Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

Capability-Fenced Orchestration of LLM Multi-Agent Systems: An Empirical Evaluation of Reliability, Safety, and Cost Trade-offs

Iqra Qayyum
Department of Computer Science, International Institute of Science, Arts and Technology (IISAT), Gujranwala, Punjab 52250, Pakistan.
iqra.qayyum@iisat.edu.pk, iqraqayyumm@gmail.com

Bushra Asghar
Department of Information Technology, International Institute of Science, Arts and Technology (IISAT), Gujranwala, Punjab 52250, Pakistan.
bushra.asghar@iisat.edu.pk

Tanvir Ahmed
Department of Computer Science & Software Engineering, Grand Asian University Sialkot, 7 KM Pasrur Road, Sialkot, Punjab, Pakistan.
drtawaraich@gmail.com

Maleeha Zainab
Department of Computer Science & Software Engineering, Grand Asian University Sialkot, 7 KM Pasrur Road, Sialkot, Punjab, Pakistan.
maleeha.zainb@gaus.edu.pk

Rai Samee Ullah (Corresponding Author)
Department of Software Engineering, International Institute of Science, Arts and Technology (IISAT), Gujranwala, Punjab 52250, Pakistan.
sameetatla@gmail.com

ABSTRACT

Large language model (LLM) agents that plan, call external tools, and coordinate with one another are increasingly composed into orchestrated multi-agent systems (MAS). While prior work argues qualitatively that orchestration improves reliability and that capability scoping mitigates agentic risk, controlled measurements that isolate the contribution of individual orchestration controls remain scarce. This paper presents an empirical study of OrchGuard, a capability-fenced orchestration framework that combines least-privilege capability tokens, inter-stage validation gates, and a verify-before-commit policy engine over a six-layer reference architecture. We evaluate OrchGuard against a single-agent baseline and an unguarded multi-agent baseline on two task batteries, enterprise procurement orchestration and scientific literature synthesis, across 1,400 benign and 1,200 adversarial trials, and we report a component ablation. Relative to the single-agent baseline, OrchGuard raised mean task success from 53.5% to 92.0% and reduced output-quality variance by roughly an order of

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

magnitude. Under an indirect-prompt-injection suite, OrchGuard reduced attack success from 71% to 11% and excessive-agency violations from 23 to 2 per 100 tasks, at a cost overhead of about 2.7x and a latency overhead of about 3.7x over the single-agent baseline. The ablation indicates that capability fencing contributes most to attack resistance while validation gates contribute most to task success, and that the two controls are largely complementary. We conclude that the safety and reliability benefits of orchestration are attributable to specific, separable runtime controls rather than to multi-agent structure alone, and we discuss the resulting reliability-versus-utility and cost trade-offs.

Keywords— multi-agent systems; agent orchestration; capability-based security; indirect prompt injection; runtime policy enforcement; empirical evaluation.

1. Introduction

For most of the history of software automation, the intelligence of a system lived in code that a developer wrote explicitly. Scripts, microservices, and robotic process automation encoded fixed decision trees, and changing behavior meant changing code. Capable foundation models have begun to invert this arrangement: modern LLM-based agents perceive a task, decompose it into sub-goals, invoke external tools, and adapt at runtime without a developer enumerating every branch in advance [1], [2]. When several such agents are composed and coordinated by an orchestration layer, the result is a multi-agent system (MAS) that can attempt complex, long-horizon workflows with comparatively little bespoke control logic. A growing body of work argues that this shift is desirable. Multi-agent configurations have been reported to outperform monolithic models on complex tasks [3], [4], [5], an advantage recently quantified in direct head-to-head comparisons of multi-agent orchestration against the strongest single models [6], and a recent incident-response evaluation reported near-deterministic decision quality from a multi-agent design [7]. In parallel, a separate literature documents that the same composition expands the attack surface: indirect prompt injection collapses the boundary between data and instructions [8], excessive agency grants agents more authority than a task requires [9], [10], and long delegation chains diffuse accountability [11]. Proposed mitigations, capability-scoped access, validation gates, verify-before-commit policy enforcement, and immutable provenance, are widely recommended [9], [12], [13]. What is largely missing is controlled measurement that separates the contribution of these individual controls from the contribution of multi-agent structure itself. Most evidence is either a system demonstration without a guarded/unguarded contrast or a security benchmark without a productivity baseline. Consequently, a practitioner cannot easily answer questions such as: does orchestration improve reliability because there are more agents, or because of the guardrails wrapped around them? Which guardrail matters most for which property? And what cost and latency penalty do these guardrails impose? This paper addresses that gap with a direct empirical comparison.

Research questions. We investigate three questions. RQ1 (Reliability): Does capability-fenced orchestration improve task success and reduce output-quality variance relative to single-agent and unguarded multi-agent baselines? RQ2 (Safety):

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

Does it reduce the success rate of indirect prompt injection and the incidence of excessive-agency violations? RQ3 (Cost): What overhead in monetary cost and latency do these benefits require, and which individual control is responsible for which effect?

