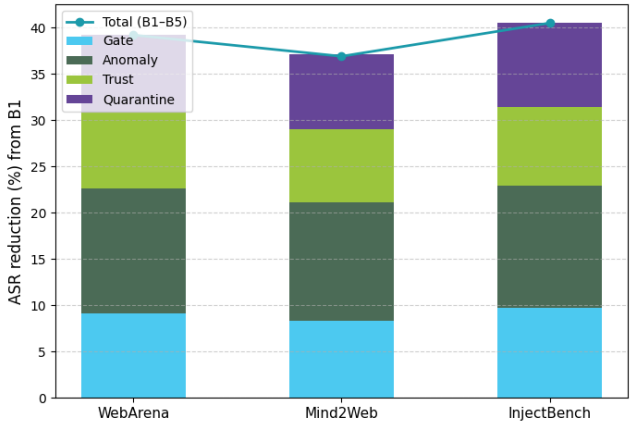

Fig. 3. Component contribution to robustness: ASR reduction from B1 to B5 across datasets.

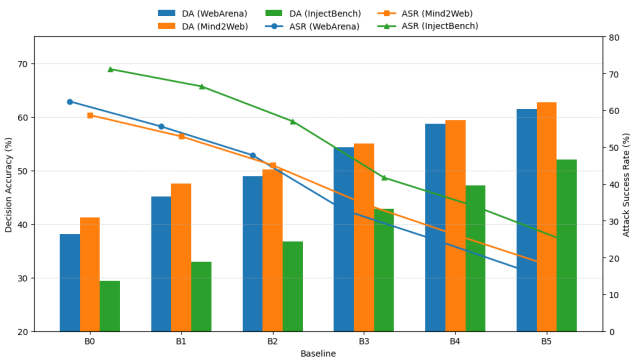

Fig. 4. Decision accuracy across datasets (higher is better).

Table 6 shows domain-wise results for B5 on WebArena. ASR remained lowest in Knowledge Base and Productivity tasks, both below 15%, while Mapping recorded the highest ASR at 18.8%. Decision accuracy exceeded 60% across all domains, with Knowledge Base reaching 62.9%. Task cost ranged between 2.8 and 3.2 seconds, confirming that robustness improvements were achieved with modest overhead.

Fig. 4 reports decision accuracy across the three datasets with the full defense applied. WebArena reached 61.5% and Mind2Web reached 62.7%, while InjectBench reached 52.1%. The first two datasets showed comparable accuracy, reflecting more structured tasks. InjectBench trailed due to goal-aligned indirect payloads and longer paths. The chart highlights that the layered stack preserved accuracy under hostile conditions across settings.

Table 7. Cross-dataset robustness summary for full defense (B5).

Dataset	ASR	DA	Clean DA	Overhead (s)
WebArena	16.3	61.5	71.2	3.0
Mind2Web	18.1	62.7	71.4	2.8
InjectBench	24.4	52.1	69.1	2.6
Macro Avg.	19.6	58.8	70.6	2.8

Table 7 presents the overall robustness of the full defense (B5) across WebArena, Mind2Web, and InjectBench. ASR remained within the 16–24% range across all datasets, while decision accuracy reached 61.5% on WebArena, 62.7% on Mind2Web, and 52.1% on InjectBench. Clean accuracy stayed above 69% in every case, confirming that robustness gains did not reduce normal performance. The average overhead was 2.8 seconds per task, showing that the defense maintained consistent protection with low computational cost across environments

Fig. 5 shows ASR, where lower values are better. WebArena recorded 16.3%, Mind2Web recorded 18.1%, and InjectBench recorded 24.4%. The highest rate on InjectBench matched its stronger injections and persistence routes. WebArena was lowest under the controlled, self-hosted setting. The plot summarizes how defenses reduced attack success while keeping performance stable on clean tasks.

