Research Article

Evaluating Agentic AI Architectures for Enterprise Applications

David Paxton[1] · Ruth Okonkwo[2] · Simon Marsh[3]

[1] Department of Computer Science & Electrical Engineering, University of Missouri–Kansas City, Kansas City, MO, United States
[2] School of Computing and Information Technology, University of Wollongong, Wollongong, NSW, Australia
[3] Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, Malaysia
Corresponding author: David Paxton — editor@jipcet.org

Journal of Innovation, Product, Computing & Emerging Technologies (JIPCET) · ISSN 2998-4475 · Volume 3, Issue 2 (2026), pp. 1–40
Received 3 August 2026 · Revised 12 October 2026 · Accepted 28 October 2026 · Published 14 November 2026
DOI: 10.5281/jipcet.2026.0042 · Licence: CC BY 4.0

Abstract

Enterprise deployments of large language model (LLM) agents have expanded faster than the evidence base that would let an engineering organisation choose between orchestration designs on anything more than intuition. This paper reports a controlled, multi-site evaluation of three widely used agentic architectures — unconstrained sequential chaining, supervisor–worker delegation, and constrained tool-calling with a typed action space — across twelve enterprise workflows drawn from software maintenance, financial operations, customer support, procurement and internal knowledge retrieval. We executed 4,800 tasks under identical model, temperature and retrieval configurations, scoring each with two independent human raters and adjudicating disagreements. Constrained tool-calling achieved a task completion rate of 0.81 (95% CI 0.79–0.83) against 0.74 (0.72–0.76) for supervisor–worker delegation and 0.62 (0.60–0.65) for unconstrained chaining, a relative improvement of 31% over the chaining baseline. The reliability gain did not carry a token premium: median cost per *completed* task fell from US$0.214 to US$0.147, because constraint suppressed the long recovery trajectories that dominate the cost distribution's upper tail. Latency behaved differently: constrained designs were slower at the median (11.4 s versus 9.2 s) but far tighter at the 95th percentile (28.1 s versus 96.7 s). We further show that 63% of unconstrained failures are attributable to four recurring error modes — argument hallucination, premature termination, loop non-termination and silent tool-error swallowing — and that three of the four are structurally preventable at the orchestration layer rather than at the prompt. We conclude with an architecture decision framework, an instrumentation checklist, and the observation that reliability in agentic systems is largely purchased through the deliberate removal of degrees of freedom.

Keywords: AI agents · large language models · enterprise AI · orchestration architecture · evaluation methodology · reliability engineering · tool use

How to cite: Paxton, D., Okonkwo, R., & Marsh, S. (2026). Evaluating agentic AI architectures for enterprise applications. Journal of Innovation, Product, Computing & Emerging Technologies, 3(2), 1–40. https://doi.org/10.5281/jipcet.2026.0042
Data and code availability: The task specifications, rater rubric, adjudication log and analysis notebooks accompany this article as supplementary material. Raw transcripts are released in redacted form; three participating organisations withheld consent for verbatim release and are represented by paraphrased traces.

Highlights

- Across 4,800 human-scored tasks in twelve enterprise workflows, orchestration architecture accounted for a 19-percentage-point absolute difference in task completion (0.62 to 0.81).
- Constrained tool-calling with a typed action space outperformed supervisor–worker delegation and unconstrained chaining in eleven of twelve workflows.
- Reliability was not purchased with tokens: cost per completed task fell 31%, because constraint eliminates the long recovery trajectories that dominate agentic spend.
- 95th-percentile latency improved 3.4-fold, while median latency worsened slightly — the trade-off that matters depends on whether a human is waiting.
- Four failure modes account for 63% of failures; three are preventable at the orchestration layer rather than in the prompt.
- Delegation manufactures a new failure surface: 21% of supervisor–worker failures were coordination losses at report boundaries.
- The effect shrinks in short, read-only workflows and reverses in genuinely open-ended ones, which we report as a boundary condition rather than a caveat.

Statement of significance

Organisations selecting an agent orchestration layer today do so largely without comparative engineering evidence, and the decision is usually framed as a choice of framework or vendor. This study reframes it as a choice about how many degrees of freedom to grant the system, and shows that the answer has consequences comparable in magnitude to a model generation change — available immediately, at no licensing cost, to any team willing to restructure its agent loop.

The result that we expect to change practice is the cost finding. Prevailing intuition treats reliability and spend as a trade-off, and teams accordingly evaluate orchestration layers on cost per model call. Our data show that the dominant contribution to spend in an unreliable system is failed work, so the architecture that fails less is also the architecture that costs less per unit of delivered value. Teams measuring cost per call are using the wrong denominator, and the substitution is arithmetic rather than difficult.

For the research community, we offer a protocol that holds model configuration fixed while varying only structure, an execution-level dataset with a published data dictionary, and a reviewer checklist (Appendix M) against which we report our own two failures. Agent evaluation is at the stage where methodological convention matters more than any individual result, and we would rather contribute to that convention than defend a number.

Declaration on the use of AI. Generative tools were used for reference formatting and grammatical revision. No text in the Results or Discussion sections was generated without author verification against the underlying measurements. The authors accept full responsibility for the content of this article.

Contents

1. Introduction
2. Background and related work
3. Architectures under evaluation
4. Workflow corpus
5. Experimental method
6. Scoring and inter-rater reliability
7. Reliability results
8. Cost results
9. Latency and tail behaviour
10. Failure taxonomy
11. Sensitivity analyses
12. Case studies
13. An architecture decision framework
14. Instrumentation checklist
15. Threats to validity
16. Discussion
17. Limitations and future work
18. Conclusion
Acknowledgements
References
Appendix A. Task specifications
Appendix B. Rater rubric
Appendix C. Full result tables
Appendix D. Per-workflow discussion
Appendix E. Statistical detail
Appendix F. Glossary
Appendix G. Reproduction guide
Appendix H. Redacted transcript excerpts
Appendix I. Peer review and editorial history
Appendix J. Data dictionary for the released dataset
Appendix K. Ethics, governance and data protection
Appendix L. Notes on terminology and framing
Appendix M. A checklist for reviewers of agent evaluation studies

1. Introduction

Within the space of roughly three years, the composition of enterprise software teams changed in a way that few technology transitions manage. Systems that plan, call tools and revise their own intermediate work moved from research demonstrations to production dependencies, and they did so without the accompanying body of comparative engineering evidence that normally precedes such a move. Practitioners are asked to choose an orchestration layer — a decision with multi-year architectural consequences — largely on the basis of vendor benchmarks, informal community reports and the persuasive power of demonstrations. This paper is an attempt to supply part of the missing evidence.

Our central question is narrow by design. Holding the model, the retrieval corpus, the temperature and the task set fixed, how much does the *shape* of the orchestration affect the reliability, cost and latency of the resulting system? We deliberately avoid the more fashionable question of which model is best. Model rankings are unstable on a scale of months; architecture decisions outlive them. If the architectural contribution to reliability is small, organisations should invest their attention elsewhere. If it is large, then the widespread practice of treating orchestration as an implementation detail is a mistake with a measurable price.

We find the contribution to be large. Across twelve workflows and 4,800 scored tasks, moving from unconstrained sequential chaining to constrained tool-calling with a typed action space raised the task completion rate from 0.62 to 0.81. The magnitude of that difference is comparable to the gap commonly observed between successive model generations, and it is available immediately, at no licensing cost, to any team willing to restructure its agent loop.

A second finding is, in our view, the more useful one. The reliability improvement is not paid for in tokens. Naive reasoning suggests that constraint should cost more: a typed action space requires schema validation, rejection and retry, all of which consume additional model calls. In aggregate, however, cost per completed task *fell* by 31%, because the dominant cost in an unconstrained system is not the successful path but the long, wandering recovery trajectory that follows an unforced error. Constraint prevents the class of error that generates those trajectories. The upper tail of the cost distribution collapses, and the mean follows it down.

Section 2 situates this work in the existing literature on tool use, planning and agent evaluation. Section 3 specifies the three architectures precisely enough to be reimplemented. Sections 4 through 6 describe the workflow corpus, the experimental procedure and the human scoring protocol, including inter-rater reliability. Sections 7 to 9 report reliability, cost and latency. Section 10 develops a failure taxonomy that we believe generalises beyond our corpus. Sections 11 and 12 present sensitivity analyses and four case studies. Sections 13 and 14 translate the findings into an architecture decision framework and an instrumentation checklist intended to be used directly by practitioners. Sections 15 to 18 address validity, discuss implications, acknowledge

limitations and conclude.

1.1 Contributions

We claim four contributions. First, a reproducible multi-site evaluation protocol for agentic architectures that holds model configuration fixed and varies only orchestration structure. Second, quantitative reliability, cost and latency results with confidence intervals across twelve enterprise workflows. Third, an empirically grounded failure taxonomy with an assessment of which failure modes are preventable at the orchestration layer as opposed to the prompt layer. Fourth, a decision framework that maps workflow properties to architecture recommendations, together with the instrumentation required to verify those recommendations locally.

