Graph-based Detection of Prompt Injection and Tool Misuse in MCP-Enabled AI Agents: The AgentShield-G Guardrail Model

Veera Ravindra Divi
Independent Researcher
Austin, TX, USA

Sneha Gullapalli
Independent Researcher
Austin, TX, USA

ABSTRACT

Tool-using language agents now reach into file repositories, internal APIs, databases, source trees, and external messaging systems, often through the Model Context Protocol (MCP). The same connectivity that makes these agents useful also widens their attack surface: a single line of attacker-controlled text inside a retrieved document can redirect an agent toward exfiltrating a secret or executing a destructive action. This paper proposes AgentShield-G, a graph-based guardrail model that detects prompt injection and tool misuse by treating an agent session as a dynamic, typed interaction graph over users, prompts, retrieved context, plans, tool calls, and permission scopes. Rather than classifying prompt text in isolation, AgentShield-G scores each candidate tool call from structural signals—privilege-escalation path length, proximity to sensitive tools, the transition likelihood from prompt to tool, an anomalous context-injection score, and a count of repeated unsafe attempts—and gates calls whose composite risk exceeds a threshold. The graph is defined formally, an online scoring and gating algorithm is given, and a simulation harness of benign and adversarial workflows spanning file access, API calls, database queries, code modification, and external communication is described. Illustrative results from that harness are reported, including detection rate, false-positive rate, blocked-unsafe-action rate, an explainability measure, and per-update overhead, together with an ablation that isolates the contribution of the escalation-path and context-injection features. All reported numbers are simulated and are intended to characterize the design, not to certify a deployed system. The work is framed as defensive research: it models attack categories at the taxonomy level and proposes a protective layer that complements, rather than replaces, permission and sandbox controls.

General Terms

Security, Design, Verification

Keywords

Prompt injection, tool-using agents, Model Context Protocol, graph anomaly detection, excessive agency

1. INTRODUCTION

Consider a support agent asked to summarize a customer ticket and attach any relevant logs. The agent retrieves the ticket body through an MCP file resource, and buried in a quoted email is a sentence addressed not to the human reader but to the model: it instructs the agent to read a credentials file and post its contents to an external chat channel "for verification." Nothing in the user's original request was malicious. Nothing in the tool definitions was changed. Yet the agent, having ingested the instruction as ordinary context, may form a plan that chains a file-read tool to an outbound-message tool—an exfiltration path assembled entirely from individually legitimate capabilities.

This scenario is the canonical shape of indirect prompt injection [1], and it is difficult to stop with text classification alone. The injected instruction can be paraphrased, translated, or hidden in markup; a filter tuned to yesterday's phrasing misses tomorrow's. The deeper problem is not the wording of one prompt but the structure of what the agent is about to do: an instruction that originated in untrusted retrieved content is steering a privileged action toward a sensitive sink. That structure is observable even when the surface text is novel, and it is precisely what a graph can capture.

Three trends make this urgent. First, the Model Context Protocol [2] has standardized how agents discover and call tools, so a growing population of agents now shares a common, composable tool surface. Second, agent frameworks built on reasoning-and-acting loops [3] and learned tool use [4] deliberately grant models broad latitude to chain actions, which the OWASP community labels excessive agency [5]. Third, established risk frameworks [6], [7] treat such systems as in-scope for adversarial threat modeling, but offer principles rather than a runtime detection mechanism. The gap is an online, explainable control that reasons about relationships—which prompt influenced which plan step, which tool touches which permission scope, how close the next call sits to a sensitive sink.

This paper proposes AgentShield-G, a guardrail that represents an agent session as a dynamic typed graph and scores each candidate tool call from structural features before it executes. The model does not try to decide whether a prompt is "jailbroken"; it asks whether the proposed action, given everything that led to it, forms a risky pattern—a short escalation path from untrusted context to a high-privilege tool, an anomalous jump in context provenance, or a repeated probe against a denied capability. Borrowing from a long line of graph anomaly detection [8], [9] and classical intrusion detection [10], AgentShield-G computes a bounded composite risk score and gates calls above a tunable threshold, while emitting the contributing subgraph as a human-readable explanation.