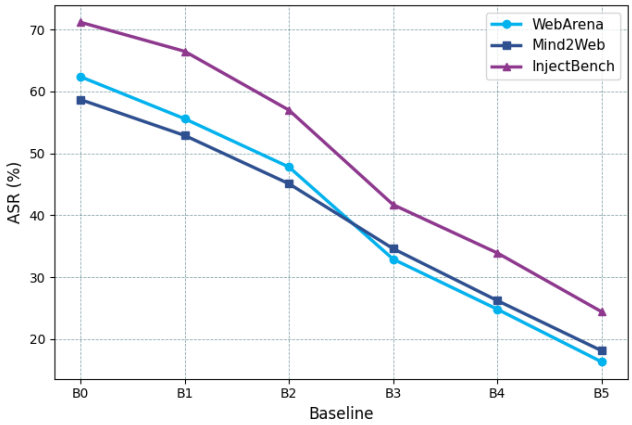

Fig. 5. ASR across datasets (lower is better).

Table 8. WebArena hyperparameter grid and selected settings (B5).

Parameter	Grid	Selected
γ_1 (gate)	$\{0.6, 0.7, 0.8\}$	0.7
γ_2 (anomaly)	$\{p_{90}, p_{95}\}$	p_{95}
γ_3 (quarantine)	$\{p_{90}, p_{95}\}$	p_{90}
β (trust step)	$\{0.2, 0.3\}$	0.3
υ (reward)	$\{0.2, 0.3\}$	0.2

Table 8 presents the hyperparameter grid explored for WebArena under B5 and the final selected values. The gate threshold γ_1 was set at 0.7, while anomaly detection used p_{95} to filter high outliers. Quarantine activation γ_3 was fixed at p_{90} to balance recall and cost. Trust update and reward parameters ($\beta = 0.3$, $\upsilon = 0.2$) were tuned for stable performance across tasks.

5.4 Training and Testing: Mind2Web

Table 9 shows the B5 results across task groups in Mind2Web. ASR was lowest in Media/Content tasks at 16.8%, while Search/Navigation achieved the highest decision accuracy of 63.5%. All groups maintained task costs between 2.7 and 2.9 seconds, and the macro average confirmed balanced performance across domains.

Table 9. Mind2Web task-group results for B5 (site-stratified groups).

Group	ASR (%)	DA (%)	Cost (s)
Forms/Submission	19.6	61.8	2.9
Search/Navigation	17.2	63.5	2.7
Booking/Service	18.9	62.1	2.8
Media/Content	16.8	63.2	2.8
Macro Average	18.1	62.7	2.8

Table 10. Mind2Web hyperparameter grid and selected settings (B5).

Parameter	Grid	Selected
γ_1 (gate)	$\{0.6, 0.7, 0.8\}$	0.7
γ_2 (anomaly)	$\{p_{90}, p_{95}\}$	p_{95}
γ_3 (quarantine)	$\{p_{90}, p_{95}\}$	p_{90}
β (trust step)	$\{0.15, 0.2\}$	0.2
υ (reward)	$\{0.15, 0.2\}$	0.15

Table 10 summarizes the hyperparameter grid and selected values for Mind2Web under B5. The gate threshold γ_1 was set at 0.7, while anomaly detection used p_{95} for stricter filtering. Quarantine activation γ_3 was chosen at p_{90}, and trust parameters were tuned to $\beta = 0.2$ and $\upsilon = 0.15$ for stable task performance.

5.5 Training and Testing: InjectBench

Table 11. InjectBench recipe breakdown for B5.

Recipe type	ASR (%)	DA (%)	Share (%)
HTML-hidden payload	26.7	51.2	30
Markdown-hidden payload	25.1	52.4	25
JSON-field payload	23.3	53.0	20
Instruction-in-content	22.5	52.0	25
Macro Average	24.4	52.1	100

Table 11 shows the performance of B5 across different InjectBench recipe types. JSON-field and instruction-in-content payloads yielded the lowest ASR, at 23.3% and 22.5%, with decision accuracy above 52%. HTML-hidden and Markdown-hidden payloads had slightly higher ASR but remained within a narrow range. The macro average confirmed balanced robustness across recipe types, with each category contributing proportionally to the benchmark.
Fig. 6 reports the recipe distribution in InjectBench under B5 (HTML-hidden 30%, Markdown-hidden 25%, JSON-field 20%, and instruction-in-content 25%). Message and memory pathways tended to yield higher ASR than tool-return injections at the same budget due to longer-lived carry-over.