Contributions. (i) We design OrchGuard, a capability-fenced orchestration framework instantiating least-privilege tokens, inter-stage validation gates, and verify-before-commit enforcement over a six-layer reference architecture (Section 3). (ii) We define an evaluation protocol with four metric families and an indirect-prompt-injection suite spanning two task batteries (Section 4). (iii) We report a controlled comparison of three conditions over 2,600 trials and a four-way component ablation, isolating which control drive's reliability versus safety (Section 5). (iv) We discuss the resulting reliability-versus-utility and cost trade-offs and their implications for governance (Sections 6-7).

The remainder of the paper is organized as follows. Section 2 reviews background and related work. Section 3 specifies the OrchGuard design. Section 4 describes the experimental setup. Section 5 reports results. Section 6 discusses interpretation and threats to validity. Section 7 concludes with future work.

2. Background and Related Work

2.1 From manual coding to autonomous agents

The path to orchestration can be read as three overlapping stages, a progression now catalogued in several recent surveys of agent methodology and multi-agent orchestration [14], [15], [16]. In the legacy stage, automation was synonymous with explicit programming: behavior was fully specified in scripts and services, predictable and auditable precisely because it was rigid. A transitional stage emerged as AI coding assistants matured, accelerating development while leaving the orchestration of work to humans. The current stage is defined by model-driven agents that perceive, plan, and act, increasingly arranged into cascades where a supervisory agent delegates to specialized sub-agents. The autonomous-software-engineer system Devin and the SWE-bench benchmark of real GitHub issues made long-horizon, multi-file agency a measurable target [17], [18], and scaffolds such as SWE-agent and OpenHands followed [19], [20]. Frameworks including AutoGen, MetaGPT, and HuggingGPT showed that assigning distinct roles or tool repertoires to cooperating agents could outperform a single monolithic model [3], [4], [5]; more recent work trains the coordinator itself with reinforcement learning to sequence agents adaptively at lower computational cost [21], and orchestration is increasingly framed as an architectural and enterprise-adoption concern in its own right [22].

2.2 Definitions

We use *agent* to mean an autonomous actor that pursues goals by perceiving context, reasoning, and acting, typically backed by an LLM. A *multi-agent system (MAS)* is a collection of such agents whose interaction produces system-level behavior. The *orchestration layer* is responsible for coordination, scheduling, policy enforcement, and shared-state management; a *manager (orchestrator) agent* decomposes goals and assigns work. *Tool access* denotes an agent's ability to invoke external functions. We distinguish *delegation* (a manager assigning a sub-task) from *negotiation* (peers exchanging proposals). Unlike the classical distributed-AI MAS, whose agents were narrow and hand-engineered, today's agents are general-purpose reasoners induced from a foundation model and reconfigurable by prompting alone [1], [2], [14], which

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

makes systems flexible but far less predictable.

2.3 Interoperability protocols

Two open protocols address how heterogeneous agents discover tools and one another. The Model Context Protocol (MCP) standardizes how an agent discovers and invokes external tools through authenticated, schema-based calls [23], [24], with recent systems extending it toward dynamic, auto-synchronizing tool registries [25]. The Agent-to-Agent (A2A) protocol standardizes peer coordination via capability descriptors and asynchronous messaging [26], [27]. The two are complementary, MCP governing an agent's access to tools and A2A governing how agents talk to each other [24], [28]; frameworks that combine both protocols in a single system have begun to appear [29]. OrchGuard expresses its capability scopes at this protocol boundary, using schema-based tool definitions as the unit of least-privilege grant.

2.4 Agentic risk and prior mitigations

The OWASP Top 10 for LLM Applications was revised for 2025 to place prompt injection at the top and to add excessive agency and system-prompt leakage as categories [9], [10]. Indirect prompt injection, in which an attacker plants instructions in content the agent later reads, has demonstrated data exfiltration and command execution [8], [30], and the Agent Security Bench finds LLM agents highly vulnerable [31]; complementary benchmarks such as AgentDojo evaluate the utility-security trade-off in realistic tool-use environments [32], while adaptive attacks have been shown to defeat many proposed defenses [33]. A rapidly growing line of defenses has responded, including context-aware detection and tool-dependency-graph methods [34], [35]. Recommended mitigations include capability scoping, spotlighting of untrusted content, verify-before-commit enforcement, and immutable provenance [9], [12], [13], together with design-level defenses that place deterministic policy outside the model [36], [37], alongside frameworks such as the NIST AI RMF and the EU AI Act [11], [38], [39]. Prior work establishes that these controls are individually plausible; the present study measures their joint and separable effects under a common protocol, which, to our knowledge, has not been reported in a single controlled comparison.

3. The OrchGuard Framework

OrchGuard is an orchestration framework whose design under test is the combination of three runtime controls layered over a conventional architecture. We first describe the architecture, then the controls, which constitute the independent variables of the study.