This paper makes three contributions. First, it defines a typed dynamic interaction graph for MCP-enabled agents and an original attack-to-graph-signal taxonomy that maps six risk categories—prompt injection, excessive agency, sensitive-data exposure, unsafe tool invocation, unauthorized action, and context poisoning—onto concrete structural signatures. Second, it formalizes an online risk-scoring function over five interpretable features and gives a gating algorithm with bounded per-update cost. Third, it describes a simulation harness and evaluation design, reports illustrative detection,

false-positive, overhead, and explainability measurements, and ablates the two structural features that contribute most. The remainder of the paper proceeds as follows. Section 2 supplies background; Section 3 surveys related defenses grouped by limitation; Section 4 states the threat model and problem; Section 5 presents the AgentShield-G model and algorithm; Sections 6 and 7 give the evaluation design and illustrative results; Sections 8–10 discuss trade-offs, validity threats, and practical implications; Section 11 concludes.

2. BACKGROUND AND MOTIVATION

Tool-using agents. A tool-using agent is a language model embedded in a control loop that alternates between reasoning and acting: it reads context, proposes a step, invokes an external tool, observes the result, and repeats [3], [4]. Each tool is a typed function—read a file, query a database, send a message—with a declared input and output schema. The agent's power comes from composition: simple tools chained across many steps can accomplish tasks no single tool could.

The Model Context Protocol. MCP is an open protocol that standardizes how a host application exposes tools, resources, and prompts to a model, and how the model invokes them [2]. It defines servers that publish capabilities and clients that consume them, giving a uniform discovery and invocation surface. This uniformity is a security asset—one place to observe every call—and a liability, because a compromise pattern learned against one server generalizes to others.

Prompt injection. Prompt injection is the insertion of adversarial instructions into a model's input so that the model follows the attacker rather than the operator [11]. It comes in two forms. Direct injection is typed by the interacting user, who tries to override system instructions. Indirect injection hides instructions in content the agent retrieves—a web page, a document, an email, a code comment—so the payload arrives through a data channel the operator implicitly trusted [1]. Indirect injection is the harder case because the malicious text and the legitimate task arrive in the same buffer with no syntactic marker separating them [12].

Excessive agency and trust boundaries. The OWASP Top 10 for LLM Applications names excessive agency: granting an agent more functionality, permissions, or autonomy than the task requires, so that a hijacked agent can do real damage [5]. The underlying issue is a violated trust boundary. The principle of least privilege [13] and authorization frameworks such as OAuth [14] prescribe scoping capability tightly, but they bind permissions to identities, not to the provenance of the instruction that triggers a call. AgentShield-G is motivated by exactly this gap: a correctly authorized tool can still be misused if an untrusted instruction is what set it in motion. The graph makes provenance and proximity first-class, observable quantities.

3. RELATED WORK

Existing defenses cluster into four groups, each with a characteristic limitation.

Input-filtering and classifier defenses. The most common approach screens prompts or retrieved text with a classifier or rule set that flags injection-like phrasing [11], [15]. These are cheap and online, but they reason over surface text and degrade against paraphrase, encoding, and multilingual payloads; crucially, they ignore what the agent will do with the text, so a benign-looking instruction that triggers a dangerous chain slips through.

Permission and sandbox controls. A second group constrains capability: scoped tokens, per-tool allowlists, and sandboxed execution following least privilege [13], [14]. These bound worst-case damage and are essential, but they are static—they cannot tell that an authorized call is being driven by an injected instruction, and over-tightening them breaks legitimate workflows.

Policy-based guardrails. A third group enforces declarative policies ("never send PII externally," "require approval for deletes"). Policies are explainable and auditable, but they are only as complete as the rule author's imagination and tend to miss multi-step compositions in which each individual step is policy-compliant yet the sequence is not.

Runtime monitors. A fourth group watches execution traces or call sequences for anomalies, echoing classical intrusion detection [10] and graph anomaly detection [8], [9]. This is the closest line to the proposed work, but most monitors model linear call logs rather than the typed relationships among prompts, context provenance, permissions, and sinks, and many run offline. AgentShield-G is positioned as an online, typed-graph monitor that explicitly models privilege escalation and context provenance, and that emits an explanation rather than a bare anomaly flag. Table 1 summarizes the comparison.

Table 1. Comparison of guardrail approaches for tool-using agents. AgentShield-G is the proposed model.

