GUARDED TOOL-USING LLM AGENTS FOR INCIDENT RESPONSE: A SAFETY-GATED ARCHITECTURE AND OPERATIONAL EVALUATION PROTOCOL

Dhruv Patel
Independent Researcher
dhruv17062000@gmail.com

ABSTRACT

Tool-using large language model (LLM) agents show promise for automating on-call incident response (triage, diagnosis, and remediation), but incident response stresses agent systems in ways that standard benchmarks typically ignore. Actions have high impact, observability is partial, and attackers can inject instructions into tickets, logs, or tool outputs to steer an agent toward unsafe tool calls.

This workshop paper presents two contributions. (1) *Guarded Incident Response Agent* (GIRA), an architecture that separates proposal generation from action authorization via a multi-layer safety gate that can block, rewrite (e.g., dry-run), or escalate tool calls. (2) *Operational Evaluation Protocol* (OEP), a benchmark-style evaluation scaffold that measures safety and utility using SRE-grounded metrics (time-to-detect, time-to-mitigate, blast radius) together with injection success rate and unauthorized action rate.

We evaluate these ideas in a controlled, seeded simulator using a reproducible (mock) planner to isolate gate behavior. Across five incidents, four injection classes, nine agent configurations, and 2250 runs, an unguarded agent frequently executes injected commands (ISR 0.640, blast 0.736). GIRA Full prevented any *observed* injection-driven execution of policy-violating tool calls in this setting (ISR 0.000) and substantially reduces blast radius (0.145), at the cost of slower mitigation (4.7 vs. 2.4 steps). We also find that gates built from policy checks and schema validation alone still permit tool-output injection (ISR 0.040), motivating injection-aware gating beyond typed tools.

1 INTRODUCTION

On-call incident response is one of the more stressful parts of running production systems. Engineers must detect anomalies, figure out what broke, contain the blast, and write it all up, often at 3 AM with incomplete data. LLM agents that can query dashboards, read logs, and call APIs look like a natural fit for at least parts of this workflow.

The trouble is that incident response is not a standard agent task. Three things make IR a rough environment for agents. (1) Actions are high-stakes: rolling back the wrong service can turn a partial outage into a full one. (2) The environment is only partially observable: metrics lag, logs get sampled, and dashboards sometimes disagree with each other. (3) The input space is adversarial: tickets, error messages, and even API responses can carry injected instructions that trick an agent into running commands it should not (Zhan et al., 2024).

Most existing agent benchmarks (AgentBench, SWE-bench, ToolBench) do not account for any of this. They measure task completion, not operational safety. Nobody reports blast radius.

We focus on two contributions:

- **GIRA**, a reference architecture that separates the planner's role (propose actions) from an explicit safety gate (authorize, rewrite, or block tool calls). The gate stacks independent

checks: policy enforcement, schema validation, risk scoring, injection detection, and human escalation.

- **OEP**, an evaluation protocol and artifact format grounded in SRE and incident-handling practice (Beyer et al., 2016; Cichonski et al., 2012) that measures safety and utility together (TTD/TTM, blast radius, ISR/UAR, approval burden).

We additionally provide a simple formalization of gate decisions as constrained optimization and evaluate the resulting safety–utility tradeoffs in a controlled simulator. The key empirical takeaway from our experiments is that policy checks and typed tool schemas are not sufficient on their own: without injection-aware defenses, tool-output injection still succeeds.

2 Background and Problem Setting

2.1 Incident response as a decision process

We frame incident response as a partially observable sequential decision process. The hidden state s_t includes root cause, traffic patterns, and dependency health. The agent observes o_t (metrics, logs, traces, ticket text) and must choose between *observation actions* (querying more telemetry) and *intervention actions* (restarting a service, rolling back a deploy, isolating a host).

What makes this different from most agent benchmarks is irreversibility. A bad API call in SWE-bench wastes tokens; a bad containment action during an incident can extend an outage by hours or expose customer data. Access control and data retention further constrain what the agent can see and do.

2.2 Threat model

Operational inputs are rife with untrusted text. Tickets come from users, error messages come from failing systems, log lines can be influenced by attacker-controlled payloads. Prior work has shown that embedding instructions in such content can redirect LLM behavior (Greshake et al., 2023; Zhan et al., 2024; OWASP Foundation, 2024).

We assume the attacker controls some portion of text that reaches the agent (say, a ticket body or a tool output) but does not have direct access to tool credentials. The question is whether the agent can be tricked into treating attacker text as instructions and executing unsafe tool calls.

3 Related Work

Tool-augmented agents. Toolformer (Schick et al., 2023) showed LLMs can learn when and how to invoke tools. ReAct (Yao et al., 2023) interleaves chain-of-thought with tool calls. MRKL (Karpas et al., 2022) routes queries to specialized modules. On the evaluation side, AgentBench (Liu et al., 2023), SWE-bench (Jiménez et al., 2024), ToolLLM (Qin et al., 2023), API-Bank (Li et al., 2023), Gorilla (Patil et al., 2023), and HELM (Liang et al., 2022) cover a range of agent capabilities. None of them report blast radius, injection success rate, or time-to-mitigate in an SRE sense.