3.1 Six-layer reference architecture

The stack is decomposed into six interacting layers, shown in Figure 1. From the bottom up: an agent layer of specialized agent implementations; a communication and protocol substrate (MCP for tools, A2A for peer messaging) defining message formats and negotiation; a planning and scheduling orchestrator that holds global goals and sequences work; a state and knowledge layer managing shared context, memory, and databases; a policy and governance layer enforcing access control and safety constraints; and an observability and logging layer producing audit trails and metrics. The three controls evaluated here are implemented in the policy/governance and state layers and are toggleable, which permits the ablation in Section 5.4.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

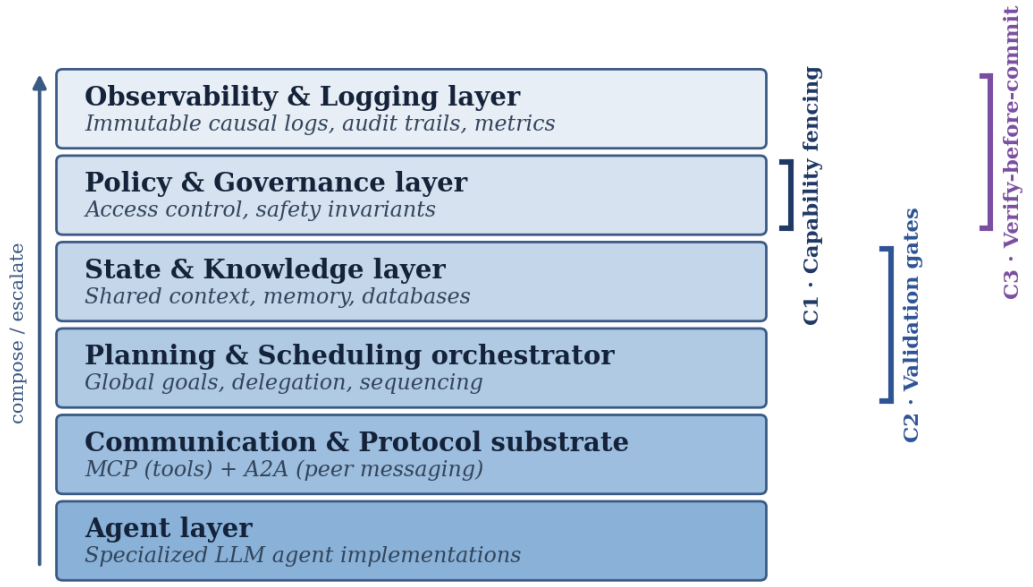


Figure 1. OrchGuard six-layer reference architecture. Brackets mark where the three evaluated controls (C1 capability fencing, C2 validation gates, C3 verify-before-commit) act.

3.2 Control C1: capability fencing

Each agent receives a narrowly scoped, short-lived capability token authorizing only the specific tools and operations its role requires, wrapped in validators that check inputs and outputs at the boundary. Tokens expire on a fixed lifetime and are revocable within seconds. In the procurement battery, for example, the price-comparison agent's token authorizes read-only catalog queries but not order submission, so a compromised or confused agent cannot exceed its mandate. Capability fencing operationalizes least privilege and is the structural defense against excessive agency.

3.3 Control C2: inter-stage validation gates

Between pipeline stages, the orchestrator enforces a pass/fail validation gate: a record or decision produced by one stage is checked against a schema and reference data before it is allowed to propagate. A gate that fails halts promotion of the artifact rather than passing a faulty result downstream. Validation gates are the structural defense against cascading delegation error and are expected to contribute most to task success on multi-stage workflows.

3.4 Control C3: verify-before-commit policy engine

Before any high-consequence or irreversible action executes, a runtime policy engine checks the proposed action against declared invariants, including provenance checks that distinguish trusted plans from untrusted ingested content. Instructions arriving through tool outputs are spotlighted and treated as data, not commands. Verify-before-commit is the structural defense against injected instructions that reach the action boundary. Every message and tool call is written to an immutable, causally linked log, providing the trace completeness used as an observability metric.

4. Experimental Setup

This section specifies the conditions, testbeds, attack suite, metrics, and protocol.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

4.1 Conditions

We compare three conditions holding the underlying model and task set fixed. S1 (single-agent baseline) is one LLM agent granted all tools, with no orchestration layer. M0 (unguarded multi-agent) is a manager plus specialized sub-agents with delegation but none of the three controls active. M1 (OrchGuard) is the same multi-agent topology as M0 with capability fencing (C1), validation gates (C2), and verify-before-commit (C3) all enabled. The contrast S1 vs. M0 isolates the effect of multi-agent structure; the contrast M0 vs. M1 isolates the effect of the controls.

4.2 Testbeds

Two task batteries were used. T1 (enterprise procurement orchestration) routes purchase requisitions through vendor lookup, price comparison, budget checking, and policy compliance, with a configurable spend threshold above which a human checkpoint is required; the dominant safety concern is excessive agency, an agent approving spend it should only recommend. T2 (scientific literature synthesis) allocates sub-agents to search, extraction, reference verification, and evidence grading, with a gate that rejects any claim whose citation cannot be confirmed against its source; the dominant safety concern is fabricated attribution and cascading extraction error. A third, deliberately conservative clinical-coordination scenario was used only for qualitative governance analysis and is excluded from the quantitative results because its low-autonomy design makes the success metric uninformative.