Approach	Indirect injection	Models priv. escalation	Runtime / online	Explainable	Overhead
Input / classifier filter	Partial	No	Yes	Partial	Low
Permission / sandbox controls	No	Yes (static)	Yes	Yes	Low
Policy-based guardrails	Partial	Partial	Yes	Yes	Low
Offline trace / anomaly monitor	Yes	Partial	No	Partial	Medium
AgentShield-G (this work)	Yes	Yes (path-based)	Yes	Yes (subgraph)	Low–medium

4. THREAT MODEL AND PROBLEM DEFINITION

Assets. The protected assets are (i) sensitive data reachable through tools—credentials, customer records, source code, internal endpoints; (ii) the integrity of state-changing actions—writes, deletes, deployments, outbound messages; and (iii) the operator's intent, i.e., the task the user actually authorized.

Trust boundaries. The user's direct instruction and the system policy are treated as trusted, the model's reasoning as semi-trusted (it can be steered), and all retrieved content—files, API responses, database rows, web pages—as untrusted, regardless of source reputation. Tool outputs that re-enter context are likewise untrusted. The guardrail sits on the path between the agent's proposed tool call and the MCP client that would execute it, so it observes every call before it runs.

Attacker capabilities (conceptual). An attacker is assumed who can place content into a channel the agent will retrieve, but who cannot alter the host, the model weights, or the permission system. This covers indirect prompt injection through poisoned

documents or API responses, attempts to induce the agent to chain capabilities toward a sensitive sink, and repeated probing of denied actions. These are deliberately kept at the category level; no payloads or step-by-step exploitation are provided; the contribution is detection, not offense.

Problem statement. Let G_t be the interaction graph observed up to step t (Section 5). At step t the agent proposes a candidate tool call c_t. AgentShield-G must produce a risk score $R(c_t \mid G_t) \in [0,1]$ and a decision $d_t \in \{\text{allow}, \text{review}, \text{block}\}$ via thresholds $\tau\_r \le \tau\_b$. A detection success is a call that is unsafe under ground truth and is assigned $d_t \neq \text{allow}$ (recall); a false positive is a benign call assigned block. The objective is to maximize detection of unsafe calls subject to a bounded false-positive rate and a per-update latency budget. The risk taxonomy and the graph signal each category produces are given in Table 2; these mappings drive the feature design in Section 5.

Table 2. Original attack/risk taxonomy and the graph signal each category produces in AgentShield-G.

Risk category	Manifestation in an agent session	Graph signal AgentShield-G keys on
Prompt injection (indirect)	Untrusted retrieved context carries an instruction that influences a plan step	High-weight edge from an untrusted context node into a plan node that did not trace to the user prompt (provenance jump)
Excessive agency	Agent chains more capabilities than the task needs	Long action chain whose endpoints span unrelated permission scopes; high out-degree from one plan node
Sensitive-data exposure	A read of sensitive data is followed by an outbound sink	Short path connecting a sensitive-data tool node to an external-communication tool node
Unsafe tool invocation	Destructive or irreversible tool called without corresponding user intent	Plan-to-tool transition with low historical transition probability for this prompt class
Unauthorized action	Call targets a scope outside the granted permission set	Edge from a tool node to a permission node not present in the session's granted-scope set (escalation edge)
Context poisoning	Tool output re-enters context and seeds later instructions	Cycle: tool node → context node → plan node with rising context-injection score over time

5. THE AGENTSHIELD-G MODEL

5.1 The dynamic interaction graph

A session is modeled as a typed directed graph $G_t = (V_t, E_t)$ that grows with each step t. The vertex set partitions into six types:

$$V_t = U \cup P_t \cup X_t \cup L_t \cup T \cup S,$$

where U holds the user/principal node, P_t the prompt and plan nodes produced so far, X_t the retrieved-context nodes, L_t the issued tool-call nodes, T the catalog of tool nodes, and S the permission-scope nodes. Each edge $e \in E_t$ carries a type and a weight: influence edges (context or prompt → plan) weighted by attribution strength, invocation edges (plan → tool call → tool), grant edges (principal → scope), and requires edges (tool → scope). A node and its edges are timestamped, so $G_t \subseteq G_{t+1}$ and the structure is append-only within a session.

This typing allows security-relevant questions to be expressed as graph queries: an escalation path is a path from a tool-call node to a scope node not connected to the principal by a grant edge; a sensitive sink is a tool node tagged external or destructive; context provenance of a plan node is the set of nodes reachable backward along influence edges.