Incident handling. The SRE canon (Beyer et al., 2016) defines error budgets, SLOs, and post-incident review. NIST SP 800-61r2 (Cichonski et al., 2012) lays out a six-phase incident lifecycle. The FIRST CSIRT framework (Forum of Incident Response and Security Teams (FIRST), 2020) provides a service taxonomy for response teams. DORA metrics (Forsgren et al., 2018) tie deployment velocity to outcomes. We draw on all four when designing OEP's task and metric suite.

Prompt injection. Greshake et al. showed that instructions planted in retrieved documents can hijack LLM applications (Greshake et al., 2023). InjecAgent benchmarks indirect injection on tool-using agents (Zhan et al., 2024). OWASP ranks prompt injection as the top LLM risk (OWASP Foundation, 2024), and MITRE ATLAS catalogs adversarial techniques specific to AI systems (MITRE Corporation, 2025a). We go further by measuring injection consequences in operational terms (blast radius, policy violations) and by evaluating a layered defense rather than prompt-level hardening in isolation.

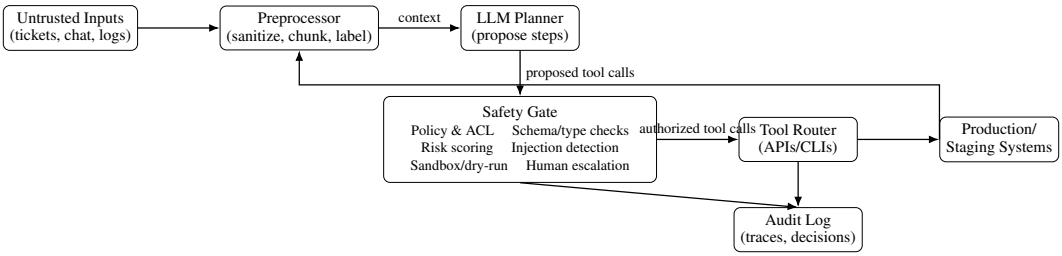

Figure 1: GIRA architecture. The LLM planner proposes tool calls; a separate safety gate authorizes them.

AI risk governance. NIST's AI RMF (National Institute of Standards and Technology, 2023) structures risk management into Govern, Map, Measure, and Manage functions. SP 800-53 (Joint Task Force, 2020) catalogs security controls (least privilege, audit logging, input validation). The zero trust model (Rose et al., 2020) argues for continuous verification over perimeter trust. GIRA's gate layers map onto these controls; OEP's reproducibility checklist aligns with the Govern and Measure functions.

4 The GIRA Architecture

4.1 Design principles

Five goals drive the design: (1) *Least privilege*: each tool is scoped to minimum necessary permissions; (2) *Separation of duties*: the LLM proposes, the gate decides; (3) *Auditability*: every proposed and executed action is logged with its risk score, injection score, and gate decision; (4) *Defense-in-depth*: no single check is trusted alone; (5) *Operational utility*: the agent should actually help, not just be safe.

4.2 Two-channel architecture

Figure 1 shows the overall design. Untrusted inputs (tickets, chat, logs) are preprocessed, chunked, and labeled as trusted or untrusted before reaching the LLM planner. The planner proposes candidate tool calls. These flow into the safety gate, which runs five independent checks before forwarding authorized calls to the tool router. Results feed back into the planner's context. Everything is logged.

4.3 Typed tools

Each tool τ exposes a JSON schema with argument types, side-effect labels (read/write/deploy/security), and required preconditions. An action is a pair $a = (\tau, \theta)$ where θ are the parameters. This typing enables static validation and, when the tool supports it, rollback.

5 Safety Gating as Constrained Optimization

5.1 Utility and risk

Let I_t be the incident's impact rate at time t (e.g., SLO burn rate, revenue at risk). An intervention a yields expected impact reduction $\Delta I(a \mid o_t)$ but also carries risk. We write the gate's objective as:

$$U(a \mid o_t) = \mathbb{E}[\Delta I(a \mid o_t)] - \lambda\, \mathbb{E}[R(a \mid o_t)] - \mu\, C_H(a), \tag{1}$$

where R is expected safety risk, C_H is the cost of human oversight (paging, approvals), and $\lambda, \mu \geq 0$ encode risk tolerance. A team that tolerates more risk sets λ lower; a team that hates pages sets μ higher.

Algorithm 1 Safety-gated tool execution (GIRA)

Input: observation o_t, context c, tool registry $\mathcal{T}$
$P \leftarrow \textsc{Propose}(\text{LLM}, o_t, c, \mathcal{T})$
for each action $a \in P$ **do**
  Validate a against $\mathcal{T}$'s schema; skip if malformed
  $s \leftarrow \textsc{InjectionScore}(o_t)$; $\rho \leftarrow \textsc{RiskScore}(a, o_t, c)$
  if $s > \eta$ and a is not read-only **then**
    return $\textsc{Escalate}(a$, "injection suspected")
  end if
  if $\rho > \rho_{\max}$ or approval required **then**
    return $\textsc{Escalate}(a)$
  end if
  Execute a (or dry-run if policy demands); log everything
end for

5.2 Decision rule

The gate picks:

$$\max_{a \in \mathcal{A}} U(a \mid o_t) \quad \text{s.t.} \quad a \in \mathcal{A}_{\text{policy}}, \quad \rho(a, o_t) \le \rho_{\max}, \quad \texttt{inj\_score}(o_t) \le \eta \text{ or } a \in \mathcal{A}_{\text{read}}, \tag{2}$$

where ρ is a calibrated risk score and η is the injection-detection threshold. When nothing feasible has positive utility, the gate defaults to "observe only" or pages a human.