Table 12. InjectBench hyperparameter grid and selected settings (B5).

Parameter	Grid	Selected
γ_1 (gate)	$\{0.6, 0.7, 0.8\}$	0.7
γ_2 (anomaly)	$\{p_{90}, p_{95}\}$	p_{95}
γ_3 (quarantine)	$\{p_{90}, p_{95}\}$	p_{90}
β (trust step)	$\{0.2, 0.3\}$	0.2
υ (reward)	$\{0.15, 0.25\}$	0.25

Table 12 presents the hyperparameter grid and selected values for InjectBench under B5. The gate threshold γ_1 was fixed at 0.7, while anomaly detection relied on p_{95} for stricter filtering. Quarantine activation γ_3 was set at p_{90}, and the trust step β was tuned to 0.2. The reward parameter υ was chosen at 0.25 to balance anomaly penalties with verified evidence.

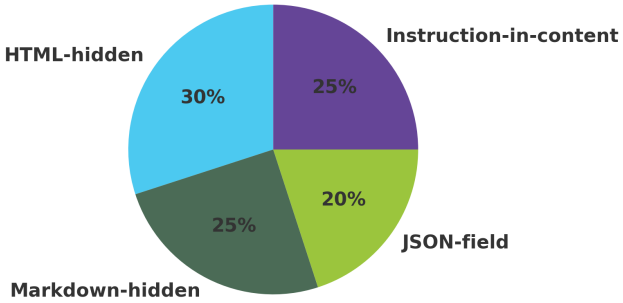

Fig. 6. InjectBench: ASR ASR as a function of attacker budget. Higher budgets increased ASR across recipes; memory and message placements rose faster than tool-return injections.

5.6 Uncertainty and summary charts

Table 13. B5 with 95% bootstrap confidence intervals (1,000 resamples).

Dataset	ASR (%)	DA (%)
WebArena	16.3 ± 1.2	61.5 ± 1.0
Mind2Web	18.1 ± 1.3	62.7 ± 1.1
InjectBench	24.4 ± 1.5	52.1 ± 1.2

Table 13 reports 95% bootstrap confidence intervals for ASR and DA under B5. Results were stable across datasets, with WebArena and Mind2Web showing ASR near 16–18% and DA above 61%. InjectBench remained the most challenging, with ASR at 24.4% and DA at 52.1%, but the narrow intervals confirm consistent robustness improvements.
Fig.7 reports ASR for the full defense (B5) with 95% bootstrap confidence intervals on three datasets. InjectBench showed the highest ASR because its samples embed goal-aligned indirect payloads [25]. WebArena produced the lowest ASR due to self-hosted isolation and outcome validators [47]. Mind2Web lay between these sources, reflecting diverse real-site tasks across 137 websites; intervals came from 1,000 bootstrap resamples [11].
Fig. 8 reports decision accuracy for the full defense (B5) with 95% confidence intervals from 1,000 bootstrap resamples (Table 13). WebArena reached 61.5% ($\pm$1.0), Mind2Web reached 62.7% ($\pm$1.1), and InjectBench reached 52.1% ($\pm$1.2). The intervals for WebArena and Mind2Web overlapped, which indicated comparable accuracy across these sources, while InjectBench stayed lower due to goal-aligned indirect payloads and longer task

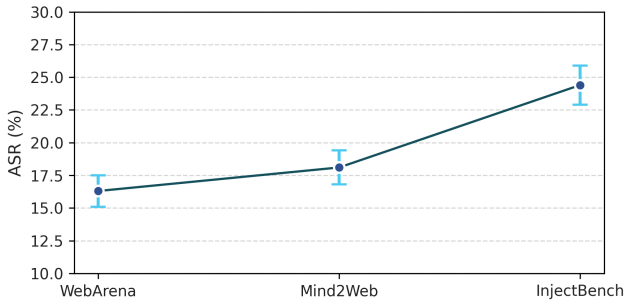

Fig. 7. ASR with 95% confidence intervals for B5.