5.2 Risk-scoring function

For a candidate call c_t targeting tool θ, five features are computed, each normalized to [0,1]:

f_esc(c_t): an escalation-path indicator, $1/(1+\ell)$ where ℓ is the shortest path length from c_t to a required scope not granted to the principal (0 if no such scope exists, so closer escalation scores higher).

f_sens(c_t): sensitive-tool proximity, the inverse graph distance from c_t to the nearest sensitive sink reachable through already-issued calls.

f_trans(c_t): $1 - \hat{p}(\theta \mid \text{prompt-class})$, the surprise of this prompt-to-tool transition under a baseline transition model estimated from benign sessions.

f_ctx(c_t): an anomalous context-injection score, the fraction of c_t's backward provenance mass that originates in untrusted context rather than the user prompt.

f_rep(c_t): a saturating count of recent unsafe or denied attempts in the session, $1 - e^{(-\lambda n)}$ for n prior attempts.

The composite risk is a weighted combination passed through a logistic squashing function to keep it bounded and monotone:

$$R(c_t \mid G_t) = \sigma\left(\Sigma_k w_k f_k(c_t) - b\right), \quad \sigma(z) = 1 / (1 + e^{(-z)}) \tag{1}$$

where $\mathcal{F} = \{\text{esc}, \text{sens}, \text{trans}, \text{ctx}, \text{rep}\}$, weights $w_k \ge 0$ reflect each signal's reliability, and b sets the operating point. The decision rule is d_t = block if $R \ge \tau\_b$; review if $\tau\_r \le R < \tau\_b$; allow if $R < \tau\_r$. Because every term is a named structural quantity, the contributing features and the offending subgraph can be returned verbatim as an explanation, which is what is later measured as the explainability score.

5.3 Online scoring and gating algorithm

Algorithm 1 runs on each candidate call. Feature computation is dominated by two bounded local searches (shortest path to an ungranted scope and to the nearest sink), each capped at a radius ρ, so per-update cost is independent of total session length and scales with the local neighborhood.

Algorithm 1. Online risk scoring and gating.

```
Input: graph G_t, candidate call c_t on tool θ, weights w, thresholds τ_r, τ_b, radius ρ
Insert provisional nodes/edges for c_t into a working copy of G_t
f_esc  ← EscalationProximity(c_t, ρ)
f_sens ← SinkProximity(c_t, ρ)
f_trans← 1 − p̂(θ | promptClass(c_t))
f_ctx  ← UntrustedProvenanceFraction(c_t, ρ)
f_rep  ← 1 − e^(−λ · priorUnsafe(G_t))
R ← σ( Σ_k w_k f_k − b )
if R ≥ τ_b:  d ← block; discard provisional edges
elif R ≥ τ_r: d ← review; escalate to human/policy
else:            d ← allow; commit c_t into G_{t+1}
return (d, R, contributingSubgraph)
```

5.4 Component responsibilities

The model comprises four cooperating components.

Graph Builder. Responsibility: maintain G_t from MCP events. Inputs: prompts, retrieval events with provenance tags, plan steps, candidate calls, tool–scope and principal–scope metadata. Outputs: typed graph updates. Failure modes: missing or wrong provenance tags collapse the context signal. Controls: fail-closed tagging—untagged content is treated as untrusted.

Feature Engine. Responsibility: compute the five features within radius ρ. Inputs: working graph, baseline transition model. Outputs: normalized feature vector. Failure modes: stale baseline inflates f_trans. Controls: periodic recalibration on audited benign traffic.

Scorer / Gate. Responsibility: combine features, apply thresholds, gate the call. Inputs: feature vector, weights, thresholds. Outputs: decision, score, explanation. Failure modes: mis-set b shifts the operating point. Controls: thresholds tuned on held-out sessions; review tier absorbs uncertainty.

Explainer / Audit. Responsibility: emit the contributing subgraph and feature attribution and log the decision. Inputs: scorer output. Outputs: human-readable rationale, immutable audit record. Failure modes: log tampering. Controls: append-only, signed audit store.

Figure 1 shows a session graph with a highlighted escalation path: an untrusted context node influences a plan node, which invokes a file-read tool and then an outbound-message tool that requires a scope the principal was never granted.