1.2 What this paper does not claim

We do not claim that constrained tool-calling is universally preferable. Section 13 identifies workflow classes — chiefly open-ended research and exploratory synthesis — in which the freedom of an unconstrained design remains valuable and in which our own results favour it. Nor do we claim that our completion rates transfer to other corpora; the *ordering* of the architectures is the transferable result, not the absolute numbers.

2. Background and related work

Work on tool-augmented language models divides usefully into three strands: the mechanics of tool invocation, the structure of multi-step planning, and the evaluation of the resulting systems. Our contribution sits in the third strand but depends on the first two.

2.1 Tool invocation

Early tool-use work established that models can learn to emit structured calls to external functions and to incorporate the returned values into subsequent reasoning. The engineering question that followed was how strictly those calls should be validated. Permissive designs accept free-form call syntax and parse it downstream; strict designs constrain generation against a schema, rejecting non-conforming output before execution. The literature reports reliability benefits from strictness, but generally in single-call settings where the compounding behaviour we study cannot appear.

2.2 Planning and decomposition

A parallel strand examines how a system decomposes a goal into steps. Approaches range from implicit decomposition inside a single generation, through explicit plan-then-execute separation, to delegation architectures in which a supervisor allocates subgoals to specialised workers. Reported comparisons are typically conducted on academic benchmarks with short horizons and unambiguous success criteria, which understates the coordination cost that appears in enterprise workflows with eight to thirty steps and contested definitions of success.

2.3 Evaluation

Agent evaluation remains methodologically unsettled. Automated scoring is attractive at scale but tends to reward well-formed output over correct output, and is especially unreliable where success depends on side effects in an external system. Human scoring is expensive and introduces rater variance that many published results do not quantify. We adopt human scoring with two independent raters and report Cohen's κ, both because our success criteria are partly judgemental and because we wish the rater variance to be visible rather than assumed away.

Three gaps motivate this study. Existing comparisons rarely hold the model fixed while varying architecture; they rarely use workflows with realistic step counts and side effects; and they rarely report cost per *completed* task, which is the quantity an operator actually budgets against. We address all three.

3. Architectures under evaluation

The three architectures are specified below at a level of detail sufficient for reimplementation. All three share the same model, the same retrieval index, the same tool implementations and the same step budget of thirty actions.

3.1 A1 — Unconstrained sequential chaining

The model receives the goal, the tool documentation as prose, and the accumulated transcript. At each turn it emits free-form text that may contain a tool invocation in a loosely specified format. A parser extracts candidate invocations with permissive regular expressions, executes those it can interpret, and appends results to the transcript. Malformed invocations are appended as text without an error signal — a choice that reflects common practice in production systems built on early frameworks. Termination occurs when the model emits a final answer or the step budget is exhausted.

3.2 A2 — Supervisor–worker delegation

A supervisor model decomposes the goal into subgoals and dispatches each to a worker with a restricted tool subset and its own step budget of eight actions. Workers return structured reports; the supervisor may accept a report, request revision once, or re-decompose. The supervisor never invokes tools directly. This design localises failure — a failed worker does not corrupt the whole transcript — at the cost of an additional coordination layer and the information loss inherent in report summarisation.

3.3 A3 — Constrained tool-calling with a typed action space

Every tool is declared with a JSON schema specifying required and optional arguments, types, enumerated values and inter-argument constraints. Generation is constrained so that non-conforming calls cannot be emitted; where the serving stack does not support constrained decoding, calls are validated and rejected with a structured error naming the violated constraint, and the rejection does not consume a step. Tool errors are surfaced as typed results that the model must handle explicitly. A monotonic progress predicate detects repetition of an already-attempted state and forces either a different action or termination.

Property	A1 Chaining	A2 Supervisor–worker	A3 Constrained
Action space	Free text	Free text per worker	Typed, schema-validated
Invalid call handling	Silently appended	Worker-local retry	Typed rejection, no step cost
Tool error visibility	Often swallowed	Summarised in report	Explicit typed result
Loop protection	Step budget only	Per-worker budget	Progress predicate
Coordination overhead	None	Supervisor turns	Schema validation
Step budget	30	30 total / 8 per worker	30

Table 1. Structural differences between the three evaluated architectures. Model, retrieval configuration and tool implementations are identical across conditions.

4. Workflow corpus

Twelve workflows were contributed by five organisations under a data-use agreement permitting publication of aggregate results and redacted traces. Each workflow is a task that the contributing organisation already performs, with an existing definition of acceptable output and an existing tool surface. We rejected candidate workflows that could be completed in fewer than five steps, on the grounds that they do not exercise the compounding behaviour of interest.

ID	Workflow	Domain	Median steps	Side effects	n
W1	Dependency upgrade with test repair	Software maintenance	17	Yes (writes)	400
W2	Flaky test triage and quarantine	Software maintenance	11	Yes (writes)	400
W3	Invoice exception resolution	Financial operations	9	Yes (ledger)	400
W4	Month-end reconciliation narrative	Financial operations	14	No	400
W5	Tier-2 support escalation drafting	Customer support	8	No	400
W6	Refund eligibility adjudication	Customer support	7	Yes (ledger)	400
W7	Supplier compliance document check	Procurement	12	No	400
W8	Purchase order anomaly investigation	Procurement	13	No	400
W9	Internal policy question answering	Knowledge retrieval	6	No	400
W10	Cross-system record reconciliation	Knowledge retrieval	19	Yes (writes)	400
W11	Incident timeline reconstruction	Operations	22	No	400
W12	Access review recommendation	Security operations	10	Yes (tickets)	400

Table 2. The twelve workflows. Step counts are medians observed under A3. Each workflow contributes 400 task instances, distributed evenly across the three architectures with sampling described in Section 5.

Workflows with side effects were executed against staging systems seeded from redacted production snapshots, with a transactional wrapper permitting rollback between task instances. This design choice is important: several failure modes documented in Section 10 are invisible in read-only evaluation because they concern the ordering and idempotence of writes rather than the content of a final answer.

5. Experimental method

5.1 Configuration control

All conditions used the same model checkpoint, temperature 0.2, top-p 0.95, an identical retrieval index rebuilt once before the campaign, and identical tool implementations behind an adapter that normalised interface differences. Prompt text differed only where architecture required it: A3's tool documentation is schema-derived and therefore not textually identical to A1's prose documentation. We regard this as an unavoidable confound and quantify it in Section 11.2 by re-running A1 with schema-derived prose.

5.2 Sampling and randomisation

For each workflow we drew 400 task instances from the contributing organisation's backlog, stratified by a difficulty proxy (number of distinct systems touched) to avoid concentrating hard instances in any condition. Instances were assigned to architectures by block randomisation within stratum. Execution order was interleaved across conditions to distribute any drift in external system behaviour evenly.

5.3 Budgets and termination

Every condition received a thirty-action budget and a wall-clock ceiling of five minutes per task. Budget exhaustion is recorded as a distinct outcome from an incorrect answer, because the two have different operational remedies. A2's per-worker budget of eight actions was chosen in pilot runs as the smallest value that did not systematically truncate legitimate subgoals.

5.4 Measurement

For each task we recorded the full transcript, every tool invocation with arguments and result, token counts by direction, wall-clock latency decomposed into model time, tool time and orchestration overhead, and the terminal outcome. Cost was computed from token counts at published list prices and is reported both per attempted and per completed task.

6. Scoring and inter-rater reliability

Two raters with domain experience in the relevant workflow scored each task independently against a written rubric (Appendix B) on three axes: task completion, side-effect correctness and output usability. Raters saw the transcript and the resulting system state but not the architecture label; because transcripts differ structurally between conditions, we acknowledge that blinding was imperfect and treat this as a threat to validity (Section 15.3).

Cohen's κ across all scored tasks was 0.79 for task completion, 0.84 for side-effect correctness and 0.61 for output usability. The lower agreement on usability is expected: it is the most judgemental axis, and we therefore avoid drawing strong conclusions from it. Disagreements were resolved by a third adjudicator whose decisions are published in the adjudication log.

Axis	Definition	Cohen's κ	Disagreement rate
Task completion	Goal achieved without human repair	0.79	9.4%
Side-effect correctness	External state left valid and idempotent	0.84	6.1%
Output usability	Result usable without rewriting	0.61	18.7%

Table 3. Scoring axes and inter-rater agreement. Task completion is the primary outcome throughout the paper; usability is reported for completeness only.

7. Reliability results

Table 4 reports task completion rates by architecture and workflow. The ordering A3 > A2 > A1 holds in eleven of twelve workflows. The single exception is W11, incident timeline reconstruction, where A1 marginally outperforms A2 and is statistically indistinguishable from A3; we discuss this instructive case in Section 12.4.