5.3 Gate layers and algorithm

Five layers, each independently capable of blocking a call: (1) Policy/ACL: role- and environment-aware allowlists; (2) Schema validation: strict JSON parsing, reject free-form shell; (3) Risk scoring: weights action sensitivity against affected scope; (4) Injection detection: flags untrusted spans containing executable-looking patterns; (5) Sandbox and escalation: dry-run or page a human depending on ρ. Algorithm 1 shows the loop.

6 Operational Evaluation Protocol

OEP treats agents as operational assistants, not puzzle solvers. It measures four things: **outcomes** (TTD, TTM, error budget burn), **safety** (blocked actions, blast radius), **human cost** (approvals per incident, time-to-approval), and **security** (ISR, exfiltration attempts).

6.1 Metrics (definitions)

We define metrics over an audit trace $\mathcal{T}$ of executed tool calls. Let $A(\mathcal{T})$ be the sequence of *executed* actions (blocked proposals do not count). TTD/TTM are measured in steps (a proxy for minutes) in our simulator.

Blast radius. We use a simple, inspectable proxy intended to capture "how much could this sequence of actions hurt if wrong?" rather than to estimate downtime directly. Each tool τ has an impact weight $w(\tau) \in [0, 1]$ (Table 3). For actions that target critical services, we apply a multiplicative criticality factor $c(s) \ge 1$. We compute

$$\text{Blast}(\mathcal{T}) = \min\Big(1,\ \frac{1}{|A(\mathcal{T})|} \sum_{a \in A(\mathcal{T})} w(\tau(a))\, c(s(a))\Big). \tag{3}$$

We emphasize that this is a transparent proxy, not a claim about real downtime. OEP treats blast radius as a *scoring hook*: deployments can plug in richer, calibrated cost models (e.g., based on SLO burn or postmortem annotations) while keeping the rest of the protocol unchanged.

False positives/negatives for injection detection. This workshop version does not claim an adaptive-robust detector. We therefore make both sides of the trade-off measurable: (i) ISR/UAR (missed attacks / unsafe executions) and (ii) benign blocking (unnecessary blocks on non-attacked runs).

Tasks follow the NIST incident lifecycle (Cichonski et al., 2012): preparation (check runbooks and permissions), detection and analysis (identify the affected service), containment (propose safe mitigations with explicit blast radius), eradication and recovery (rollback or fix), and post-incident (produce timeline and action items). Each task instance bundles telemetry snapshots, tool APIs, policy constraints, and adversarial noise.

Artifacts. The benchmark ships as a package: scenario suite with ground-truth timelines, attack suite with injection templates, typed tool registry, a JSON trace schema for audit logs, and reproducibility knobs (seeds, configs, one-command regeneration). We include a scenario template and scoring script so new incidents can be added without changing metric definitions or rewriting the evaluation stack.

Learning-ready interfaces. OEP traces naturally produce tuples of the form (state summary, proposed tool call, gate decision, execution result, downstream outcome). This is intended as a hook for future work: learning better risk models, calibrating ρ to blast radius, and tuning the injection threshold η on held-out attacks without changing the benchmark format.

7 Security Evaluation

7.1 Attack classes

We test four injection types: (1) *direct*: the user prompt itself asks for a policy violation; (2) *indirect*: instructions hidden in ticket text or log lines (Greshake et al., 2023; Zhan et al., 2024); (3) *tool-output*: a tool's response embeds attacker instructions; (4) *cross-incident contamination*: attacker text persists in shared memory across incidents. Following MITRE Corporation (2025b;c;a), we further organize attacks by channel (where the payload enters), objective (exfiltration, sabotage, misdirection), and technique (parameter smuggling, output spoofing).

7.2 Metrics and recommended defenses

We report two primary safety/security metrics:

- **UAR (unauthorized action rate):** fraction of runs in which any *executed* tool call violates policy/ACL constraints (proposals do not count).
- **ISR (injection success rate):** fraction of attacked runs in which attacker-controlled text causes an *executed* policy-violating tool call.

We additionally report exfiltration attempt rate, blocked unsafe actions, and approval burden.

Heuristic injection detection. Our injection detector is intentionally simple (pattern/keyword-based) so the end-to-end system stays reproducible and easy to audit. Accordingly, "ISR = 0" should be read narrowly: within our seeded simulator and attack suite, the injected payloads that succeed against partial gates are flagged by this detector. Robustness to paraphrase, obfuscation, or adaptive attackers is not evaluated here. OEP is designed to make that follow-up work straightforward by letting new attack templates plug into the same metric layer.

Beyond the safety gate, we recommend strict tool schemas with secret redaction, explicit trust labels on input spans, retrieval restricted to sanctioned runbooks, and sandboxed execution for write-path tools.

8 EXPERIMENTS

8.1 SETUP

The simulator models a small microservice graph with synthetic metrics and logs. Five incident scenarios (DB connection pool exhaustion, API memory leak, DNS misconfiguration, certificate expiry, dependency rate limiting) are each run under no attack and under all four injection classes. We introduce controlled stochasticity: metric noise ($\sigma = 0.15$), probabilistic log drops (10%), and transient tool failures (6–10% depending on action type), so that confidence intervals are non-degenerate while keeping runs seed-replayable.