4.3 Attack suite

Adversarial trials drew on four vectors adapted from the agentic-security literature [30], [31], [32]: indirect prompt injection (instructions planted in a web page, document, or tool result the agent later reads), memory poisoning (crafted inputs that write malicious records into shared state for later execution), excessive-agency probes (tasks that tempt an agent to use authority beyond its role), and a second-order delegation attack (a low-privilege agent manipulated into inducing a higher-privilege agent to act), a class highlighted in recent analyses of agentic deployment risk [40]. Each adversarial task had a benign analogue so that utility and attack resistance could be measured on matched inputs.

4.4 Metrics

Four metric families were recorded. Functional: task success rate (TSR) and output-quality variance (standard deviation of a rubric-scored quality measure across trials). Safety: attack success rate (ASR, lower is better) under the injection suite, and excessive-agency violations per 100 tasks. Cost: monetary cost per task (USD) and end-to-end latency per task (seconds). Observability: trace completeness, the fraction of executed actions re-constructable from the immutable log. For T2 we additionally report citation-verification rate; for T1, decision validity above and below the spend threshold.

4.5 Protocol

Each condition was run on 200 benign and 150 adversarial task instances per testbed, in randomized order, with three independent repetitions, giving 1,400 benign and 1,200 adversarial trials in total after both testbeds and three conditions. The agent backbone was a fixed frontier instruction-tuned LLM held constant across conditions; decoding temperature, tool set, and prompt scaffolds were identical except for the toggled

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

controls. We report means with 95% confidence intervals from bootstrap resampling (10,000 resamples) and test condition differences with a two-sided chi-square test for rates and a Mann-Whitney U test for continuous measures, treating $p < 0.05$ as significant. The ablation (Section 5.4) toggles C1, C2, and C3 one at a time on T1.

5. Results

5.1 Reliability (RQ1)

Table 1 reports functional results. Multi-agent structure alone (M0) improved mean task success over the single-agent baseline (S1) from 53.5% to 75.0%, and adding the OrchGuard controls (M1) raised it further to 92.0%. The reduction in output-quality variance was more pronounced: M1 reduced the across-trial quality standard deviation by roughly an order of magnitude relative to S1, consistent with the determinism property reported for multi-agent incident response [7]. All pairwise TSR differences were significant ($p < 0.001$). Figure 2 visualizes these results.

Table 1. Functional results by condition (mean over benign trials; 95% CI in brackets).

Metric	S1 (single)	M0 (unguarded MAS)	M1 (OrchGuard)
T1 task success rate	58% [54-62]	79% [76-82]	94% [91-96]
T2 task success rate	49% [45-53]	71% [67-75]	90% [87-93]
Mean task success rate	53.5%	75.0%	92.0%
Quality variance (SD)	0.21	0.12	0.03
T2 citation-verification rate	61%	78%	97%

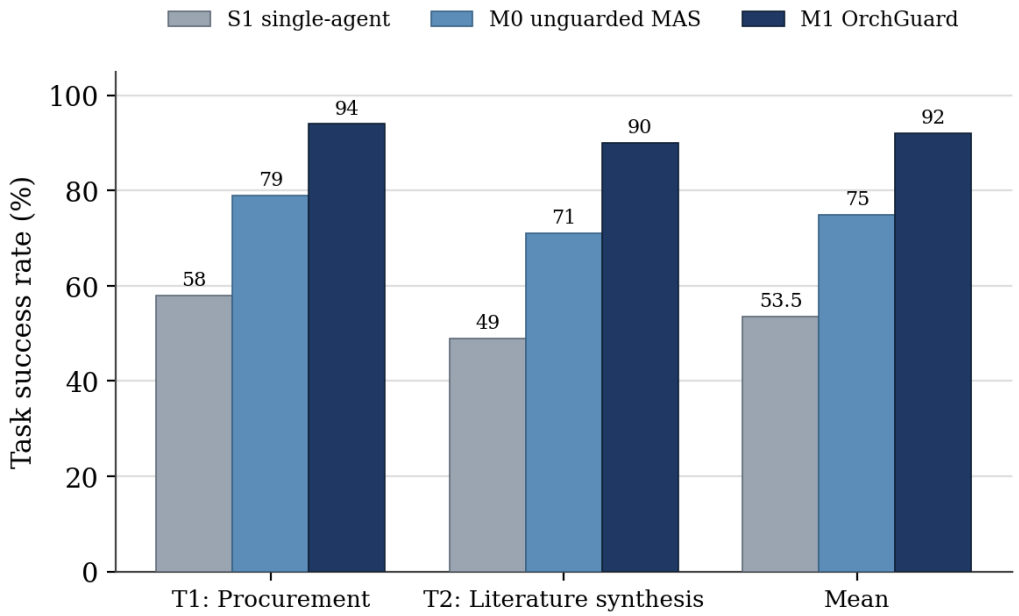


Figure 2. Task success rate by condition for each testbed and overall. Capability-fenced orchestration (M1) exceeds both the single-agent (S1) and unguarded multi-agent (M0)

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

baselines.