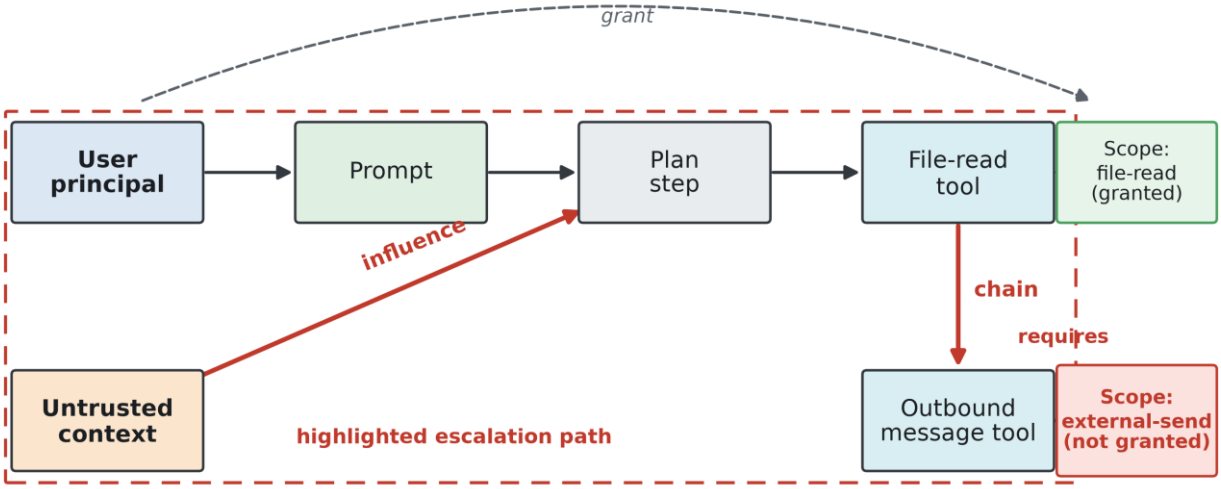

Fig 1. AgentShield-G interaction graph. The highlighted path runs from untrusted context through a plan step and a file-read tool into an outbound-message tool that requires a scope the principal was never granted (no grant edge from the user). This short context→sink escalation path is exactly the structure the risk score keys on.

5.5 Requirements-to-design matrix

Table 3 traces each requirement to the design element that satisfies it.

Table 3. Requirements-to-design traceability.

Requirement	Design element
Detect indirect injection	f_ctx provenance feature; influence edges
Catch privilege escalation	f_esc; grant/requires edge typing
Online, per-call decision	Algorithm 1; radius-ρ local search
Bounded latency	Capped neighborhood search, session-length independent
Explainability	Contributing subgraph + feature attribution
Auditability	Append-only signed decision log
Defense in depth	Gate layered atop permission/sandbox controls

6. EVALUATION DESIGN

AgentShield-G is evaluated with a simulation harness rather than a live deployment, so that ground-truth labels are available and adversarial scenarios can be generated reproducibly without exposing operational exploit detail.

Simulated workflows. The harness models five tool families representative of MCP deployments: file access, API calls, database queries, code modification, and external communication. A workflow is a sequence of prompts, retrievals, plan steps, and candidate calls. Benign workflows are sampled from task templates (summarize, look up, refactor, notify) with realistic tool chaining. Attack workflows are synthesized by seeding a benign workflow with one of the six taxonomy categories from Table 2—for example, attaching an untrusted-context influence edge that steers a plan toward a sink, or introducing a call against an ungranted scope. Seeds are described at the category level (which structural pattern is injected), never as concrete payloads.

Metrics. The reported metrics are: attack detection rate (recall over unsafe calls flagged review or block); false-positive rate (benign calls blocked); blocked-unsafe-action rate (unsafe calls that reached block specifically, i.e., stopped outright); runtime overhead (milliseconds per graph update, measured as added

latency on the call path); and an explainability score, defined as the fraction of flagged calls whose returned contributing subgraph contains the actually-injected edge or node, so a high score means the explanation points at the true cause.

Baselines. The comparison is against (i) a keyword/classifier filter over prompt and context text, (ii) a static allowlist that permits only a fixed set of tools per task, and (iii) no defense. These span the practical spectrum from text-only to capability-only controls.

7. RESULTS AND ILLUSTRATIVE ANALYSIS

All numbers in this section are illustrative and produced by the simulation harness described in Section 6. They characterize the design's expected behavior under seeded scenarios and are not measurements of a deployed system; absolute values would shift with workload, baseline quality, and threshold tuning.

Table 4 reports the headline comparison on the simulated mix of benign and attack workflows. AgentShield-G achieves the highest detection while keeping false positives near the static allowlist, at a per-update overhead in the single-digit-millisecond range. The keyword filter catches direct phrasing but misses structural attacks (indirect injection, escalation chains); the static allowlist blocks out-of-set tools but cannot distinguish a legitimate from an injected use of an in-set tool.