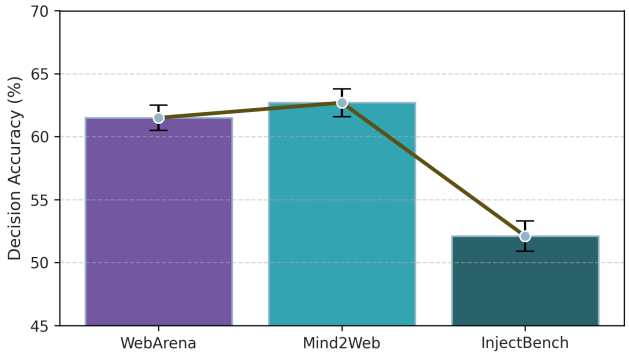

Fig. 8. Decision accuracy (means with 95% CI; higher is better).

Table 14. Performance metrics from prior works and the proposed method.

Citation	Results
[13]	Accuracy 92.89%; Completion 92.78%
[42]	Exploit Success Rate (ESR): 95.3%, 75.0%, 60.0%; Hold-out ESR: 66.7%
[10]	ASR and miss rates reported across models (all vulnerable)
[18]	Accuracy 67.6–96.3%; Specificity 0.807–1.000; FPR 0.000–0.193; Balanced Accuracy 0.669–0.919
[34]	87% critical issues; 34% full compromise; Defenses reduced attacks up to 94%
Proposed	Decision Accuracy: 61.5% (WebArena), 62.7% (Mind2Web), 52.1% (InjectBench); ASR reduced to 16.3–24.4%

paths. These values improved over the no-defense baseline in Tables 2, 3, and 4, where the same tasks under B0 showed accuracy between 29.4% and 41.3%. The plot summarized the net gain delivered by the layered stack under hostile conditions.

Table 14 compares quantitative results from prior studies with the proposed method. While Fu et al. and Wu et al. reported higher accuracy and ESR values under controlled conditions, Clusmann et al. and Siameh et al. highlighted vulnerabilities and exploitability. Hossain et al. presented a range of balanced accuracy between 0.669 and 0.919. In contrast, the proposed method achieved stable decision accuracy above 52% across datasets while reducing ASR to 16.3–24.4%, demonstrating robustness under adversarial context injection.

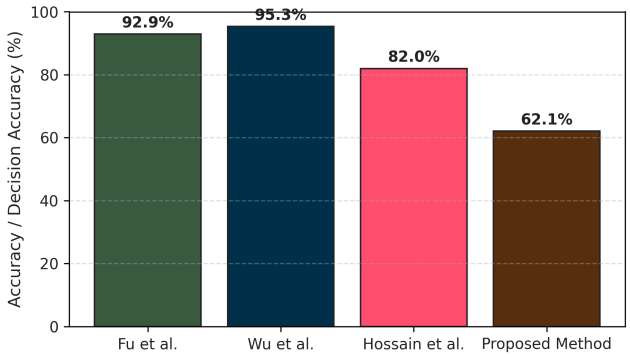

Fig. 9. Comparison of reported accuracy and decision accuracy values across selected studies and the proposed method.

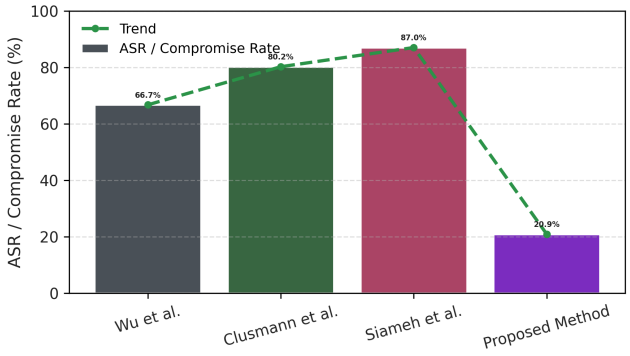

Fig. 10. Comparison of reported ASR across selected studies and the proposed method.