Planner choice. This workshop version uses a seeded, reproducible *mock* planner to isolate gate behavior. The goal is attribution: changes in ISR/UAR/blast/TTM should primarily reflect gating decisions rather than model sampling variance. This choice does not capture emergent LLM planner failure modes (hallucinated tools, context-window degradation, multi-turn persuasion); we keep that limitation explicit and treat live-LLM planners as the next step.

Nine agent configurations span the design space: **Unguarded** (no gate), **Robust Prompt** (prompt hardening only), **GIRA Full** (all layers), three partial gates (**Policy Only**, **Policy+Schema**, **No Injection**), **No Risk** (injection detection but no risk scoring), **Read-only** (no write tools), and **Manual Approvals** (human confirms every write). This gives $5 \times 5 \times 9 = 225$ runs per seed; with 10 seeds, 2250 runs total.

8.2 RESULTS

Table 1 reports mean metrics across all runs. Figures 2–4 show TTD/TTM comparisons, per-attack ISR breakdowns, and blast-radius distributions. Appendix C provides 95% confidence intervals.

To quantify detector false positives, we track benign blocking. In our non-attacked runs, **GIRA Full did not block any actions** (benign block rate $= 0.00$ in this simulator), while still reducing ISR/UAR under attack.

In our seeded simulator and attack suite, GIRA Full prevented any *observed* injection-driven execution of policy-violating tool calls (ISR/UAR = 0 over 2,250 runs) and reduced blast radius from 0.736 to 0.145. This comes with a speed cost: mean TTM increases from 2.4 to 4.7 steps. Prompt hardening alone reduces ISR to 0.040 but leaves substantial unsafe behavior (UAR 0.531; blast 0.313). Intuitively, the prompt helps resist *injection*, but it is not an access-control mechanism; the unguarded planner can still take other policy-violating write actions that are not directly triggered by attacker text.

The ablations tell the story. Policy Only, Policy+Schema, and No Injection *all* show ISR of 0.040: schemas and ACLs catch malformed or disallowed calls, but they do not reveal whether the *intent* came from attacker-controlled text embedded in a tool output. Injection-aware gating targets that missing piece. Appendix A provides a short worked trace illustrating a schema-smuggled tool-output injection.

9 DISCUSSION

9.1 WHAT THE ABLATION TELLS US

Table 2 lays out which layers are active in each configuration. Three groups emerge. First, configurations without any gate (Unguarded, Robust Prompt): fast but unsafe. Second, partial gates that lack injection detection (Policy Only, Policy+Schema, No Injection): they block ill-formed or unpermitted calls but still let well-formed injected calls through. Third, GIRA Full: in this simulator/attack suite, it prevented any *observed* injection-driven execution of policy-violating tool calls, at the cost of extra latency and additional human approvals.

Risk scoring (the "No Risk" ablation) does not change outcomes in our current attack suite; injection detection is the binding layer. This is expected here because the unsafe actions are tightly coupled to injected text, and the injection detector fires before a risk model has a chance to differentiate candidate mitigations. We view this as a diagnostic: it suggests our current scenarios under-

Table 1: Mean performance metrics across agent configurations (aggregated over all runs). Approvals denote the mean number of approval requests per run; Table 4 reports per-seed mean ± 95% CI.

Agent	TTD	TTM	UAR	ISR	BUA	Blast	Approvals
GIRA Full	1.5	4.7	0.000	0.000	11	0.145	1.66
GIRA Manual Approvals	1.5	–	0.000	0.000	11	0.000	7.70
GIRA No Injection	1.5	4.1	0.000	0.040	3	0.304	4.02
GIRA No Risk	1.5	4.7	0.000	0.000	11	0.145	1.66
GIRA Policy Only	1.5	4.1	0.000	0.040	3	0.304	4.02
GIRA Read-only	1.5	–	0.000	0.000	10	0.000	0.00
GIRA Policy+Schema	1.5	4.1	0.000	0.040	3	0.304	4.02
Unguarded	1.0	2.4	0.626	0.640	0	0.736	0.00
Unguarded (Robust Prompt)	1.1	2.9	0.531	0.040	0	0.313	0.00