Workflow	A1 chaining	A2 supervisor	A3 constrained	A3 − A1
W1	0.54	0.69	0.78	+0.24
W2	0.61	0.72	0.83	+0.22
W3	0.66	0.77	0.88	+0.22
W4	0.63	0.71	0.79	+0.16
W5	0.71	0.78	0.84	+0.13
W6	0.69	0.80	0.91	+0.22
W7	0.58	0.73	0.82	+0.24
W8	0.55	0.70	0.80	+0.25
W9	0.74	0.79	0.86	+0.12
W10	0.49	0.68	0.77	+0.28
W11	0.62	0.60	0.65	+0.03
W12	0.63	0.75	0.85	+0.22
All	0.62	0.74	0.81	+0.19

Table 4. Task completion rate by workflow and architecture (n = 400 per workflow). Pooled 95% confidence intervals: A1 0.60–0.65, A2 0.72–0.76, A3 0.79–0.83.

The pooled difference between A3 and A1 is 0.19 in absolute terms and 31% in relative terms. A paired analysis at the workflow level gives a mean difference of 0.194 with a standard error of 0.019; the null hypothesis of no architectural effect is rejected decisively. More interesting than the significance is the consistency: the effect appears in every domain, across step counts from six to twenty-two, and with and without side effects.

7.1 Where the gain concentrates

Decomposing the A3 − A1 difference by workflow property shows the gain concentrating in workflows with many steps and with write side effects. Workflows above the median step count gain 0.23 on average; those below gain 0.15. Workflows with write side

effects gain 0.24; read-only workflows gain 0.14. Both patterns are consistent with a compounding account of failure: constraint prevents individual missteps, and the value of preventing a misstep grows with the number of subsequent steps that would inherit its consequences.

7.2 Budget exhaustion as a distinct outcome

A1 exhausted its action budget on 14.2% of tasks; A2 on 7.8%; A3 on 2.1%. Budget exhaustion is the signature of loop non-termination, and its near-elimination under A3 is attributable specifically to the progress predicate rather than to the typed action space. We separate these two mechanisms in the ablation reported in Section 11.3.

8. Cost results

Cost per attempted task rises modestly with constraint, as expected from schema validation and rejection overhead. Cost per *completed* task moves in the opposite direction, and by a larger margin. Table 5 gives the decomposition.

Metric	A1	A2	A3
Median tokens per attempted task	8,410	11,920	7,980
Median cost per attempted task (US$)	0.133	0.181	0.119
Median cost per completed task (US$)	0.214	0.196	0.147
90th percentile cost per task (US$)	0.982	0.611	0.243
Share of spend on failed tasks	38%	26%	19%

Table 5. Cost decomposition. A2 consumes the most tokens per attempt because supervisor turns and worker report summarisation are additive; A3 is cheapest per completed task because its failure distribution has a far shorter tail.

The 90th-percentile figures carry the explanation. Under A1, a single unforced error — most often a hallucinated argument accepted silently by the permissive parser — initiates a recovery trajectory in which the model repeatedly reinterprets an inconsistent transcript. These trajectories consume the step budget and produce nothing. They are rare in count and dominant in cost. Under A3 the initiating error is structurally impossible, so the trajectory never begins.

For operators this reframes the budgeting question. The relevant quantity is not tokens per call but the fraction of spend attributable to tasks that will be redone by a human. That fraction is 38% under A1 and 19% under A3.

9. Latency and tail behaviour

Constraint costs median latency and buys tail latency. A3's median of 11.4 s exceeds A1's 9.2 s because validation, rejection and re-generation occur on the successful path. At the 95th percentile the relationship inverts sharply: 28.1 s for A3 against 96.7 s for A1, with A2 intermediate at 61.3 s.

Percentile	A1 (s)	A2 (s)	A3 (s)
p50	9.2	14.8	11.4

Percentile	A1 (s)	A2 (s)	A3 (s)
p75	18.6	26.4	16.9
p90	48.3	44.1	22.7
p95	96.7	61.3	28.1
p99	241.0	138.6	44.2

Table 6. End-to-end latency distribution by architecture, all workflows pooled.

Which end of this distribution matters depends on the deployment. An interactive assistant with a human waiting is governed by the median; a queued batch process with a service-level objective is governed by the tail. Teams frequently optimise the metric they can see most easily in a demonstration, which is the median, and are then surprised by the operational behaviour of the tail. We recommend that latency requirements be stated as a percentile before an architecture is selected.

10. Failure taxonomy

We coded 1,842 failed tasks into a taxonomy developed inductively on a 200-task development sample and then applied to the remainder. Four modes account for 63% of all failures and for a substantially larger share of wasted spend.

10.1 M1 — Argument hallucination

The model invokes a real tool with a plausible but non-existent identifier: an account number, a file path, a ticket reference. Under a permissive parser the call executes, returns an error or an empty result, and the error is appended as ordinary text. The model then reasons over a transcript containing a false premise. This mode accounts for 24% of A1 failures and 3% of A3 failures; the residual under A3 consists of identifiers that are schema-valid but semantically wrong, which no type system can catch.

10.2 M2 — Premature termination

The system emits a confident final answer while a required step remains outstanding — typically a write that was planned but never executed. This mode is dangerous precisely because the output looks complete; it is the failure most likely to reach a user unchecked. It accounts for 17% of A1 failures. A3 reduces but does not eliminate it (6%), because a typed action space constrains how actions are expressed, not whether the model believes it has finished.

10.3 M3 — Loop non-termination

The system revisits an already-attempted state, often with cosmetic variation, until the budget is exhausted. This is the principal driver of the cost tail described in Section 8. The progress predicate in A3 addresses it directly, reducing its incidence from 13% of failures to under 1%.

10.4 M4 — Silent tool-error swallowing

A tool returns an error that the orchestration layer discards, converts to an empty string, or embeds in prose that the model does not distinguish from data. Subsequent

reasoning proceeds as if the call had succeeded. At 9% of A1 failures this mode is the least frequent of the four and the easiest to fix: surfacing tool errors as typed results is a change of a few dozen lines in most codebases.

Mode	Description	A1	A2	A3	Preventable at orchestration layer?
M1	Argument hallucination	24%	12%	3%	Largely yes (schema validation)
M2	Premature termination	17%	14%	6%	Partly (completion predicate)
M3	Loop non-termination	13%	8%	<1%	Yes (progress predicate)
M4	Silent tool-error swallowing	9%	5%	<1%	Yes (typed results)
M5	Retrieval miss	11%	10%	11%	No (retrieval layer)
M6	Genuine reasoning error	14%	15%	16%	No (model capability)
M7	Coordination loss	—	21%	—	Architecture-specific
M8	Other / unclassified	12%	15%	13%	—

Table 7. Failure mode distribution as a share of that architecture's failures. Percentages are within-architecture and therefore not comparable in absolute count; A3 has 47% fewer failures in total.

Two observations deserve emphasis. First, M5 and M6 — retrieval misses and genuine reasoning errors — are essentially constant across architectures. This is the expected result and a useful sanity check: orchestration cannot fix a missing document or an incorrect inference. Second, M7, coordination loss, is *created* by A2. Twenty-one per cent of supervisor–worker failures arise because a worker's report omitted a detail the supervisor needed. Delegation localises failure and simultaneously manufactures a new failure surface at every report boundary.

11. Sensitivity analyses

11.1 Model substitution

We repeated the full campaign on W1, W6 and W10 with two additional model checkpoints of differing capability. Absolute completion rates shifted by up to 0.11, but the architecture ordering was preserved in all three cases, and the A3 − A1 gap narrowed only modestly with the strongest model (from 0.25 to 0.19 on W1). Stronger models make fewer unforced errors but do not stop making them, and the compounding mechanism is unchanged.

11.2 Prompt-parity control

To address the documentation confound noted in Section 5.1, we re-ran A1 on all twelve workflows with tool documentation generated from the same schemas used by A3, rendered as prose. Completion rose from 0.62 to 0.65. Better documentation therefore explains roughly three points of the nineteen-point gap; the remaining sixteen are attributable to enforcement rather than description. Telling a model the shape of a valid call is not equivalent to making an invalid call impossible.

11.3 Ablating A3's mechanisms

A3 combines three mechanisms. Removing each in turn from the full configuration gives: without schema constraint, 0.71; without the progress predicate, 0.75; without typed tool errors, 0.78. The mechanisms are complementary and not redundant, and schema constraint is the single largest contributor. A team able to make only one change should make that one.

Configuration	Completion	Δ from full A3
Full A3	0.81	—
− schema constraint	0.71	−0.10
− progress predicate	0.75	−0.06
− typed tool errors	0.78	−0.03
− all three (≈ A1)	0.63	−0.18

Table 8. Ablation of A3's three mechanisms, pooled across workflows.

11.4 Step-budget sensitivity

Doubling the budget from thirty to sixty actions raised A1's completion from 0.62 to 0.66 and A3's from 0.81 to 0.82, while roughly doubling A1's cost on failed tasks. Additional budget mostly extends unproductive trajectories. Budget is not a substitute for structure.

12. Case studies

12.1 W10 — Cross-system record reconciliation

W10 shows the largest architectural gap (+0.28) and illustrates the compounding mechanism cleanly. The task requires reading records from three systems, identifying discrepancies and writing corrections. Under A1, a hallucinated record identifier in an early read produced an empty result that the model interpreted as an absent record, leading it to create a duplicate rather than correct an existing one. The write succeeded; the outcome was wrong and left the system in a state requiring manual repair. Under A3 the identifier violated its schema pattern, the call was rejected with a typed error, and the model re-derived the identifier from the source system.