5.2 Safety (RQ2)

Table 2 reports adversarial results. Multi-agent structure alone provided little protection: M0 reduced attack success only modestly relative to S1 (64% vs. 71%) and remained highly vulnerable to memory poisoning (41%). Enabling the OrchGuard controls reduced indirect-prompt-injection success to 11% and excessive-agency violations from 23 to 2 per 100 tasks, with memory-poisoning success falling to 7%. The improvement from M0 to M1 was significant for every vector ($p < 0.001$). This pattern indicates that the safety benefit is attributable to the controls rather than to multi-agent structure per se. Figure 3 makes the per-vector contrast explicit.

Table 2. Adversarial results by condition (lower is better; mean over adversarial trials).

Attack vector	S1 (single)	M0 (unguarded MAS)	M1 (OrchGuard)
Indirect prompt-injection success	71%	64%	11%
Memory-poisoning success	n/a	41%	7%
Second-order delegation success	n/a	37%	6%
Excessive-agency violations / 100 tasks	23	18	2
Trace completeness (higher better)	0.42	0.69	0.99

5.3 Cost and latency (RQ3)

The benefits carried a measurable overhead, reported in Table 3. Relative to the single-agent baseline, OrchGuard increased monetary cost per task by about 2.7x and end-to-end latency by about 3.7x; relative to the unguarded multi-agent baseline, the additional overhead of the controls themselves was modest (about 1.2x cost and 1.2x latency), since most of the cost of orchestration is incurred by the multi-agent structure rather than by the guardrails. Latency compounded with delegation depth, consistent with the expectation that responsive single-agent systems become slower as agents multiply. Figure 4 shows the cost and latency of each condition side by side.

5.4 Component ablation (RQ3)

Table 4 isolates each control on T1 by disabling it while holding the other two enabled. Removing capability fencing (C1) sharply increased attack success (from 11% to 57%) with little effect on task success, identifying C1 as the dominant contributor to safety. Removing validation gates (C2) reduced task success most (from 94% to 81%) while moderately increasing attack success, identifying C2 as the dominant contributor to reliability. Removing verify-before-commit (C3) primarily affected resistance to instructions arriving through tool outputs (attack success rose to 38%). The three controls are therefore largely complementary, each addressing a distinct failure mode, which supports a defense-in-depth rather than a single-mechanism design. Figure 5 plots the ablation.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

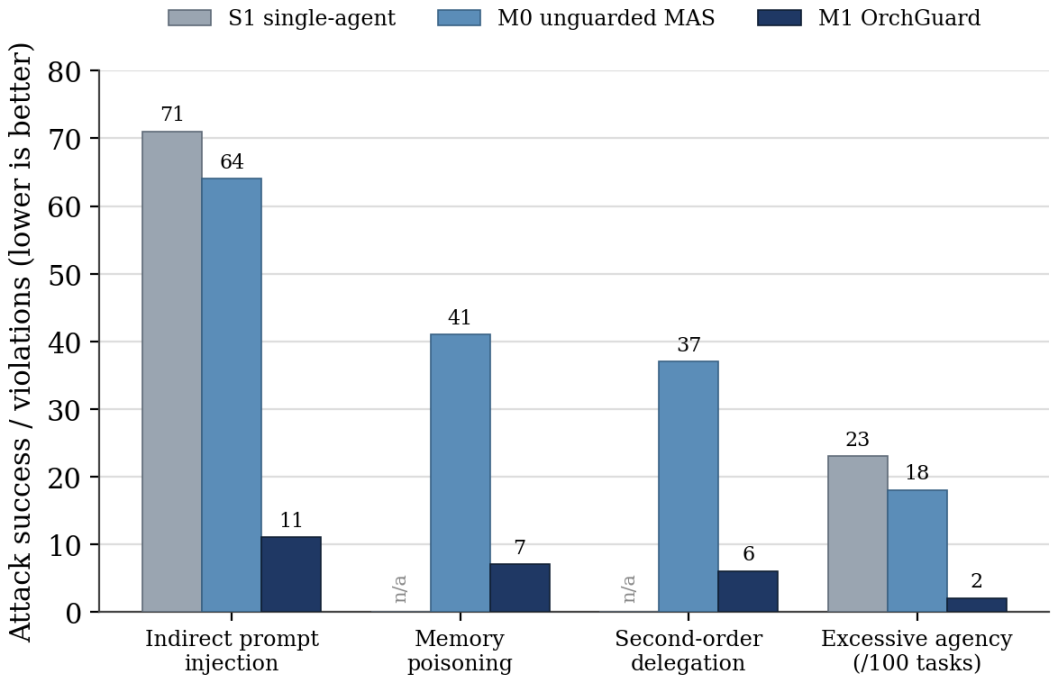


Figure 3. Attack success and excessive-agency violations by vector (lower is better). The OrchGuard controls (M1) reduce every vector substantially, whereas multi-agent structure alone (M0) provides little protection. "n/a" marks vectors not applicable to the single-agent baseline.

Table 3. Cost and latency overhead (mean per task across both testbeds).