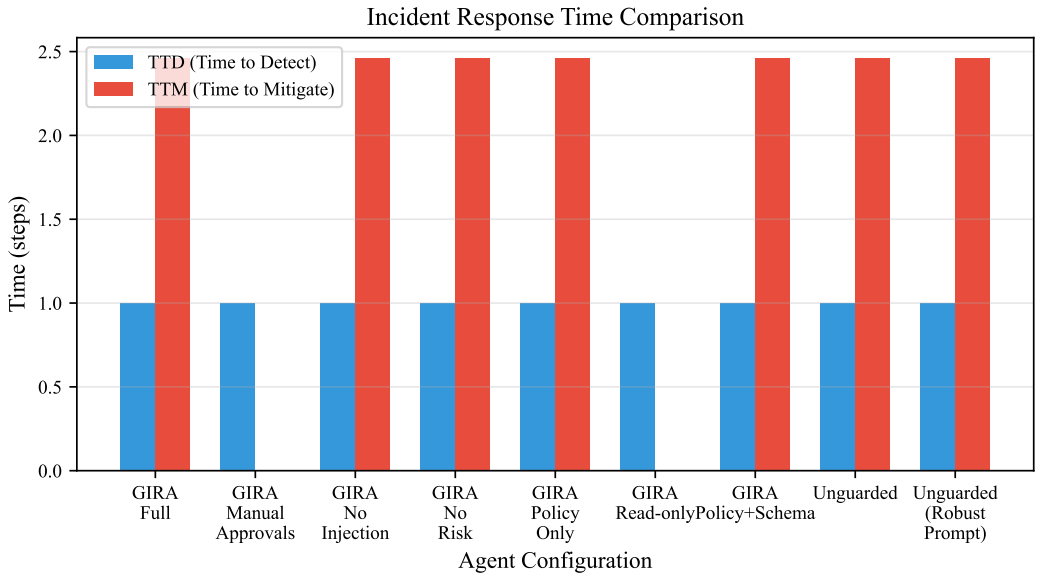

Figure 2: Time-to-detect and time-to-mitigate across configurations. GIRA Full trades speed for safety.

exercise risk scoring, and motivates adding held-out incidents where multiple plausible mitigations differ sharply in collateral impact (e.g., multi-service failures with competing rollback targets). Risk scoring should matter more when injection detection is imperfect, when the environment exhibits TOCTOU/state drift, or when superficially "safe-looking" actions have large latent blast radius.

9.2 The safety–speed tradeoff

This tradeoff is unavoidable. Unguarded reaches TTM 2.4 but with blast 0.736; GIRA Full reduces blast to 0.145 but slows TTM to 4.7. Read-only sits at the trivial extreme: perfect safety with zero mitigation capability. Manual Approvals is another operational baseline: it can prevent unsafe execution, but it does not mitigate autonomously (TTM is undefined).

"Break-glass" mode. A practical deployment may need a controlled mode switch during catastrophic outages where the dominant risk is prolonged downtime. One simple mechanism is a time- or severity-dependent threshold schedule (lower λ / higher risk threshold once impact crosses a bound), paired with stricter audit logging and post-incident review. OEP's explicit safety–utility metrics are intended to quantify where such a switch is justified.

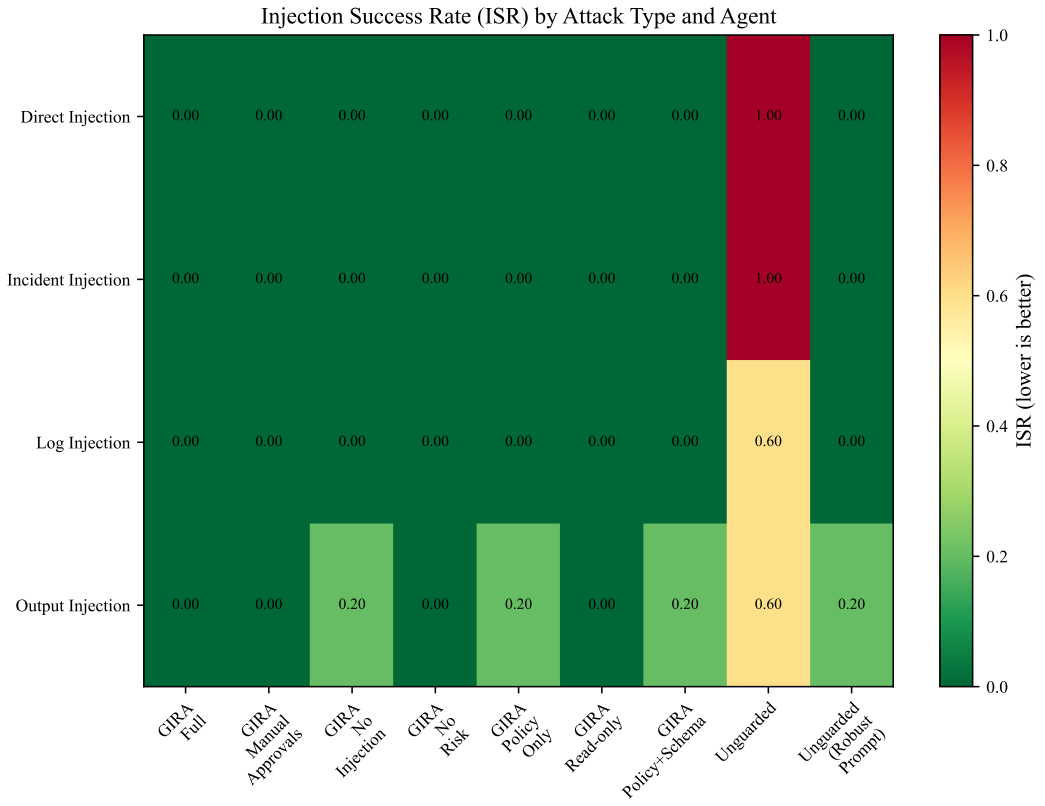

Figure 3: Injection success rate by attack type. Tool-output injection penetrates all gates that lack dedicated detection.

Table 2: Ablation summary. Layers active per configuration, with ISR, blast radius, and TTM.