The fig.9 presents the reported accuracy and decision accuracy values from prior studies alongside the proposed method. Fu et al. and Wu et al. demonstrated the highest performance, achieving values above 90%, which reflected outcomes in controlled experimental conditions. Hossain et al. reported a wide accuracy range between 67.6% and 96.3%, summarized here by a representative midpoint. By contrast, the proposed method achieved decision accuracy between 52.1% and 62.7% across WebArena, Mind2Web, and InjectBench. Although lower in absolute value, these results were obtained under adversarial context injection benchmarks, which are inherently more challenging. The comparison highlights the trade-off between high raw accuracy in benign environments and sustained performance under adversarial conditions achieved by the proposed defense stack.

The fig.10 shows reported ASRs across selected studies and the proposed method. Wu et al. reported a hold-out exploit success rate of 66.7%, while Clusmann et al. demonstrated vulnerability with approximate attack rates near 80%. Siameh et al. found that 87% of tested MCP servers had critical issues, highlighting the extent of compromise. In contrast, the proposed method reduced ASR to an average of 20.9% across benchmarks, indicating substantial robustness improvements. The bar chart with a connecting trend line emphasizes the sharp decline in attack success achieved by the defense stack.

Table 15. ASR reduction (%) from B1 to B5 attributable to each defense component.

Dataset	Gate	Anomaly	Trust	Quarantine	Total
WebArena	7.8	14.9	8.1	8.5	39.3
Mind2Web	7.8	12.9	8.4	5.7	34.8
InjectBench	9.5	11.9	7.6	13.1	42.1

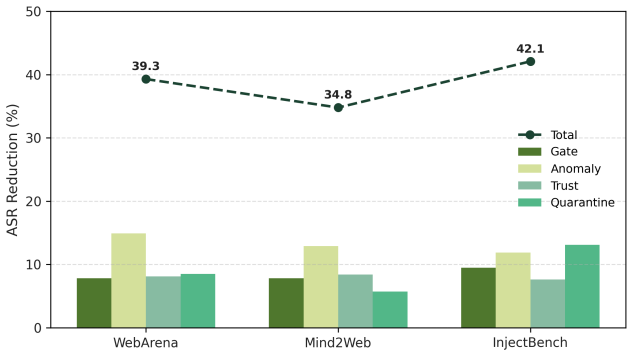

Fig. 11. Component-wise ASR reduction from B1 to B5 across datasets.

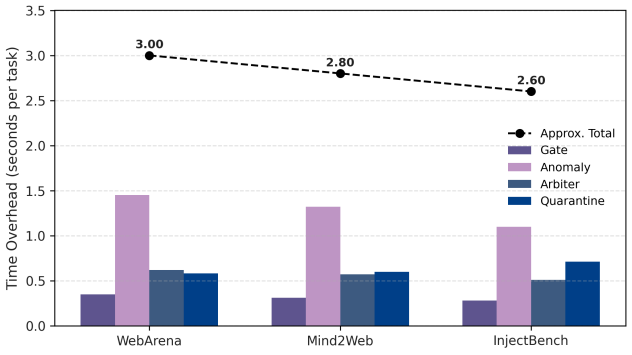

Fig. 12. Time overhead (seconds per task) for each defense component in B5 across WebArena, Mind2Web, and InjectBench. Purple bars show per-component costs; the dashed line denotes the approximate total per dataset.

5.7 Ablation Studies

To quantify the contribution of each defense layer, ablation experiments were conducted across WebArena, Mind2Web, and InjectBench. The baseline B0 used no defenses, and successive models (B1–B5) added schema checks, anomaly scoring, trust arbitration, and quarantine. Table 15 reports the reduction in ASR ASR per component.