12.2 W6 — Refund eligibility adjudication

W6 has the highest absolute completion under A3 (0.91) despite touching a ledger, because its action space is small and highly typed: eligibility decisions draw on an enumerated set of reason codes. Where the domain already has a controlled vocabulary, encoding it in the action space is close to free and captures most of the available benefit.

12.3 W1 — Dependency upgrade with test repair

W1 exercises the longest sequences of writes. Its A2 result (0.69) is notably better than A1 (0.54) and reveals a genuine strength of delegation: a worker that fails to repair one test does not corrupt the transcript for the others. Delegation is a form of blast-radius control, and where independent subgoals genuinely exist it is valuable. Its weakness is visible in the coordination-loss figures of Table 7.

12.4 W11 — Incident timeline reconstruction, the exception

W11 is the only workflow where A1 is competitive. The task is open-ended: assemble a narrative from logs, chat transcripts and deployment records where the relevant evidence is not known in advance. A typed action space over a fixed tool set constrains exploration in ways that hurt here, and A2's report boundaries lose exactly the incidental detail that later proves significant. We take W11 seriously as a boundary condition: where the shape of a good solution cannot be specified beforehand, constraint has less to offer and may cost something.

13. An architecture decision framework

The results support a small number of actionable rules. We state them as defaults with explicit exceptions rather than as universal recommendations.

Workflow property	Recommended default	Rationale
Write side effects present	A3 constrained	Largest measured gain; prevents invalid-state writes
Median steps > 12	A3 constrained	Compounding failure dominates at long horizons
Controlled vocabulary exists	A3 constrained	Typing is nearly free and captures most benefit
Independent parallel subgoals	A2 supervisor	Blast-radius control outweighs report loss
Tail latency in the SLO	A3 constrained	p95 improves 3.4× over chaining
Open-ended exploration	A1 or hybrid	Constraint restricts useful search (see W11)
Read-only, ≤ 6 steps	Any; prefer simplest	Architectural effect is smallest here

Table 9. Decision defaults mapping workflow properties to architecture, derived from Sections 7–12.

A hybrid deserves mention. Several participating teams have since adopted an arrangement in which an unconstrained exploratory phase produces a plan, which is then executed through a typed action space. Our data do not evaluate this design, and we flag it as the most promising direction for follow-up work rather than as a recommendation.

14. Instrumentation checklist

The results above are only actionable for a team that can measure its own system. The following instrumentation was necessary to produce every number in this paper and is, in our experience, absent from most production deployments.

Terminal outcome taxonomy. Record success, incorrect answer, budget exhaustion and harness error as distinct outcomes. Collapsing them hides the loop problem entirely.

Per-call argument logging. Log tool arguments and results verbatim. Argument hallucination is undetectable from token counts and latency alone.

Cost per completed task. Compute spend divided by successful outcomes, not by attempts. This is the only cost figure that corresponds to delivered value.

Latency percentiles. Report p50, p90, p95 and p99. A median-only dashboard is structurally unable to show the failure mode that most affects operators.

Repeated-state detection. Hash the attempted-action set per task and count repetitions. This is the cheapest available proxy for M3.

Side-effect validity checks. Assert external-state invariants after every task. Premature termination frequently produces output that reads as correct.

Rater agreement on any human scoring. Publish κ. Unreported rater variance makes small differences uninterpretable.

15. Threats to validity

15.1 Construct validity

Task completion as scored by human raters is a proxy for operational value. A task may be completed and still be useless, or fail on a technicality while producing a helpful artefact. We report usability as a secondary axis to expose this gap, while noting its lower rater agreement.

15.2 Internal validity

Prompt text could not be held perfectly constant across architectures; Section 11.2 quantifies the residual at roughly three completion points. External systems drifted over the eleven-week campaign; interleaved execution distributes that drift across conditions but does not remove it.

15.3 Blinding

Raters were not told the architecture but could often infer it from transcript structure. We consider the risk of bias highest on the usability axis, which is the axis we rely on least, and lowest on side-effect correctness, which is verified against system state rather than judged.

15.4 External validity

Five organisations, all with mature engineering practices and staging environments, are not a representative sample of enterprises. Absolute completion rates should not be transferred to other settings. The architecture ordering, which is consistent across twelve workflows, three model checkpoints and six domains, is the result we expect to generalise.

15.5 Researcher allegiance and reporting bias

Two of the three authors had implemented constrained architectures professionally before this study began and expected A3 to win. Researcher allegiance is a documented source of inflated effect sizes in comparative work, and it is not neutralised by good intentions. Three features of our design limit but do not remove it: the primary outcome was defined and the rubric written before any execution; scoring was performed by raters who were not authors and who had no stake in the architectural question; and

side-effect correctness, our highest-agreement axis, is verified against machine-checkable system state rather than judged. Against this, the failure taxonomy was developed and applied by the authors, with only a partial independent re-coding, and Reviewer 2 was right to press on it (Appendix I.2).

A reader who wishes to discount our interpretation while retaining our measurements can do so. The completion rates, token counts, latencies and terminal outcomes in the released dataset are machine-recorded and independent of any authorial judgement. Only the failure-mode column and the usability axis depend materially on human coding, and both are separable in the data. We have tried to structure the release so that our conclusions are auditable rather than merely stated.

16. Discussion

The most economical summary of our findings is that reliability in agentic systems is bought by removing degrees of freedom. This is an unfashionable conclusion. The appeal of agentic architectures rests substantially on their generality, and constraint reads as a retreat from that generality. Our data suggest the retreat is worth making wherever the shape of an acceptable solution is known in advance — which, in enterprise settings, is most of the time.

There is a useful analogy with the history of type systems in software engineering. The argument for static typing was never that programmers cannot write correct dynamic code; it was that at scale, with many contributors and long-lived systems, the cost of making a class of error impossible is lower than the cost of detecting instances of it. Our M1 and M4 results are that argument transposed to a new substrate. An argument that cannot be malformed does not need to be validated downstream, and an error that cannot be swallowed does not need to be traced.

The cost result deserves particular attention from anyone building a business case. The prevailing mental model treats reliability and cost as a trade-off: better output is assumed to require more tokens. Our data contradict this at the architecture level. The dominant contribution to spend in an unconstrained system is failed work, and failed work is the thing constraint eliminates. Teams evaluating orchestration layers on cost per call are measuring the wrong denominator.

Finally, the A2 results complicate a common assumption. Delegation is frequently presented as the natural response to unreliability — divide the problem, contain the damage. It does contain damage, as W1 shows. But it also introduces a new failure surface at every report boundary, and in our corpus coordination loss accounts for more than a fifth of A2's failures. Adding a layer of indirection to an unreliable system produces a differently unreliable system unless the boundaries are themselves typed.

16.1 Implications for procurement

Organisations increasingly buy an orchestration layer rather than build one, and procurement questionnaires in this area tend to ask about model support, connector

breadth and observability dashboards. Our results suggest three questions that matter more. Does the platform enforce a typed action space, or merely document one? Are tool errors surfaced to the model as distinguishable results, or flattened into text? Is there a mechanism that detects repetition of an already-attempted state, and can its threshold be inspected? A vendor unable to answer these is selling a chaining architecture with better documentation, and our prompt-parity control (Section 11.2) indicates what better documentation is worth: about three points of nineteen.

A fourth question concerns measurement rather than mechanism. Ask whether the platform can report cost per completed task, where completion is defined by the buyer rather than by the platform. Most cannot, because they do not know which executions succeeded. A platform that cannot distinguish success from termination cannot help an operator manage spend, whatever its dashboard shows.

16.2 Implications for team structure

The participants in this study allocated engineering attention in a pattern that our results suggest is inverted. Prompt revision is visible, iterative and satisfying; harness work is structural, unglamorous and difficult to demonstrate in a meeting. The organisations that moved fastest during the campaign were those that assigned one engineer explicit ownership of the orchestration layer, with the reliability metrics of Section 14 as their stated remit. Where the harness was collectively owned it tended to accrete permissive behaviour, because every individual permissive choice resolved an immediate obstacle.

We note this cautiously. It is an observation about five organisations, gathered informally alongside a study designed to measure something else, and it is exactly the kind of claim about organisational behaviour that a reviewer rightly asked us to remove elsewhere in this paper. We record it as a hypothesis worth designing a study around, not as a finding.

16.3 Why the effect has been easy to miss

If architecture accounts for a nineteen-point difference, it is reasonable to ask why this is not already common knowledge. Three features of prevailing practice conspire to hide it. Public benchmarks favour short, read-only tasks, where our own data show the smallest gap (0.12 on W9). Vendor comparisons vary model and architecture together, so any architectural contribution is absorbed into a model claim. And production dashboards report cost per call and median latency, the two metrics on which constrained designs look marginally worse. A team could operate an unconstrained system for a year, watch every metric it collects, and never see the nineteen points it is leaving unclaimed.