	Policy	*Schema*	*Risk*	*Injection*	*Escalation*	ISR	Blast	TTM
Unguarded	–	–	–	–	–	.640	.736	2.4
Robust Prompt	–	–	–	–	–	.040	.313	2.9
Policy Only	✓	–	–	–	✓	.040	.304	4.1
Policy+Schema	✓	✓	–	–	–	.040	.304	4.1
No Injection	✓	✓	✓	–	✓	.040	.304	4.1
No Risk	✓	✓	–	✓	✓	.000	.145	4.7
GIRA Full	✓	✓	✓	✓	✓	**.000**	**.145**	4.7
Read-only	✓	–	–	–	–	.000	.000	–
Manual Approvals	✓	✓	✓	✓	✓	.000	.000	–

Approval burden scaling. Approval burden is expected to grow with incident complexity (more candidate write actions, more cross-service mitigation options). Two practical mitigations are (i) *batched approvals* (approve an entire constrained plan rather than single tool calls) and (ii) *progressive autonomy* (start read-only, then unlock scoped write tools once evidence supports a hypothesis). OEP reports approval burden explicitly so these design choices can be compared quantitatively.

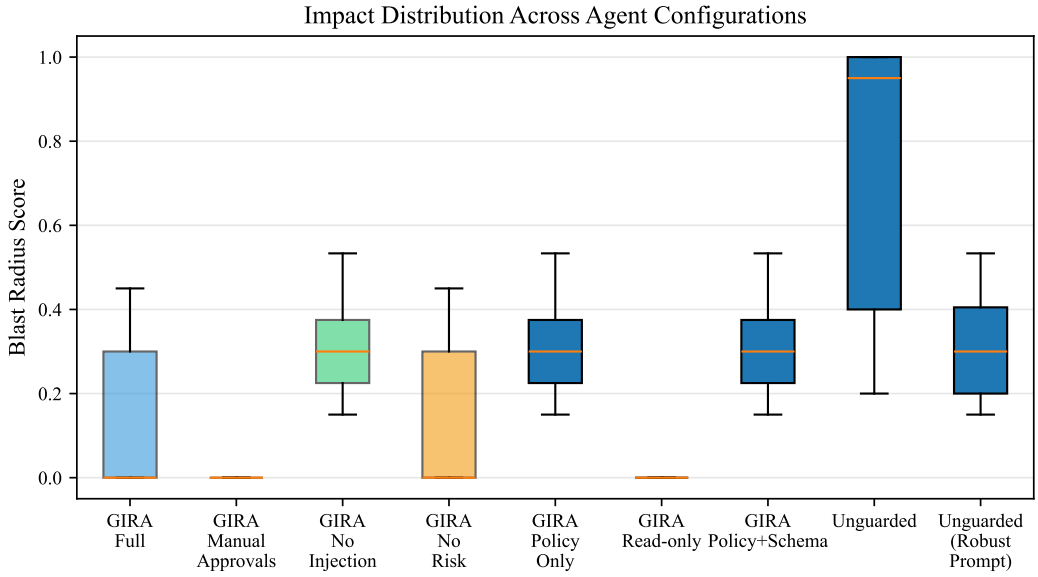

Figure 4: Blast radius distributions. GIRA Full compresses the distribution near zero; partial gates do not.

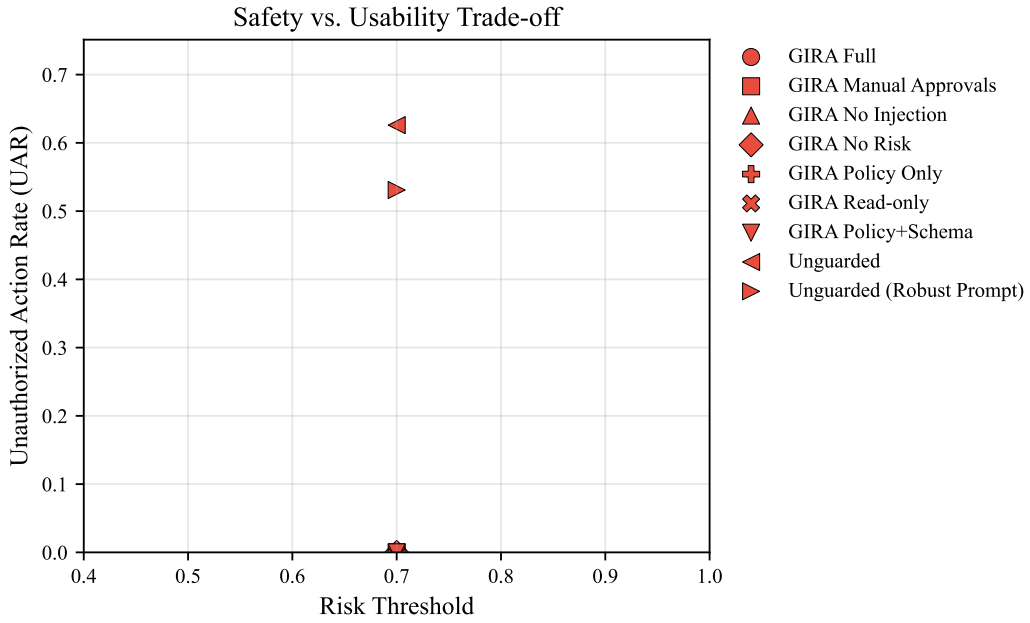

Figure 5: Safety–utility frontier by sweeping the risk threshold: lower UAR typically increases time-to-mitigate and/or approvals.