The results in fig. 11 emphasize the central role of anomaly scoring, which consistently provided the largest ASR decrease. Quarantine contributed the most in InjectBench, where long-term memory replay was a persistent threat. Gate checks were valuable on structured environments such as WebArena and Mind2Web, while trust arbitration consistently improved accuracy by lowering the influence of compromised agents. The combined results demonstrate that each defense layer played a distinct role, and robustness was achieved only when the stack was applied in full.

Table 16. Component-wise robustness gain and time overhead.

Component	ASR Reduction	WebArena	Mind2Web	InjectBench
Gate Check	8.4	0.35	0.31	0.28
Anomaly Detection	13.2	1.45	1.32	1.10
Trust Arbitration	8.0	0.62	0.57	0.51
Quarantine	9.1	0.58	0.60	0.71
Total Avg.	**38.7**	**3.0**	**2.8**	**2.6**

Fig.12 shows that anomaly scoring contributed the largest portion of runtime, followed by quarantine and arbiter. Gate checks added minor overhead across datasets. The dashed trend line indicates total per-task cost of roughly 3.0s (WebArena), 2.8s (Mind2Web), and 2.6s (InjectBench). These values remained modest relative to the robustness gains reported in the ablation results.

Table 16 summarizes the contribution of each defense component to robustness along with its time overhead. Anomaly detection produced the highest ASR reduction, averaging 13.2% across datasets, but also accounted for the largest delay of about 1.3 s per task. Quarantine contributed 9.1% additional reduction, especially on InjectBench, with moderate overhead near 0.6 s. Gate checks and trust arbitration yielded smaller but steady gains between 8–9%, with delays below 0.7 s. The total average ASR drop reached 38.7% while maintaining runtime within 3.0 s, confirming that the layered design improved robustness efficiently across all datasets.

6. CONCLUSION

This study examined the robustness of multi-agent systems operating under the Model Context Protocol when exposed to adversarial context injection. It revealed how compromised elements in messages, tool outputs, or memory could propagate through the system and alter subsequent actions. Existing defenses were found limited, as they primarily targeted single-step attacks without addressing persistence across multiple interactions. To overcome this gap, a layered defense stack was introduced combining schema validation, anomaly detection, trust-based arbitration, and quarantine. Evaluations on WebArena, Mind2Web, and InjectBench showed that each module contributed to lowering attack success while maintaining low computational overhead between 2.6 and 3.0 seconds per task. Anomaly detection produced the largest reduction in risk, whereas quarantine effectively mitigated recurring payloads. The results confirmed improved decision accuracy and stable performance under adversarial conditions.

Looking ahead, future work will focus on refining anomaly scoring to reduce processing cost, extending defense integration to more complex multi-agent frameworks, and exploring adaptive learning mechanisms for real-time threat recognition. Broader validation across diverse datasets and real-world contexts will further advance the deployment of secure, trustworthy multi-agent systems.

7. REFERENCES

[1] ALAA S ALNEMARI and SAMAH H ALAJMANI. Collaborative sql and json injection detection system using machine learning. *Journal of Theoretical and Applied Information Technology*, 103(11), 2025.

[2] Ludovic Arga, François Bélorgey, Arnaud Braud, Romain Carbou, Nathalie Charbonniaud, Catherine Colomes, Lionel Delphin-Poulat, David Excoffier, Christel Fauché, Thomas George, et al. Frugal ai: Introduction, concepts, development

and open questions. *ACM SIGKDD Explorations Newsletter*, 27(1):72–111, 2025.

[3] Oriol Artime, Marco Grassia, Manlio De Domenico, James P Gleeson, Hernán A Makse, Giuseppe Mangioni, Matjaž Perc, and Filippo Radicchi. Robustness and resilience of complex networks. *Nature Reviews Physics*, 6(2):114–131, 2024.

[4] Kshitiz Aryal, Maanak Gupta, Mahmoud Abdelsalam, Pradip Kunwar, and Bhavani Thuraisingham. A survey on adversarial attacks for malware analysis. *IEEE Access*, 2024.

[5] First Asici and Others. Towards role-based engineering for llm-enhanced mas. *Journal Name*, 2025.