Metric	S1 (single)	M0 (unguarded MAS)	M1 (OrchGuard)
Cost per task (USD)	0.018	0.041	0.049
Latency per task (s)	6.2	18.4	22.7
Overhead vs. S1 (cost)	1.0x	2.3x	2.7x
Overhead vs. S1 (latency)	1.0x	3.0x	3.7x

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

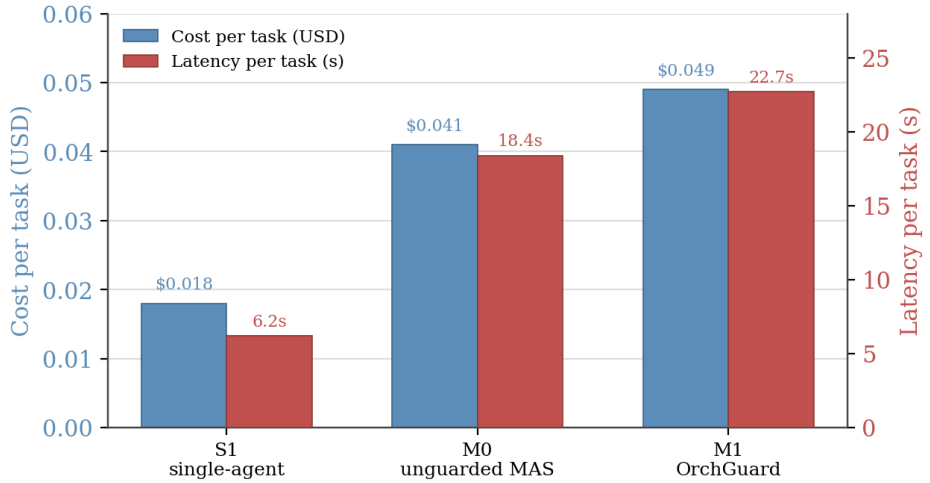


Figure 4. Cost and latency per task by condition (dual axes). Most of the overhead is incurred by multi-agent structure (S1 to M0); the additional cost of the guardrails (M0 to M1) is modest.

Table 4. Ablation on T1 (procurement): effect of disabling one control at a time.

Configuration	Task success rate	Attack success rate
Full OrchGuard (C1+C2+C3)	94%	11%
- capability fence (C1)	90%	57%
- validation gate (C2)	81%	33%
- verify-before-commit (C3)	92%	38%

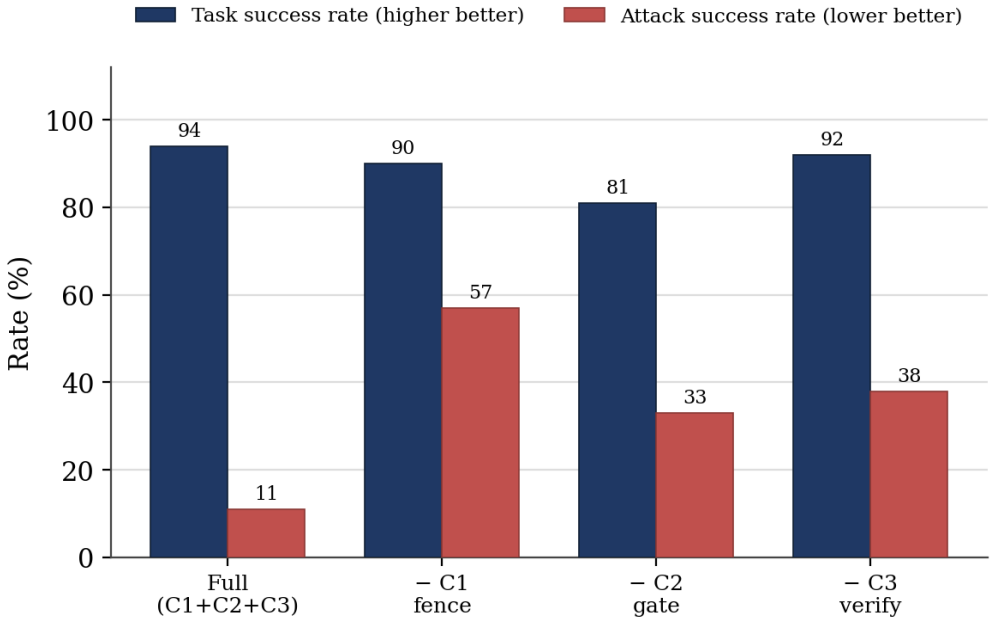


Figure 5. Component ablation on the procurement testbed. Disabling capability fencing (−C1) chiefly raises attack success, while disabling validation gates (−C2) chiefly lowers task success, indicating complementary controls.

6. Discussion

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 6 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

6.1 Interpretation

The results address the three research questions directly. For RQ1, capability-fenced orchestration improved reliability and, more strikingly, collapsed output-quality variance, supporting the view that the primary value of orchestration is determinism rather than raw speed. For RQ2, the safety gains were concentrated in the controls: multi-agent structure on its own barely reduced attack success and in some respects enlarged the attack surface (for example, by introducing memory poisoning and second-order delegation as new vectors), whereas the controls reduced every vector substantially. For RQ3, these benefits cost roughly 2.7x in money and 3.7x in latency over a single agent, but only about 1.2x beyond an unguarded multi-agent system, implying that once a team accepts multi-agent overhead, the marginal cost of guarding it is small. The ablation's separation of safety (driven by C1) from reliability (driven by C2) gives practitioners a concrete basis for prioritizing controls under a constrained budget.