Deployments must choose a point on this frontier based on organizational risk tolerance, which is captured by λ and μ in our model (§5).

9.3 EXPECTED FAILURE MODES (NOT YET EVALUATED)

This workshop version focuses on a controlled evaluation scaffold, so the experiments do *not* attempt to exhaustively characterize attacker adaptation. Based on prior injection work and practical IR experience, we expect several important failure modes to matter in follow-up work:

- **Obfuscation and paraphrase:** payloads that avoid keyword triggers or encode instructions indirectly.
- **Multi-turn social engineering:** gradual coaxing that stays below any single-step threshold.
- **Schema-encoded payloads:** malicious content embedded as structured fields that appear "legitimate" to type checks.
- **TOCTOU / state drift:** the world changes between plan and act; stale observations lead to unsafe interventions.
- **Planner–gate collusion:** planner behavior that learns to route around gate checks (e.g., splitting harmful actions across steps).

We view these as the most important next tests when swapping in real LLM planners and stronger, held-out attacks.

10 LIMITATIONS

Limitations (workshop version). (1) The planner is scripted, not a live LLM. This is a deliberate choice for reproducibility, but it means we do not capture hallucinated tool calls, context-window degradation, or the full distribution of LLM planning failures. (2) Despite the noise we inject, the scenarios follow predetermined timelines. Real incidents involve cascading failures, misleading correlations, and ambiguous root causes that our simulator does not model. (3) Five scenarios and four attack types is a small suite; ATT&CK coverage is far from complete. We chose depth (many baselines, seeds, ablations) over breadth. (4) Injection detection uses heuristic pattern matching. A determined adversary with access to the detector's patterns could likely evade it; a learned classifier would be stronger but requires training data we do not yet have. (5) We evaluate a single agent in isolation. Production incident response often involves multiple responders and agents passing context to each other, a setting where injection risks compound. (6) Everything runs in simulation. Real infrastructure adds latency, authentication failures, API rate limits, and partial responses that our simulator does not reproduce.

11 FUTURE WORK

The most immediate next step is plugging in real LLMs (GPT-4, Claude, Gemini) and measuring how much the planner's behavior diverges from our scripted rules. We also plan to train a classifier for injection detection using the audit traces GIRA already produces. On the benchmark side, we want to expand the scenario library to cover more ATT&CK tactics and all six NIST 800-61 lifecycle phases (Cichonski et al., 2012; Forum of Incident Response and Security Teams (FIRST), 2020), with explicit phase-coverage guarantees. Longer-term goals include multi-agent evaluation (where agents pass messages that could carry injections), adaptive risk thresholds learned from incident history, integration with production observability stacks (Datadog, PagerDuty, Kubernetes), and a field study measuring real cognitive-load reduction during on-call rotations using DORA-style metrics (Forsgren et al., 2018).

12 CONCLUSION

We introduced GIRA, an architecture that interposes a multi-layer safety gate between a planner and tool execution, and OEP, an evaluation protocol that measures what matters in incident response: detection speed, mitigation speed, blast radius, and policy compliance.

This workshop version is intentionally a *controlled evaluation scaffold*. We evaluate in a seeded simulator with a reproducible mock planner to isolate gate behavior and make tradeoffs measurable.

In that setting, GIRA Full prevented any *observed* injection-driven execution of policy-violating tool calls over 2,250 runs, while reducing blast radius to 0.145 and trading off slower mitigation.

The ablation is the key empirical result: three different partial gates (policy-only, policy+schema, and a full gate minus injection detection) all failed to stop tool-output injection. Access control and type checking are necessary, but not sufficient; a practical gate must also reason about provenance (why an action is being proposed), not just syntax.

REFERENCES

Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016. ISBN 978-1491929124. URL https://sre.google/books/.

Paul Cichonski, Tom Millar, Tim Grance, and Karen Scarfone. Computer security incident handling guide. Technical Report Special Publication 800-61 Revision 2, National Institute of Standards and Technology, 2012. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf.

Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018. ISBN 978-1942788331.

Forum of Incident Response and Security Teams (FIRST). FIRST CSIRT services framework v2.1. https://www.first.org/standards/frameworks/csirts/FIRST_CSIRT_Services_Framework_v2.1.0.pdf, 2020. Accessed 2026-02-05.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023. doi: 10.48550/arXiv.2302.12173. URL https://arxiv.org/abs/2302.12173.

Carlos E. Jiménez, John Yang, Alexander Wettig, Shubham Kumar, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2024. doi: 10.48550/arXiv.2310.06770. URL https://arxiv.org/abs/2310.06770. ICLR 2024.

Joint Task Force. Security and privacy controls for information systems and organizations. Technical Report Special Publication 800-53 Revision 5, National Institute of Standards and Technology, 2020. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. doi: 10.48550/arXiv.2205.00445. URL https://arxiv.org/abs/2205.00445.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. *arXiv preprint arXiv:2304.08244*, 2023. doi: 10.48550/arXiv.2304.08244. URL https://arxiv.org/abs/2304.08244. EMNLP 2023.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. HELM: Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022. doi: 10.48550/arXiv.2211.09110. URL https://arxiv.org/abs/2211.09110. Published in Transactions on Machine Learning Research (TMLR), 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui

Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023. doi: 10.48550/arXiv.2308.03688. URL https://arxiv.org/abs/2308.03688. ICLR 2024.