17. Limitations and future work

Our corpus contains no workflow requiring genuine multi-agent negotiation, no workflow with a human in the loop mid-execution, and no workflow spanning more than

five minutes of wall-clock time. Each of these is common in practice and may behave differently. We also did not evaluate the hybrid explore-then-constrain design that several participants have since adopted, and we regard that as the single most valuable extension.

Three further directions follow from our failure taxonomy. M2, premature termination, is the mode least improved by our interventions and the most dangerous operationally; a learned or specified completion predicate is an obvious target. M5, retrieval miss, is architecture-invariant here and should be attacked at the retrieval layer with the same measurement discipline we have applied to orchestration. And the interaction between constraint and model capability — the observation that stronger models narrow but do not close the gap — deserves a study designed for it rather than a sensitivity analysis.

18. Conclusion

Across 4,800 human-scored tasks in twelve enterprise workflows, the architecture of an agentic system accounted for a nineteen-point absolute difference in task completion, a 31% reduction in cost per completed task, and a 3.4-fold improvement in 95th-percentile latency. The effect is consistent across domains, step counts, side-effect profiles and model checkpoints. Three specific mechanisms — schema-constrained action spaces, typed tool errors and a progress predicate — account for most of it, and all three are implementable without changing models or vendors.

Orchestration is not an implementation detail. It is the layer at which the largest available reliability gains currently sit, and it is measurable today by any team willing to instrument its terminal outcomes and divide its spend by its successes.

17.1 A note on the pace of the field

Any empirical claim about language-model systems risks obsolescence before publication, and we have tried to structure this study so that its useful lifetime exceeds that of its numbers. Our sensitivity analysis across three model checkpoints (Section 11.1) is the evidence we can offer: absolute completion moved by up to eleven points while the ordering of the architectures held. The mechanism we propose — compounding of unforced error — predicts that this should continue to hold as long as models make any unforced errors at all, and predicts the gap should narrow gradually rather than close abruptly. A replication in two years that found the gap at ten points rather than nineteen would confirm rather than refute our account; one that found it unchanged would suggest that capability gains are being spent on harder tasks rather than on fewer slips.

We would encourage authors of future comparisons to report the ordering of conditions as their headline result and the absolute rates as supporting detail. The literature currently does the reverse, which is one reason its findings age so badly.

Acknowledgements

We thank the five participating organisations, whose engineering teams absorbed considerable disruption to permit controlled execution against staging systems, and the eleven raters who scored transcripts over eleven weeks. We thank the reviewers for the prompt-parity control in Section 11.2, which materially changed our interpretation of the results. Any remaining errors are the authors'.

Author contributions. D.P. designed the study and the scoring protocol and wrote the manuscript. R.O. built the evaluation harness and conducted the statistical analysis. S.M. developed the failure taxonomy and led the case studies. All authors reviewed and approved the final version.

Competing interests. The authors declare no competing financial interests. D.P. serves as Editor-in-Chief of this journal and took no part in the editorial handling, reviewer selection or decision for this submission, which was managed independently by a handling editor.

References

[1] Abadi, N., & Steele, P. (2025). Compounding error in multi-step language model pipelines. *Transactions on Applied Machine Intelligence*, 7(2), 118–141.

[2] Bhatt, S., Oyelaran, T., & Ferris, K. (2025). Schema-constrained decoding for reliable tool invocation. *Proceedings of the Conference on Applied Language Systems*, 441–456.

[3] Cardoso, M., & Whitfield, A. (2024). Delegation architectures for autonomous software agents. *Journal of Software Architecture Practice*, 12(4), 289–312.

[4] Delacroix, H. (2026). Cost models for agentic workloads. *JIPCET*, 3(1), 44–68.

[5] Eriksen, L., Park, J., & Aziz, M. (2025). Human scoring protocols for agent evaluation. *Empirical Software Engineering Reports*, 30(6), 1102–1130.

[6] Fanshawe, R. (2024). Tail latency as a service-level concern in generative systems. *Operations Engineering Quarterly*, 19(3), 55–72.

[7] Grieves, T., & Nakamura, S. (2025). Progress predicates and loop prevention in agent loops. *Proceedings of the Symposium on Reliable Autonomy*, 77–92.

[8] Hale, E., & Marsh, S. (2024). Tool-calling constraints in production assistants. *Proceedings of APSEC*, 202–215.

[9] Ibarra, C. (2026). Idempotence requirements for agent-initiated writes. *Data Systems Practice*, 8(1), 33–51.

[10] Jelinek, P., & Okonkwo, R. (2025). Reliability bounds for tool-augmented language models. *Journal of Emerging Computation*, 18(1), 44–61.

[11] Kowalski, D., & Fitzgerald, M. (2024). Retrieval failure as a confound in agent benchmarks. *Information Retrieval Practice*, 22(5), 611–629.

[12] Lindqvist, A. (2025). Instrumenting non-deterministic systems. *Observability Engineering*, 4(2), 88–107.

[13] Moretti, G., & Sundaram, V. (2026). Blast-radius control in multi-agent orchestration. *Transactions on Distributed Intelligence*, 3(2), 205–228.

[14] Nwosu, B. (2024). Silent error propagation in language model pipelines. *Software Quality Letters*, 15(4), 300–317.

[15] Ó Braonáin, S., & Tan, W. (2025). Enumerated action spaces for adjudication tasks. *Proceedings of the Workshop on Applied Decision Systems*, 12–27.

[16] Pfeiffer, R. (2026). Prompt parity in architectural comparisons. *Methods in Machine Intelligence*, 2(1), 9–24.

[17] Rahman, A. (2026). Evaluation cost models for assistive systems. *JIPCET*, 1(1), 12–29.

[18] Sørensen, K., & Villanueva, I. (2025). Staging-environment fidelity in agent evaluation. *Empirical Systems Research*, 11(3), 145–166.

[19] Thorne, J. (2024). Budget exhaustion as a diagnostic signal. *Applied Autonomy Notes*, 6(2), 41–52.

[20] Ueda, H., & Castellanos, P. (2026). Typed error surfaces for external tool calls. *Proceedings of the Conference on Software Interfaces*, 512–530.

[21] Vasquez, L. (2025). Inter-rater reliability in qualitative agent assessment. *Research Methods in Computing*, 9(4), 233–255.

[22] Wren, O., & Adeyemi, F. (2026). Hybrid exploration and constrained execution. *Preprint series in applied intelligence*, AI-2026-114.

Appendix A. Task specifications

Each workflow specification states the goal, the available tools, the acceptance criteria and the external-state invariants asserted after execution. Specifications are reproduced here in abbreviated form; full machine-readable versions accompany the supplementary material.

W1 — Dependency upgrade with test repair

Goal. Raise a named dependency to a target version and restore a green test suite.
Tools. repo read/write, build runner, test runner, changelog search
Acceptance. Suite passes; no unrelated files modified; lockfile consistent.

W2 — Flaky test triage and quarantine

Goal. Identify non-deterministic tests from run history and quarantine with justification.
Tools. test history query, repo write, issue tracker
Acceptance. Only genuinely flaky tests quarantined; each carries a linked issue.

W3 — Invoice exception resolution

Goal. Resolve a held invoice by reconciling purchase order, receipt and invoice lines.
Tools. ledger read/write, PO search, document store
Acceptance. Ledger balanced; single posting; audit note attached.

W4 — Month-end reconciliation narrative

Goal. Produce a variance narrative explaining movements above threshold.
Tools. ledger read, prior-period read, note store
Acceptance. Every flagged variance addressed with a cited figure.

W5 — Tier-2 support escalation drafting

Goal. Draft an escalation containing reproduction steps and prior remediation.
Tools. ticket read, knowledge search, telemetry query
Acceptance. All mandatory escalation fields present and evidenced.

W6 — Refund eligibility adjudication

Goal. Decide eligibility against policy and post the outcome with a reason code.
Tools. policy search, order read, ledger write
Acceptance. Reason code from the enumerated set; single ledger entry.

W7 — Supplier compliance document check

Goal. Verify that a supplier's submitted documents satisfy current requirements.
Tools. document store, requirements registry, supplier record
Acceptance. Each requirement marked satisfied or not with a document reference.

W8 — Purchase order anomaly investigation

Goal. Explain an anomalous purchase order against historical patterns.
Tools. PO history query, supplier record, price index
Acceptance. Explanation cites at least two independent comparators.

W9 — Internal policy question answering

Goal. Answer a staff policy question with citations to the governing document.
Tools. policy corpus search, version history
Acceptance. Answer correct against current version; citation resolves.

W10 — Cross-system record reconciliation

Goal. Reconcile a customer record across CRM, billing and support systems.
Tools. three system read/write adapters, match index
Acceptance. No duplicates created; all three systems consistent.

W11 — Incident timeline reconstruction

Goal. Assemble a chronological incident narrative from heterogeneous sources.
Tools. log search, chat archive, deploy history
Acceptance. Timeline complete to the minute; each entry sourced.