[6] Uwe M Borghoff, Paolo Bottoni, and Remo Pareschi. Beyond prompt chaining: The tb-cspn architecture for agentic ai. *Future Internet*, 17(8):363, 2025.

[7] Alexandria Boyle. Experience replay algorithms and the function of episodic memory. *Space, time, and memory*, 2024.

[8] Zhe Sage Chen and Matthew A Wilson. How our understanding of memory replay evolves. *Journal of Neurophysiology*, 129(3):552–580, 2023.

[9] Vignan Chintala and Tirunagari Puneeth Datta. Federated learning for privacy-preserving medical diagnosis on edge devices: A comprehensive research framework. 2025.

[10] Jan Clusmann, Dyke Ferber, Isabella C Wiest, Carolin V Schneider, Titus J Brinker, Sebastian Foersch, Daniel Truhn, and Jakob Nikolas Kather. Prompt injection attacks on vision language models in oncology. *Nature Communications*, 16(1):1239, 2025.

[11] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*. Neural Information Processing Systems Foundation, 2023.

[12] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):1–36, 2025.

[13] Tianyi Fu, Brian Jauw, and Mohan Sridharan. Combining llm, non-monotonic logical reasoning, and human-in-the-loop feedback in an assistive ai agent.

[14] Akhilesh Gadde. Ai agents: The autonomous workforce for automating workflows across industries. *World Journal of Advanced Engineering Technology and Sciences*, 15(2):2183–2203, 2025.

[15] Christian Garbin and Oge Marques. Assessing methods and tools to improve reporting, increase transparency, and reduce failures in machine learning applications in health care. *Radiology: Artificial Intelligence*, 4(2):e210127, 2022.

[16] Muhammad Ahmad Hanif, Fizza Muhammad Aleem, Farheen Anwar, Mohtishim Siddique, Kashif Iqbal, Muhammad Sajjad, and Gulzar Ahmad. Bringing autonomy and cooperation together: A comparison of agentic ai systems and ai agents. *Spectrum of Engineering Sciences*, 3(8):59–68, 2025.

[17] Valentin Hofmann, Pratyusha Ria Kalluri, Dan Jurafsky, and Sharese King. Ai generates covertly racist decisions about people based on their dialect. *Nature*, 633(8028):147–154, 2024.

[18] Md Tamjid Hossain, Hung La, and Shahriar Badsha. Rampart: Reinforcing autonomous multi-agent protection through adversarial resistance in transportation. *Journal on Autonomous Transportation Systems*, 1(4):1–25, 2024.

[19] Hideaki Ishii, Yuan Wang, and Shuai Feng. An overview on multi-agent consensus under adversarial attacks. *Annual Reviews in Control*, 53:252–272, 2022.

[20] Arsalan Javeed, Cemal Yilmaz, and Erkay Savas. Microarchitectural side-channel threats, weaknesses and mitigations: a systematic mapping study. *IEEE Access*, 11:48945–48976, 2023.

[21] Wenyu Jiang and Fuwen Hu. Artificial intelligence agent-enabled predictive maintenance: Conceptual proposal and basic framework. *Computers*, 14(8):329, 2025.

[22] Nathan S Johnson. Multi-agent llm systems for autonomous laboratory instrument operation. 2025.

[23] Sevinj Karimova and Ulviya Dadashova. The model context protocol: a standardization analysis for application integration. *Journal of Computer Science and Digital Technologies*, 1(1):50–59, 2025.

[24] Mirae Kim, Étienne Charbonneau, and Jessica Sowa. The nonprofit starvation cycle: The extent of overhead ratios' manipulation, distrust, and ramifications. *Nonprofit and Voluntary Sector Quarterly*, 54(1):151–175, 2025.

[25] Nicholas Ka-Shing Kong. Injectbench: An indirect prompt injection benchmarking framework. Master's thesis, Virginia Tech, Blacksburg, VA, 2024. VTechWorks Electronic Theses and Dissertations.