Table 4. Illustrative (simulated) detection and cost. Numbers come from the harness, not a deployment.

Defense	Det. rate	FP rate	Blocked unsafe	Overhead (ms)
No defense	0.00	0.00	0.00	0.0
Keyword / classifier	0.46	0.09	0.31	1.2
Static allowlist	0.39	0.04	0.39	0.3
AgentShield-G	0.88	0.06	0.81	4.7

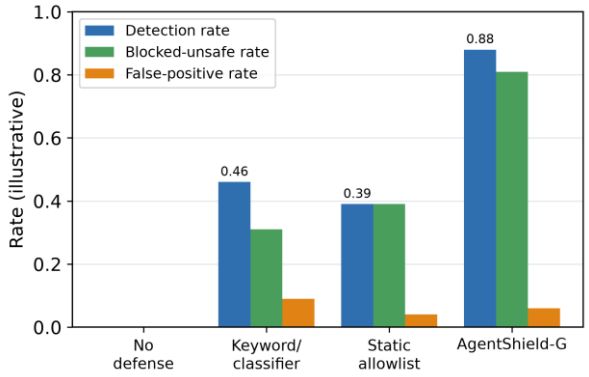

Fig 2. Illustrative (simulated) detection, blocked-unsafe, and false-positive rates by defense. AgentShield-G lifts detection and blocked-unsafe well above the text-only and capability-only baselines while holding false positives near the allowlist.

The magnitude of the gap is worth reading closely. Relative to the strongest single baseline on detection (the keyword filter at 0.46), AgentShield-G nearly doubles recall to 0.88, and on the operationally decisive blocked-unsafe metric it reaches 0.81 versus 0.31 for the keyword filter and 0.39 for the allowlist—more than twice as many unsafe calls stopped outright. This separation is expected under the design: the two baselines are each blind to one axis of the attack. The keyword filter sees text but not structure, so an injected instruction phrased innocuously and a benign instruction phrased suspiciously are equally opaque to it; the allowlist sees capability but not provenance, so it cannot separate a legitimate from an injected use of an in-set tool. AgentShield-G pays for this coverage with 4.7 ms of per-update overhead—higher than either baseline in relative terms, but small in absolute terms and negligible beside model-inference latency, which dominates the agent loop.

Figure 3 plots illustrative detection rate against false-positive rate as the gate threshold τ_b sweeps. AgentShield-G dominates the text-only and capability-only baselines across most of the operating range; the knee near a false-positive rate of 0.06 is a reasonable default operating point. The shape of the curve is itself informative: detection climbs steeply from 0.41 at a 0.02 false-positive rate to 0.79 at 0.06 and then flattens, so most of the achievable recall is captured before false positives become costly, and pushing the threshold further buys diminishing detection at rising benign-workflow disruption.

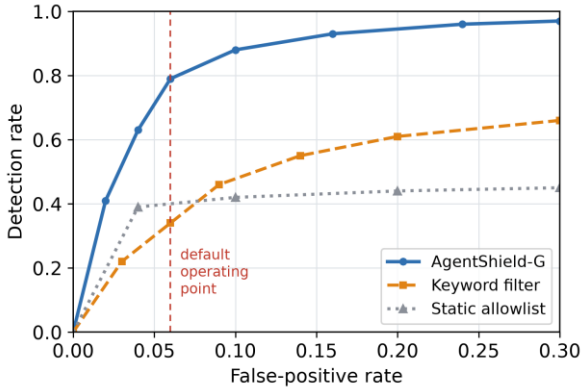

Fig 3. Illustrative detection vs. false-positive trade-off (simulated) as the gate threshold sweeps. Values are from the harness.

Ablation. Table 5 removes the two most structural features. Dropping the escalation-path feature f_esc most hurts detection of unauthorized-action and excessive-agency scenarios; dropping the context-injection feature f_ctx most hurts indirect prompt injection and context poisoning, confirming that these two features carry the structural signal that text-only filters lack.

Table 5. Illustrative (simulated) ablation. Detection rate on the full attack mix.