W12 — Access review recommendation

Goal. Recommend retain or revoke for each entitlement in a review batch.
Tools. identity directory, usage telemetry, ticket write
Acceptance. Every entitlement decided; revocations ticketed.

Appendix B. Rater rubric

Raters scored each task on three binary axes with a written decision procedure. The procedure below was applied without modification throughout the campaign; a change log confirming this accompanies the supplementary material.

B.1 Task completion

Score 1 if the stated goal was achieved such that a competent practitioner would accept the result without further work. Score 0 if any required step is missing, any produced artefact requires correction, or the system terminated on budget exhaustion. Partial credit is not available; borderline cases are referred to adjudication rather than split.

B.2 Side-effect correctness

Score 1 if all external-state invariants hold after execution and no unintended write occurred. Duplicate creation, non-idempotent repetition and orphaned records each score 0 regardless of whether the final answer was correct.

B.3 Output usability

Score 1 if the produced text or artefact could be forwarded to its intended recipient without rewriting. Formatting defects, unexplained jargon and unsupported assertions score 0. Raters were instructed that this axis is independent of correctness.

B.4 Adjudication

Where the two raters disagreed, a third rater with no prior exposure to the task decided the score after reading the transcript, the system state and both raters' written justifications. Adjudicated tasks are flagged in the released data; excluding them entirely changes the pooled A3 − A1 difference by 0.006.

Appendix C. Full result tables

Workflow	A1 p50 (s)	A1 p95 (s)	A2 p50	A2 p95	A3 p50	A3 p95
W1	14.1	188.4	22.6	104.2	18.3	41.7
W2	10.2	96.1	16.1	62.8	12.9	27.4
W3	7.8	61.3	12.4	48.9	9.6	20.1
W4	11.6	104.7	17.8	70.4	13.8	30.2
W5	6.9	48.2	11.2	40.1	8.4	17.6
W6	6.1	42.9	10.4	36.8	7.9	15.8
W7	9.4	82.6	15.2	58.3	11.7	24.9
W8	10.8	98.4	16.6	64.7	13.2	28.6
W9	5.2	31.8	9.1	28.4	6.8	13.2
W10	15.7	214.6	24.9	121.8	20.1	48.3
W11	18.4	236.1	28.3	142.6	23.7	56.9
W12	8.6	72.4	14.1	54.2	10.9	22.8

Table C1. Latency percentiles by workflow and architecture.

Workflow	A1 cost/completed	A2 cost/completed	A3 cost/completed	Δ %
W1	0.318	0.264	0.201	−37%
W2	0.221	0.198	0.152	−31%
W3	0.164	0.152	0.114	−30%
W4	0.242	0.214	0.168	−31%
W5	0.138	0.131	0.099	−28%
W6	0.126	0.118	0.086	−32%
W7	0.204	0.186	0.141	−31%
W8	0.236	0.208	0.159	−33%
W9	0.102	0.098	0.076	−25%
W10	0.361	0.288	0.224	−38%
W11	0.394	0.412	0.318	−19%
W12	0.186	0.168	0.128	−31%

Table C2. Cost per completed task in US dollars, by workflow and architecture.

Appendix D. Per-workflow discussion

Sections 12.1 to 12.4 discussed four workflows in depth. For completeness, and because practitioners tend to look for the workflow closest to their own, we record short notes on each of the twelve. Each note states what made the workflow difficult, which failure mode dominated, and what we would recommend to a team implementing it.

W1 Dependency upgrade with test repair

What made it hard. Long write sequences with interdependent files, and a success criterion (green suite) that is expensive to evaluate at every step.

Dominant failure mode. M2 premature termination under A1: the system reported the upgrade complete while two tests remained red, having convinced itself they were pre-existing failures.

Recommendation. Assert the invariant — suite green — as a machine check before allowing termination. This single change removed most remaining A3 failures in our pilot.

W2 Flaky test triage and quarantine

What made it hard. The judgement is statistical, and the evidence lives in run history rather than in code.

Dominant failure mode. M5 retrieval miss: history queries with too narrow a window classified genuinely flaky tests as merely failing.

Recommendation. Fix the query window in the tool rather than leaving it to the model. Architecture does little here; tool design does most of the work.

W3 Invoice exception resolution

What made it hard. Three documents must agree, and the write is financially consequential.

Dominant failure mode. M1 argument hallucination under A1, with invented purchase-order numbers producing empty reads treated as absent records.

Recommendation. Type the identifier formats. Purchase orders and invoices almost always have checkable patterns, which makes this workflow unusually amenable to constraint.

W4 Month-end reconciliation narrative

What made it hard. The output is prose, so acceptance is partly judgemental and rater agreement is lower.

Dominant failure mode. M6 genuine reasoning error: explanations that were internally coherent but attributed a variance to the wrong driver.

Recommendation. Require every claim to cite a retrieved figure. This does not improve the reasoning but makes an error visible to a reviewer in seconds.

W5 Tier-2 support escalation drafting

What made it hard. Short and read-only, therefore the workflow where architecture matters least.

Dominant failure mode. M2 premature termination, usually an omitted mandatory field.

Recommendation. A template completeness check is sufficient. Do not over-engineer the orchestration for workflows of this shape.

W6 Refund eligibility adjudication

What made it hard. Financially consequential but with a small, enumerated decision space.

Dominant failure mode. M6 reasoning error on genuinely borderline policy cases.

Recommendation. Encode the reason codes as an enumeration and route low-confidence cases to a human. This workflow achieved our highest completion rate, 0.91.

W7 Supplier compliance document check

What made it hard. Requirements change over time and documents are heterogeneous scans and PDFs.

Dominant failure mode. M5 retrieval miss against superseded requirement versions.

Recommendation. Version the requirements registry and make the version an explicit tool argument.

W8 Purchase order anomaly investigation

What made it hard. Requires comparison against historical distributions rather than a lookup.

Dominant failure mode. M3 loop non-termination under A1, repeatedly re-querying overlapping history windows.

Recommendation. A progress predicate over the attempted-query set eliminated this almost entirely.

W9 Internal policy question answering

What made it hard. The simplest workflow in the corpus; six median steps and no writes.

Dominant failure mode. M5 retrieval miss, essentially the only mode that matters here.

Recommendation. Invest in retrieval, not orchestration. The A3 − A1 gap of 0.12 is the smallest we measured and is not worth a rearchitecture on its own.

W10 Cross-system record reconciliation

What made it hard. Three systems, bidirectional writes, and a duplicate-creation hazard.

Dominant failure mode. M1 argument hallucination cascading into non-idempotent writes.

Recommendation. Type the identifiers and make every write idempotent by construction. This workflow produced our largest gain, +0.28, and our most damaging observed failures.

W11 Incident timeline reconstruction

What made it hard. Genuinely open-ended: the relevant evidence is not known before the search begins.

Dominant failure mode. M8 unclassified, reflecting the difficulty of coding failures on a task whose acceptable solutions are diverse.

Recommendation. Do not constrain the exploratory phase. This is the one workflow where we would consider chaining, or a hybrid, over a typed action space.

W12 Access review recommendation

What made it hard. High volume, uniform structure, consequential revocations.

Dominant failure mode. M2 premature termination, batches reported complete with entitlements undecided.

Recommendation. Enumerate the batch and require a decision per element before termination. Structure does almost all the work in workflows of this shape.

Appendix E. Statistical detail

This appendix records the analytical choices behind the intervals reported in the body of the paper. We have kept the analysis deliberately simple, on the view that an architectural effect of nineteen percentage points does not require sophisticated machinery to establish, and that simple methods are easier for a reader to check.

E.1 Intervals on completion rates

Completion is a binary outcome per task. Pooled intervals are Wilson score intervals at the 95% level, computed on n = 4,800 per architecture. Wilson was preferred to the normal approximation because several per-workflow rates approach 0.9, where the normal interval misbehaves. Per-workflow intervals, computed on n = 400, have half-widths between 0.030 and 0.049 and are reported in the supplementary tables rather than in the body.

E.2 Paired workflow-level comparison

Because workflows differ enormously in intrinsic difficulty, the primary comparison is paired at the workflow level: for each workflow we take the difference in completion rate between two architectures and analyse the twelve differences. The A3 − A1 differences have mean 0.194, standard deviation 0.065 and standard error 0.019. A two-sided paired t-test gives t = 10.4 on 11 degrees of freedom. A distribution-free alternative, the Wilcoxon signed-rank test, rejects at the same level with all twelve differences positive.

E.3 Clustering

Tasks within a workflow are not independent: they share a tool surface, a retrieval corpus and an organisational context. Treating 4,800 tasks as independent would understate the intervals. Our workflow-level pairing is a deliberately conservative response, effectively reducing the analytical sample size to twelve. A cluster-robust logistic model with workflow-level clustering gives an odds ratio for A3 versus A1 of 2.62 with a 95% interval of 2.11 to 3.25, consistent with the paired result.

E.4 Cost and latency

Cost and latency distributions are strongly right-skewed, so we report medians and percentiles rather than means throughout, and we bootstrap percentile intervals with 10,000 resamples stratified by workflow. The reported 3.4-fold p95 latency improvement has a bootstrap interval of 2.9 to 4.1. We deliberately avoid reporting mean latency, which in this setting is a statement about the tail disguised as a statement about typical behaviour.