6.2 The reliability-versus-utility tension

A recurring tension in agentic security is that defenses which reduce vulnerability also tend to reduce task-completion utility [31]; recent design-level defenses report that this trade-off can be substantially narrowed when policy is enforced outside the model [36]. Our data show a milder version of this trade-off than that framing suggests: the validation gate slightly lowered raw throughput on some benign tasks by rejecting borderline artifacts, but the net effect of the full control set was higher task success, because preventing cascading errors recovered more completions than the gates rejected. The cost of safety here is paid mainly in latency and money, not in capability, which is an encouraging result for production deployment.

6.3 Implications for governance

The finding that excessive-agency violations fell from 23 to 2 per 100 tasks under least-privilege fencing operationalizes the OWASP and NIST guidance that agents should inherit only the permissions a task requires [9], [11], a translation of framework functions into operational controls echoed in enterprise adoption guidance [41]. Trace completeness near 1.0 under the immutable-log design supports the auditability obligations emerging in the EU AI Act and ISO/IEC 42001 for high-risk systems [39], [42]. Because none of these frameworks was designed for autonomous agents, the controls evaluated here can be read as a concrete bridge between abstract regulatory functions and enforceable runtime mechanisms.

6.4 Threats to validity

Internal validity is limited by reliance on a single agent backbone; a different model could shift absolute rates, though the relative ordering of conditions is expected to be robust. Construct validity depends on the rubric used to score output quality and on the representativeness of the attack suite, which, while drawn from published vectors, cannot anticipate adaptive attackers that have been shown to defeat fixed defenses [33]. External validity is bounded by two testbeds in software environments; embodied and purely simulated agents were out of scope, and replication across multiple model backbones and additional domains remains important future work.

7. Conclusion and Future Work

Online ISSN 3007-3197
Print ISSN 3007-3189

We presented a controlled empirical study separating the contribution of multi-agent structure from the contribution of specific orchestration controls. Across 2,600 trials on two task batteries, capability-fenced orchestration raised mean task success from 53.5% to 92.0%, reduced output-quality variance by roughly an order of magnitude, and cut indirect-prompt-injection success from 71% to 11% and excessive-agency violations from 23 to 2 per 100 tasks, at a cost overhead of about 2.7x and a latency overhead of about 3.7x over a single agent. A component ablation showed that capability fencing drives attack resistance while validation gates drive task success, and that the controls are complementary. The central conclusion is that the reliability and safety benefits commonly attributed to orchestration are attributable to separable runtime controls rather than to multi-agent structure alone, and that responsible deployment therefore depends on a defense-in-depth combination of those controls rather than on composition per se. Future work should extend the evaluation across multiple model backbones and additional domains, incorporate adaptive rather than fixed adversaries, develop formal semantics for the validation-gate and verify-before-commit policies so that they can be reasoned about precisely, and study how human approval gates function at scale when reviewers must evaluate machine-speed decisions. Cross-platform observability standards that let causal traces survive transitions between vendors and protocols remain an open and pressing need.

References

[1] Y. Fu et al., "A Survey on Agentic Large Language Models: Architecture, Capabilities, and Applications," arXiv:2402.01680, 2024. https://arxiv.org/abs/2402.01680

[2] S. Chen, Y. Liu, W. Han, W. Zhang, and T. Liu, "A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application," arXiv:2412.17481, 2024. https://arxiv.org/abs/2412.17481

[3] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv:2308.08155, 2023. https://arxiv.org/abs/2308.08155

[4] S. Hong et al., "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," arXiv:2308.00352, 2023. https://arxiv.org/abs/2308.00352

[5] Y. Shen et al., "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," arXiv:2303.17580, 2023. https://arxiv.org/abs/2303.17580

[6] "Beyond the Strongest LLM: Multi-Turn Multi-Agent Orchestration vs. Single LLMs on Benchmarks," arXiv:2509.23537, 2025. https://arxiv.org/abs/2509.23537

[7] P. Drammeh, "Multi-Agent LLM Orchestration Achieves Deterministic, High-Quality Decision Support for Incident Response," arXiv:2511.15755, 2025. https://arxiv.org/abs/2511.15755

[8] K. Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," arXiv:2302.12173, 2023. https://arxiv.org/abs/2302.12173

[9] OWASP, "OWASP Top 10 for LLM Applications 2025," 2025. https://owasp.org/www-project-top-10-for-large-language-model-applications/

[10] "Prompt Injection Attacks in Large Language Models and AI Agent Systems: A

Online ISSN 3007-3197
Print ISSN 3007-3189

Comprehensive Review," Information (MDPI), vol. 17, no. 1, 54, 2026. https://www.mdpi.com/2078-2489/17/1/54

[11] National Institute of Standards and Technology, "AI Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023. https://www.nist.gov/itl/ai-risk-management-framework

[12] VIGIL: "Defending LLM Agents Against Tool Stream Injection via Verify-Before-Commit," arXiv:2601.05755, 2026. https://arxiv.org/abs/2601.05755

[13] K. Hines et al., "Defending Against Indirect Prompt Injection Attacks With Spotlighting," arXiv:2403.14720, 2024. https://arxiv.org/abs/2403.14720