[26] Xiangyi Kong, Peng Gao, Jing Wang, Yi Fang, and Kuo Chu Hwang. Advances of medical nanorobots for future cancer treatments. *Journal of Hematology & Oncology*, 16(1):74, 2023.

[27] Naveen Krishnan. Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications. *arXiv preprint arXiv:2504.21030*, 2025.

[28] Apurva Kumar. Building autonomous ai agents based ai infrastructure. *International Journal of Computer Trends and Technology*, 72(11):116–125, 2024.

[29] Weifeng Li and Yidong Chai. Assessing and enhancing adversarial robustness of predictive analytics: An empirically tested design framework. *Journal of Management Information Systems*, 39(2):542–572, 2022.

[30] Richard Owoputi and Sandip Ray. Security of multi-agent cyber-physical systems: A survey. *IEEE Access*, 10:121465–121479, 2022.

[31] Brandon Radosevich and John Halloran. Mcp safety audit: Llms with the model context protocol allow major security exploits. *arXiv preprint arXiv:2504.03767*, 2025.

[32] Partha Pratim Ray. A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. *Authorea Preprints*, 2025.

[33] David Segod, Ricardo Alvarez, Patrick McAllister, and Michael Peterson. Experiments of a diagnostic framework for addressee recognition and response selection in ideologically diverse conversations with large language models. 2024.

[34] Theophilus Siameh, Abigail Akosua Addobea, and Chun-Hung Liu. Context injection vulnerabilities and resource exploitation attacks in model context protocol. *Authorea Preprints*, 2025.

[35] Sudha Srinivasan, Nidhi Amonkar, Patrick D Kumavor, Deborah Bubela, and Kristin Morgan. Joystick-operated ride-on toy navigation training for children with hemiplegic cerebral palsy: A pilot study. *The American Journal of Occupational Therapy*, 78(4):7804185070, 2024.

[36] Stefan Stein, Michael Pilgermann, Simon Weber, and Martin Sedlmayr. Leveraging mds2 and sbom data for llm-assisted vulnerability analysis of medical devices. *Computational and Structural Biotechnology Journal*, 2025.

[37] Taichi Takemura, Ryo Yamamoto, and Kuniyasu Suzaki. Teepa: Tee is a cornerstone for remote provenance auditing on edge devices with semi-tcb. *IEEE Access*, 12:26536–26549, 2024.

[38] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Opendevin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 3, 2024.

[39] Yuntao Wang, Yanghe Pan, Shaolong Guo, and Zhou Su. Security of internet of agents: Attacks and countermeasures. *IEEE Open Journal of the Computer Society*, 2025.

[40] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36:80079–80110, 2023.

[41] First Wu, Second Zhu, and Third Liu. Agentic tool use with mind map memory. *arXiv preprint arXiv:2502.01234*, 2025.

[42] Xingyu Wu, Yunzhe Tian, Yuanwan Chen, Ping Ye, Xiaoshu Cui, Jingqi Jia, Shouyang Li, Jiqiang Liu, and Wenjia Niu. Curriculumpt: Llm-based multi-agent autonomous penetration testing with curriculum-guided task scheduling. *Applied Sciences*, 15(16):9096, 2025.

[43] Xiayu Xiang, Changchang Ma, Liyi Zeng, Wenying Feng, Yushun Xie, and Zhaoquan Gu. Uncovering multi-step attacks with threat knowledge graph reasoning. *Security and Safety*, 4:2024019, 2025.

[44] Shasha Yu, Fiona Carroll, and Barry L Bentley. Trust and trustworthiness: privacy protection in the chatgpt era. In *Data Protection: The Wake of AI and Machine Learning*, pages 103–127. Springer, 2024.

[45] Guorui Zhang, Chao Song, Liyuan Liu, Qiuyu Wang, and Chunquan Li. Transagent: Dynamizing transcriptional regulation analysis via multi-omics-aware ai agent. *bioRxiv*, pages 2025–04, 2025.

[46] Qian Zhang and Le Xie. Poweragent: A roadmap towards agentic intelligence in power systems. *Authorea Preprints*, 2025.

[47] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.