E.5 Multiplicity

We report three pairwise architecture comparisons on the primary outcome and apply a Holm–Bonferroni correction across them; all three remain significant at the 0.05 level. The subgroup analyses in Section 7.1 and the sensitivity analyses in Section 11 are explicitly exploratory, were not pre-specified, and should be read as descriptive.

E.6 Deviations from the analysis plan

Two deviations are recorded. First, the prompt-parity control in Section 11.2 was added after peer review and was not in the original plan. Second, we had planned to report mean cost per task and switched to medians and percentiles once the skew of the distributions became apparent during piloting; both figures are available in the supplementary material so that a reader can verify that the choice does not favour our conclusion.

Appendix F. Glossary

Terminology in this field is used inconsistently across vendors and papers. The definitions below are the ones used throughout this article.

Action space. The set of operations an agent may perform, together with the specification of how each is expressed. A typed action space specifies argument types and permitted values; a free-text action space does not.

Agentic system. A system in which a language model selects and sequences its own actions over multiple turns in pursuit of a goal, rather than producing a single response to a single prompt.

Attempted task. One execution of one task instance under one architecture, regardless of outcome. Cost per attempted task is the denominator most dashboards use.

Budget exhaustion. Termination because the permitted number of actions was consumed without a final answer. Recorded separately from an incorrect answer because the operational remedy differs.

Completed task. An attempted task scored 1 on the task-completion axis by the rating procedure in Appendix B. Cost per completed task is the denominator we recommend.

Compounding failure. The mechanism by which a single early error corrupts the transcript and thereby degrades every subsequent decision, so that failure probability grows super-linearly in step count.

Constrained decoding. Generation restricted at sampling time so that output cannot violate a supplied schema. Where the serving stack does not support it, post-hoc validation with structured rejection is a close substitute.

Coordination loss. Failure arising because information needed downstream was omitted from a summarised report at a delegation boundary. Specific to multi-agent designs.

Idempotence. The property that repeating an operation leaves external state unchanged after the first application. Essential where an agent may retry a write.

Orchestration layer. The code that decides what the model is asked next, what tools are exposed, how their results are represented, and when execution stops. The subject of this paper.

Premature termination. Emission of a final answer while a required step remains outstanding. The most operationally dangerous failure mode in our taxonomy because the output appears complete.

Progress predicate. A check that each action advances the task state, used to detect and interrupt repetition of already-attempted states.

Side effect. A change to external system state produced by a tool call, as distinct from information returned to the model. Workflows with side effects require rollback infrastructure to evaluate safely.

Step budget. The maximum number of actions permitted per task. A cap on cost, not a reliability mechanism; see Section 11.4.

Supervisor–worker delegation. An architecture in which one model decomposes a goal and dispatches subgoals to models with restricted tool access, receiving structured reports.

Tail latency. Latency at a high percentile, typically p95 or p99. The governing quantity for queued and batch deployments, as distinct from the median.

Typed tool error. A tool failure surfaced to the model as a structured, distinguishable result rather than as prose or an empty value.

Task instance. One concrete unit of work drawn from an organisational backlog, executed once per architecture so that comparisons are paired.

Terminal outcome. The reason execution stopped: completion, incorrect answer, budget exhaustion or harness error. Recording these separately is the single cheapest instrumentation improvement available to most teams.

Tool surface. The set of external operations exposed to a system for a given workflow, together with their preconditions. Distinct from the action space, which specifies how those operations may be expressed.

Transcript. The accumulated record of goal, model output and tool results that constitutes the system's working context. Corruption of the transcript is the mechanism by which a single early error degrades later decisions.

Unclassified failure. A failure that the taxonomy of Section 10 does not accommodate, reported as M8. We report it explicitly because a low unclassified share is evidence that a taxonomy is complete, and a hidden one is evidence of nothing.

Unforced error. A failure that the model's own capability did not require it to make, and which the orchestration layer could have prevented. M1, M3 and M4 are largely unforced; M5 and M6 are not.

Appendix G. Reproduction guide

We have attempted to make this study reproducible at three levels of effort, on the view that most readers will want the cheapest of the three.

G.1 Verifying our analysis

The scored outcomes, token counts and latencies for all 14,400 executions are released as a single tabular file, together with the notebooks that produce every table and interval in this article. Verification requires no model access and runs in under a minute.

G.2 Re-running the evaluation on our corpus

The harness, tool adapters and task specifications are released, but three of the twelve workflows depend on proprietary staging systems that cannot be distributed. Nine workflows are re-runnable against the supplied fixtures. A full re-run consumed approximately 118 million tokens in our campaign; readers should budget accordingly.

G.3 Applying the protocol to your own workflows

This is the use we most want to enable, and the instrumentation checklist in Section 14 is written for it. The minimum viable local replication is: choose two workflows above and below your median step count, implement the three A3 mechanisms behind a feature flag, run two hundred task instances per condition, and score with two raters against the Appendix B rubric. In our experience this is two to three weeks of one engineer's time and produces an estimate precise enough to justify or refuse a rearchitecture.

G.4 A note on negative results

We would consider it a useful outcome if a replication in another organisation found no architectural effect. Section 15.4 is candid that our five participants are not representative, and the mechanism we propose — compounding of unforced error — predicts that the effect should shrink towards zero in short, read-only, low-cardinality workflows. Our W9 result, a gap of 0.12, is already the beginning of that pattern. We encourage replications to report step count and side-effect profile prominently, so that a null result can be interpreted rather than merely recorded.

Appendix H. Redacted transcript excerpts

Four excerpts are reproduced to make the failure modes of Section 10 concrete. Identifiers, organisation names and customer details are replaced by placeholders in square brackets. Excerpts are abridged at the points marked with an ellipsis; unabridged versions are in the supplementary material for the nine workflows whose owners consented to verbatim release.

H.1 M1 argument hallucination (W3, architecture A1)

Model. I will retrieve the purchase order associated with this invoice. Calling po_lookup with number [PO-48127].
Tool. (returns) {}
Model. The purchase order does not exist, so this invoice has no matching commitment. I will post it as an unmatched expense to the miscellaneous account. …

The invoice did have a purchase order; its number was [PO-48217], which appears in the invoice document the model had already read. The permissive parser executed the invented number, the empty result was appended without an error signal, and the model drew a confident and wrong inference from it. Under A3 the same generated number satisfied the format pattern but failed the existence precondition declared on the tool, which returned a typed `NOT_FOUND` result; the model then re-read the invoice and corrected the digit transposition.

H.2 M2 premature termination (W1, architecture A1)

Model. The dependency has been raised to [4.2.0] and I have updated three call sites. Running the test suite.
Tool. (returns) 412 passed, 2 failed
Model. Two failures remain, both in [module], which appear unrelated to this upgrade and are likely pre-existing. The upgrade is complete.
Model. (final answer) Dependency upgraded successfully; test suite green apart from two unrelated pre-existing failures. …

Both failures were caused by the upgrade. The transcript shows the model asserting a diagnosis it did not investigate, and then restating it in the final answer in stronger terms than the intermediate reasoning supported. This pattern — an unverified hypothesis hardening into a reported fact across two turns — recurred in a substantial minority of M2 cases and is, in our view, the single most important target for future work. Note that no type system prevents it: every call in this transcript is well-formed.

H.3 M3 loop non-termination (W8, architecture A1)

Model. Querying purchase history for supplier [S-2214], window 180 days.
Tool. (returns) 47 records
Model. This may not be sufficient context. Querying purchase history for supplier [S-2214], window 200 days.
Tool. (returns) 51 records
Model. Let me widen slightly. Querying purchase history for supplier [S-2214], window 240 days. … (nineteen further queries, windows between 180 and 400 days) … **Harness.**

Step budget exhausted.

The information required to answer was present in the first result. Twenty-two of the thirty available actions were spent on near-identical queries, and the task produced no output at all. This trajectory cost 14.2 times the median successful task on the same workflow, which illustrates why cost tails and not medians govern agentic spend. A progress predicate over the attempted-query set interrupts this after the second repetition.

H.4 M7 coordination loss (W1, architecture A2)

Worker 2 report. Repaired failing assertion in [test_file]; suite now passes locally for the affected module.
Supervisor. Accepting report. All subgoals complete; assembling final answer.
Model. (final answer) Upgrade complete with three call sites updated and one test repaired. …

Worker 2 had also modified a shared fixture in order to make its assertion pass, a change that broke two tests owned by another worker's subgoal. The report did not mention the fixture, the supervisor could not have known to ask, and the final answer was accordingly wrong. This is the failure surface that delegation creates: correctness at every boundary depends on a summary being complete with respect to concerns the summariser was not told about. Typing the report — requiring an explicit enumeration of every file touched — removed this class of failure in a subsequent pilot, at the cost of longer reports.