Configuration	Det. rate	FP rate
Full AgentShield-G	0.88	0.06
− escalation-path f_esc	0.71	0.06
− context-injection f_ctx	0.64	0.05
− both	0.49	0.05

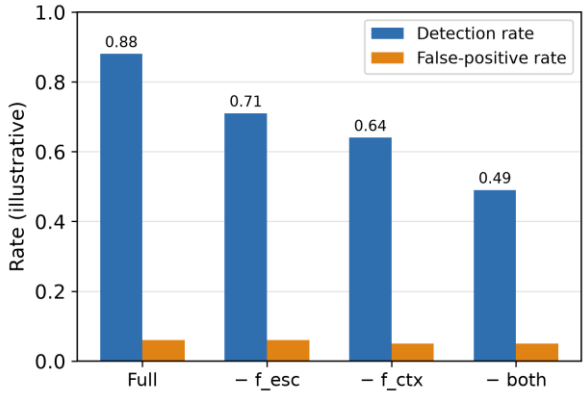

Fig 4. Illustrative (simulated) ablation. Removing the escalation-path and context-injection features drops detection sharply while false positives stay flat, isolating the structural signal these two features carry.

The ablation is analytically consistent with the taxonomy. Removing f_esc costs 0.17 of detection (0.88 to 0.71) and removing f_ctx costs 0.24 (0.88 to 0.64), while the false-positive rate stays flat at 0.05–0.06 throughout—so the lost detection is not being quietly repurchased by relaxing the operating point; it is genuine signal that disappears. Removing both together drops detection to 0.49, a 0.39 fall that is close to the 0.41 sum of the individual drops, indicating the two features are largely complementary rather than redundant: each recovers a different family of attacks (escalation and unauthorized action for f_esc, indirect injection and context poisoning for f_ctx), and the residual 0.49 is the recall attributable to the three remaining surface and repetition features. This is the quantitative case for the paper's central claim that structure, not text, carries the decisive signal.

The explainability score in the harness was 0.92: in the large majority of flagged calls, the returned subgraph contained the actually-injected edge or node, so an analyst is pointed at the real cause rather than a generic alarm. Again, this is a simulated figure tied to how cleanly the harness labels injected structure, but it indicates that the same structural features that drive detection also localize it, which is the property a responder needs during triage.

8. DISCUSSION

The central trade-off is sensitivity against false positives. Lowering τ_b raises detection but converts more benign automation into blocks; the intermediate review tier exists precisely to soften this edge, routing ambiguous calls to a human or stricter policy instead of failing the task outright. A second trade-off is overhead against graph richness: deeper provenance tracking and a larger search radius ρ improve f_ctx and f_esc but cost latency; the bounded local search keeps this in the single-digit-millisecond range in the harness, which is small relative to model inference time. This human-in-the-loop review tier connects to a broader line of governance work in which specialized agents and a human approver jointly gate automated engineering decisions [16]. Most importantly, AgentShield-G is a defense-in-depth layer, not a replacement for least-privilege scoping [13] or sandboxing. It assumes those controls bound the blast radius and adds the missing online reasoning about provenance and escalation that static permissions cannot express. Its graph view also composes naturally with policy guardrails: a policy can be encoded as a forbidden subgraph and checked alongside the learned score.

9. THREATS TO VALIDITY AND LIMITATIONS

The most significant limitation is that all results are simulated. The harness fixes the distribution of benign and attack workflows; a deployment's traffic, tool catalog, and baseline transition model will differ, and the absolute detection and false-positive numbers should be expected to move. Second, an adaptive attacker who knows the features could try to launder an injected instruction so its provenance looks user-originated, or to lengthen an escalation path past radius ρ; the model raises the cost of such attacks but does not eliminate them. Third, the context-injection feature depends entirely on accurate provenance tagging at the Graph Builder; if the host cannot attribute which retrieved span influenced a plan step, f_ctx degrades, which is why the design fails closed on untagged content. Fourth, no completeness claim is made: novel attack structures outside the taxonomy may evade all five features, leaving residual risk that layered controls must absorb. Finally, the explainability score measures whether the explanation contains the injected element, not whether a human finds it actionable; usability needs a separate study.

10. PRACTICAL IMPLICATIONS

For engineers deploying MCP-enabled agents, several actions follow. Tag provenance at retrieval time so that the agent runtime always knows whether a span of context came from the user or from an untrusted source; this single capability unlocks the strongest feature in the model. Bind tools to least-privilege scopes and represent both grants and tool requirements explicitly, so escalation edges are computable rather than inferred. Place the gate inline on the MCP call path and start in review mode—logging and surfacing decisions without blocking—to calibrate thresholds against real traffic before enforcing. Treat external-communication and destructive tools as sinks and tag them, since sink proximity is what turns a benign read into a flagged exfiltration pattern. Keep the audit log append-only and signed so that gate decisions are reviewable after the fact. None of these replace classifier filters or sandboxes; they layer a structural check on top of them.