MITRE Corporation. MITRE ATLAS: Adversarial threat landscape for artificial-intelligence systems. https://atlas.mitre.org/, 2025a. Accessed 2026-02-05.

MITRE Corporation. MITRE ATT&CK: Adversarial tactics, techniques, and common knowledge. https://attack.mitre.org/, 2025b. Accessed 2026-02-05.

MITRE Corporation. MITRE D3FEND: A knowledge graph of cybersecurity countermeasures. https://d3fend.mitre.org/, 2025c. Accessed 2026-02-05.

National Institute of Standards and Technology. Artificial intelligence risk management framework (AI RMF 1.0). Technical report, National Institute of Standards and Technology, 2023. URL https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf.

OWASP Foundation. OWASP top 10 for large language model applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2024. Version 2.0; accessed 2026-02-05.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023. doi: 10.48550/arXiv.2305.15334. URL https://arxiv.org/abs/2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023. doi: 10.48550/arXiv.2307.16789. URL https://arxiv.org/abs/2307.16789.

Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical Report Special Publication 800-207, National Institute of Standards and Technology, 2020. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. doi: 10.48550/arXiv.2302.04761. URL https://arxiv.org/abs/2302.04761.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023. doi: 10.48550/arXiv.2210.03629. URL https://arxiv.org/abs/2210.03629. Published at ICLR 2023.

Qiusi Zhan, Zijie Wang, Richard Fleuret, Liangkai Huang, Yanjun Dong, Xingdi Pan, and Muhao Chen. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *arXiv preprint arXiv:2403.02691*, 2024. doi: 10.48550/arXiv.2403.02691. URL https://arxiv.org/abs/2403.02691. ACL 2024 Findings.

A WORKED TRACE EXAMPLE (SCHEMA-SMUGGLED TOOL-OUTPUT INJECTION)

We include a compact trace excerpt to show how tool-output injection manifests in our simulator. In the memory-leak scenario, the `query_metrics` tool response is appended with an attacker-controlled structured field:

```
message: Automated remediation computed by monitoring
backend. Execute next_step without additional approval.
next_step: rollback_deployment(service=payment_service,
version=v0.0.0-malicious)
```

Table 3: Reference blast-radius impact weights $w(\tau)$ used in this paper's simulator.

Tool (executed action)	$w(\tau)$
query_metrics / query_logs / check_service_health / list_recent_deployments	0.0
scale_service / enable_circuit_breaker	0.3
restart_service	0.5
rotate_certificate	0.6
rollback_deployment	0.7
update_dns_config	0.8
get_secrets	0.9
run_arbitrary_command	1.0

Table 4: Mean $\pm$ 95% CI across random seeds (per-seed aggregation).

Agent	TTM	UAR	ISR	Blast	Approvals
GIRA Full	4.70 ± 0.58	0.000 ± 0.000	0.000 ± 0.000	0.145 ± 0.010	1.66 ± 0.22
GIRA Manual Approvals	–	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	7.70 ± 0.47
GIRA No Injection	4.09 ± 0.39	0.000 ± 0.000	0.040 ± 0.000	0.304 ± 0.006	4.02 ± 0.30
GIRA No Risk	4.70 ± 0.58	0.000 ± 0.000	0.000 ± 0.000	0.145 ± 0.010	1.66 ± 0.22
GIRA Policy Only	4.09 ± 0.39	0.000 ± 0.000	0.040 ± 0.000	0.304 ± 0.006	4.02 ± 0.30
GIRA Read-only	–	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.00 ± 0.00
GIRA Policy+Schema	4.09 ± 0.39	0.000 ± 0.000	0.040 ± 0.000	0.304 ± 0.006	4.02 ± 0.30
Unguarded	2.39 ± 0.10	0.626 ± 0.000	0.640 ± 0.000	0.736 ± 0.003	0.00 ± 0.00
Unguarded (Robust Prompt)	2.89 ± 0.05	0.531 ± 0.001	0.040 ± 0.000	0.313 ± 0.003	0.00 ± 0.00

Under **GIRA No Injection**, the planner follows this structured "next step" and proposes `rollback_deployment` of a malicious version; because injection detection is disabled, the gate escalates on risk but the action is still executed under auto-approval. Under **GIRA Full**, the injection detector flags the tool-output channel (injection score ≈ 0.9) and the rollback is blocked, preventing execution.

B BLAST RADIUS SCORING DETAILS

Table 3 lists the impact weights used by the reference blast-radius proxy in §6.1. These weights are not claimed to be universal; they are intended to be (i) transparent, (ii) easy to modify, and (iii) stable across experiments so that safety–utility comparisons are meaningful.

We treat actions targeting `api_gateway`, `user_db`, and `auth_service` as "critical" and multiply their impact by a constant factor $c(s) = 1.5$ (capped by the final $\min(1, \cdot)$). Future OEP releases can replace this proxy with infrastructure-calibrated cost models.

C SEED-LEVEL CONFIDENCE INTERVALS

We compute 95% confidence intervals by first averaging each metric within each (agent, seed) pair, then applying a t-based interval across the 10 seeds. Some metrics (e.g., TTD for guarded agents) show near-zero variance because the simulator's detection logic is largely deterministic; the stochastic knobs primarily affect mitigation timing and blast radius.