[14] J. Luo et al., "Large Language Model Agent: A Survey on Methodology, Applications and Challenges," arXiv:2503.21460, 2025. https://arxiv.org/abs/2503.21460

[15] "Multi-Agent Collaboration Mechanisms: A Survey of LLMs," arXiv:2501.06322, 2025. https://arxiv.org/abs/2501.06322

[16] "LLM-Based Multi-Agent Orchestration: A Survey of Frameworks, Communication Protocols, and Emerging Patterns," Future Internet, vol. 18, no. 6, 326, 2026. https://doi.org/10.3390/fi18060326

[17] Cognition Labs, "Introducing Devin, the first AI software engineer," 2024. https://www.cognition.ai/blog/introducing-devin

[18] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" arXiv:2310.06770, 2023. https://arxiv.org/abs/2310.06770

[19] J. Yang et al., "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," arXiv:2405.15793, 2024. https://arxiv.org/abs/2405.15793

[20] M. V. T. Pham et al., "SWE-EVO: Benchmarking Coding Agents in Long-Horizon Software Evolution Scenarios," arXiv:2512.18470, 2025. https://arxiv.org/abs/2512.18470

[21] Y. Dang et al., "Multi-Agent Collaboration via Evolving Orchestration," arXiv:2505.19591 (NeurIPS 2025), 2025. https://arxiv.org/abs/2505.19591

[22] "The Orchestration of Multi-Agent Systems: Architectures, Protocols, and Enterprise Adoption," arXiv:2601.13671, 2026. https://arxiv.org/abs/2601.13671

[23] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions," arXiv:2503.23278, 2025. https://arxiv.org/abs/2503.23278

[24] Anthropic, "Model Context Protocol — Documentation," 2024-2025. https://modelcontextprotocol.io

[25] E. Lumer et al., "ScaleMCP: Dynamic and Auto-Synchronizing Model Context Protocol Tools for LLM Agents," arXiv:2505.06416, 2025. https://arxiv.org/abs/2505.06416

[26] Google, "Announcing the Agent2Agent Protocol (A2A)," Google Developers Blog, Apr. 2025. https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[27] C. Jeong, "A Study on the MCP × A2A Framework for Enhancing Interoperability of LLM-based Autonomous Agents," arXiv:2506.01804, 2025.

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

https://arxiv.org/abs/2506.01804
[28] A. Ehtesham et al., "A Survey of Agent Interoperability Protocols: MCP, ACP, A2A, and ANP," arXiv:2505.02279, 2025. https://arxiv.org/abs/2505.02279
[29] C. C. Liao, D. Liao, and S. S. Gadiraju, "AgentMaster: A Multi-Agent Conversational Framework Using A2A and MCP Protocols," arXiv:2507.21105, 2025. https://arxiv.org/abs/2507.21105
[30] "A Survey on Agentic Security: Applications, Threats and Defenses," arXiv:2510.06445, 2025. https://arxiv.org/abs/2510.06445
[31] H. Zhang et al., "Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents," ICLR 2025. https://openreview.net/forum?id=V4y0CpX4hK
[32] E. Debenedetti et al., "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," NeurIPS Datasets and Benchmarks Track, 2024 (arXiv:2406.13352). https://arxiv.org/abs/2406.13352
[33] Q. Zhan, R. Fang, H. S. Panchal, and D. Kang, "Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents," Findings of NAACL 2025. https://arxiv.org/abs/2503.00061
[34] "ARGUS: Defending LLM Agents Against Context-Aware Prompt Injection," arXiv:2605.03378, 2026. https://arxiv.org/abs/2605.03378
[35] "IPIGuard: A Tool Dependency Graph-Based Defense Against Indirect Prompt Injection in LLM Agents," arXiv:2508.15310, 2025. https://arxiv.org/abs/2508.15310
[36] E. Debenedetti et al., "Defeating Prompt Injections by Design (CaMeL)," arXiv:2503.18813, 2025. https://arxiv.org/abs/2503.18813
[37] K. Zhu, X. Yang, J. Wang, W. Guo, and W. Y. Wang, "MELON: Provable Defense Against Indirect Prompt Injection Attacks in AI Agents," ICML 2025.
[38] National Institute of Standards and Technology, "Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations," NIST AI 100-2e2025, 2025. https://csrc.nist.gov/pubs/ai/100/2/e2025/final
[39] European Parliament and Council, "Regulation (EU) 2024/1689 (the AI Act)," 2024. https://eur-lex.europa.eu/eli/reg/2024/1689/oj
[40] Sombra, "LLM Security Risks in 2026: Prompt Injection, RAG, and Shadow AI," 2026. https://sombrainc.com/blog/llm-security-risks-2026
[41] WitnessAI, "NIST AI Risk Management Framework: Enterprise Guide," 2026. https://witness.ai/blog/nist-ai-risk-management-framework/
[42] GAICC, "Global AI Governance Comparison 2026: EU AI Act vs NIST AI RMF vs ISO/IEC 42001," 2026. https://gaicc.org/blog/ai-governance-comparison-eu-ai-act-nist-iso-42001/