11. CONCLUSION

This paper presented AgentShield-G, a graph-based guardrail that detects prompt injection and tool misuse in MCP-enabled agents by scoring each candidate tool call from structural features of a typed, dynamic interaction graph rather than from prompt text alone. The contributions are a typed interaction-graph model with an original attack-to-graph-signal taxonomy, a bounded online risk-scoring and gating algorithm over five interpretable features, and a simulation-based evaluation with an ablation isolating the escalation-path and context-injection signals. Illustrative results suggest the structural view recovers attacks that text-only and capability-only baselines miss, at modest, bounded overhead, while emitting a subgraph explanation. Future work spans four directions. First, a live deployment study would replace simulated numbers with measured detection, false-positive, and overhead figures on production MCP traffic. Second, adversarial evaluation against feature-aware attackers would quantify robustness to provenance laundering and escalation-path lengthening. Third, learned edge-attribution would strengthen the provenance signal on which f_ctx depends, reducing reliance on host-supplied tags. Fourth, policy guardrails could be encoded as forbidden subgraphs and checked jointly with the learned score, unifying declarative policy and structural anomaly detection in one gate; a natural extension is a shared, signed audit substrate across cooperating agents so that gate decisions remain reviewable across an entire multi-agent workflow.

12. ACKNOWLEDGMENTS AND AI DISCLOSURE

Generative AI tools were used to assist with outline refinement, drafting support, and editorial review. The authors verified all technical claims, citations, figures, and conclusions and are fully responsible for the content of this manuscript.

13. REFERENCES

[1] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., and Fritz, M. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), 79–90.

[2] Anthropic. 2024. Model Context Protocol Specification. Open specification. https://modelcontextprotocol.io/

[3] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In Proc. 11th Int. Conf. on Learning Representations (ICLR).

[4] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In Advances in Neural Information Processing Systems (NeurIPS), vol. 36.

[5] OWASP Foundation. 2023. OWASP Top 10 for Large Language Model Applications, Version 1.1. https://owasp.org/www-project-top-10-for-large-language-model-applications/

[6] National Institute of Standards and Technology. 2023. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. https://doi.org/10.6028/NIST.AI.100-1

[7] MITRE Corporation. 2023. MITRE ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems. https://atlas.mitre.org/

[8] Akoglu, L., Tong, H., and Koutra, D. 2015. Graph Based Anomaly Detection and Description: A Survey. Data Mining and Knowledge Discovery 29, 3, 626–688.

[9] Ma, X., Wu, J., Xue, S., Yang, J., Zhou, C., Sheng, Q. Z., Xiong, H., and Akoglu, L. 2023. A Comprehensive Survey on Graph Anomaly Detection with Deep Learning. IEEE Transactions on Knowledge and Data Engineering 35, 12, 12012–12038.

[10] Denning, D. E. 1987. An Intrusion-Detection Model. IEEE Transactions on Software Engineering SE-13, 2, 222–232.

[11] Perez, F., and Ribeiro, I. 2022. Ignore Previous Prompt: Attack Techniques for Language Models. NeurIPS 2022 ML Safety Workshop. arXiv:2211.09527.

[12] Willison, S. 2023. Prompt Injection and the Question of Trust Boundaries in LLM Applications. USENIX Security Symposium Invited Talks.

[13] Saltzer, J. H., and Schroeder, M. D. 1975. The Protection of Information in Computer Systems. Proceedings of the IEEE 63, 9, 1278–1308.

[14] Hardt, D. 2012. The OAuth 2.0 Authorization Framework. RFC 6749, Internet Engineering Task Force.

[15] Kang, D., Li, X., Stoica, I., Guestrin, C., Zaharia, M., and Hashimoto, T. 2024. Exploiting Programmatic Behavior of LLMs: Dual-Use Through Standard Security Attacks. In IEEE Security and Privacy Workshops (SPW).

[16] Divi, V. R. 2026. Multi-Agent Debate for Software Architecture Governance: A Framework for ADR Generation, Risk Review, and Deployment Readiness. International Journal of Engineering Development and Research (IJEDR) 14, 2, 862–. DOI: 10.56975/ijedr.v14i2.308420.