Appendix I. Peer review and editorial history

In keeping with the journal's transparency policy, the editorial timeline and the substance of the reviewer reports are published alongside the article. Reviewers chose to remain anonymous; their reports are reproduced with their consent and lightly edited only to remove identifying detail.

Date	Event
3 August 2026	Submission received; automated completeness and scope screening passed
6 August 2026	Assigned to handling editor (Editor-in-Chief recused as an author)
11 August 2026	Three reviewers invited; two accepted within 48 hours
14 August 2026	Third reviewer accepted following a second invitation round
19 September 2026	All three reports returned; decision of major revision issued
12 October 2026	Revised manuscript and point-by-point response received
24 October 2026	Reviewers 1 and 3 confirmed satisfaction; Reviewer 2 declined further review
28 October 2026	Accepted by handling editor
14 November 2026	Published in Volume 3, Issue 2

Table I1. Editorial timeline. Time from submission to first decision was 47 days, within the journal's 56-day target.

I.1 Reviewer 1

Recommendation: major revision. The central comparison is well posed and the multi-site design is a genuine strength; I am not aware of another study that holds model configuration fixed across this many realistic workflows. My substantive concern is a confound. The A3 condition receives schema-derived tool documentation, which is plausibly better documentation than A1's prose, and the authors cannot currently distinguish the effect of enforcement from the effect of description. Without a control I do not think the headline claim can stand as stated. I would also ask the authors to report clustering explicitly; 4,800 tasks across twelve workflows is not 4,800 independent observations, and the intervals as presented are too narrow.

Author response. We accept both points. We have added a prompt-parity control (Section 11.2) in which A1 is re-run with schema-derived documentation rendered as prose; it accounts for approximately three of the nineteen percentage points, and we now state this explicitly in the abstract's framing and in the discussion. On clustering, we have made the paired workflow-level analysis primary, reducing our effective sample to twelve, and added a cluster-robust model as a secondary check (Appendix E.3).

I.2 Reviewer 2

Recommendation: major revision. I am sceptical of the failure taxonomy. Taxonomies developed inductively on a development sample and then applied by the same team tend to reproduce the authors' prior beliefs, and here the taxonomy conveniently divides into modes the proposed architecture fixes and modes it does not. I would want to see the coding done by someone with no stake in the result, and I would want the unclassified share reported prominently rather than in a table footnote. A smaller point: the latency discussion asserts that teams optimise the median because demonstrations show the

median. That may well be true, but it is an assertion about organisational behaviour and the paper offers no evidence for it.

Author response. The concern about taxonomy is fair and we have addressed it partially rather than fully. Two of the eleven raters, neither of whom was involved in developing the taxonomy or in the architectural implementation, re-coded a random 20% sample; agreement with the original coding was κ = 0.71, which we now report. We have not been able to arrange fully independent coding of the whole set and acknowledge this in Section 15. The unclassified share now appears as an explicit row (M8) in Table 7. On latency, we have softened the claim to a recommendation about stating requirements as percentiles and removed the unsupported inference about why teams do otherwise.

I.3 Reviewer 3

Recommendation: minor revision. This is a useful paper and I expect practitioners to read it, which raises the bar on the decision framework in Section 13. As drafted it reads as though constrained tool-calling is almost always correct, and the W11 exception is confined to a single subsection late in the paper. Given that a substantial fraction of real deployments are exploratory in exactly the way W11 is, I would ask the authors to give the boundary condition equal prominence. I would also encourage them to say plainly that the absolute numbers will not transfer; readers habitually quote completion rates out of context.

Author response. We have promoted the boundary condition into Section 1.2, into the framework table as an explicit row, and into the conclusion's framing. We have added a sentence in Section 1.2 and in Section 15.4 stating that the ordering rather than the absolute rates is the transferable result.

I.4 Handling editor's note on acceptance

The revision addresses the principal methodological objection with a control that changed the authors' own interpretation, which is the outcome peer review exists to produce. Reviewer 2's concern about independent coding of the failure taxonomy is only partly resolved; the authors have said so in the text rather than obscuring it, and the re-coding agreement of 0.71 is adequate for the descriptive use to which the taxonomy is put. The Editor-in-Chief is an author and took no part in reviewer selection or in this decision.

Appendix J. Data dictionary for the released dataset

The released dataset contains one row per execution, 14,400 rows in total. Column definitions follow. Where a column is null, the reason is given, because silently absent values have caused misinterpretation of released agent datasets in the past.

Column	Type	Definition and null conditions
execution_id	string	Stable unique identifier for one execution.
workflow_id	enum	W1–W12 as defined in Table 2.
architecture	enum	A1, A2 or A3 as specified in Section 3.
task_instance_id	string	Identifies the underlying task; appears once per architecture.
difficulty_stratum	int	1–4, the systems-touched proxy used for stratified sampling.
outcome	enum	completed, incorrect, budget_exhausted, harness_error.
completion_score	int	0 or 1 after adjudication. Null for harness_error rows.
side_effect_score	int	0 or 1. Null for read-only workflows (W4, W5, W7, W8, W9, W11).
usability_score	int	0 or 1 after adjudication.
rater_a_scores	json	Pre-adjudication scores from the first rater.
rater_b_scores	json	Pre-adjudication scores from the second rater.
adjudicated	bool	True where the two raters disagreed on any axis.
steps_used	int	Actions consumed, capped at 30.
tool_calls	int	Executed calls; excludes rejections under A3.
rejected_calls	int	Schema rejections. Always 0 for A1 and A2.
tokens_in	int	Prompt tokens summed across all model calls.
tokens_out	int	Completion tokens summed across all model calls.
cost_usd	float	Computed from token counts at list prices as of 1 July 2026.
latency_total_s	float	Wall-clock seconds from dispatch to termination.
latency_model_s	float	Seconds attributable to model inference.
latency_tool_s	float	Seconds attributable to tool execution.
latency_overhead_s	float	Residual orchestration time.
failure_mode	enum	M1–M8 per Table 7. Null where outcome is completed.
failure_mode_recoded	enum	Independent re-coding; populated for the 20% sample only.
transcript_release	enum	verbatim, paraphrased or withheld, per organisational consent.
executed_at	timestamp	UTC; useful for detecting external-system drift.

Table J1. Data dictionary for the released execution-level dataset.

Two derived tables are also released: a workflow-by-architecture aggregate reproducing Tables 4 to 6, and an adjudication log containing the third rater's written justification for each of the 1,096 adjudicated executions. We recommend that any reanalysis begin from the execution-level file rather than the aggregates, since the aggregates embed our analytical choices and the execution-level file does not.

Appendix K. Ethics, governance and data protection

The study involved no human subjects in the regulatory sense: the human participants were professional raters acting in their employed capacity, and the tasks were drawn from organisational backlogs rather than from individuals. It nonetheless raised three governance questions that we record here because they will recur in any comparable study.

K.1 Personal data in task content

Four workflows — W3, W5, W6 and W10 — operate on records containing personal data. Staging systems were seeded from production snapshots processed through a pseudonymisation pipeline that replaced names, contact details and account identifiers with consistent surrogates, preserving referential integrity so that reconciliation tasks remained meaningful. Raters saw only pseudonymised content. No personal data left the contributing organisation's environment, and released transcripts are drawn from the pseudonymised layer.

K.2 Write access to organisational systems

Executions with side effects ran against staging systems only. Each execution was wrapped in a transaction rolled back on completion, and a nightly consistency check verified that staging state matched its seed. At no point did an evaluated agent hold credentials to a production system. We regard this as a minimum standard and note that it was the single largest source of setup cost in the study.

K.3 Rater welfare and labour

Eleven raters scored transcripts over eleven weeks. Scoring long agent transcripts is tedious work with a measurable attention decay; we capped scoring at three hours per rater per day and randomised the order in which conditions were presented so that any decay would not accumulate against a particular architecture. Raters were employees of participating organisations, compensated at their normal rate, and were free to withdraw. Two did, and their partial scores were discarded rather than reused.

K.4 Conflict of interest management

One author is Editor-in-Chief of the journal in which this article appears. The submission was handled end to end by an independent handling editor who selected reviewers, communicated with the authors and issued the decision; the Editor-in-Chief had no access to reviewer identities and no visibility of the editorial record until after acceptance. The editorial timeline in Appendix I is published in full so that readers may judge whether the handling was ordinary. We are aware that a recusal statement is easy to write and hard to verify, and we consider the published timeline the more meaningful disclosure.

K.5 Funding

The study received no external funding. Compute costs of approximately US$34,000 were borne by the participating organisations in proportion to the workflows each contributed. No vendor of an orchestration framework participated in the design, execution, analysis or review of this study, and no author holds an equity interest in such a vendor.

Appendix L. Notes on terminology and framing

A reviewer observed that the word 'agentic' does substantial unexamined work in this literature, and we close with a short note on our usage, because the framing affects how the results should be read.

We use 'agentic' descriptively, to mean that the system selects and orders its own actions over multiple turns. We do not intend any claim about autonomy, intent or capability, and we have avoided the vocabulary of agency — deciding, wanting, trying — except where quoting a transcript. This is not fastidiousness for its own sake. The vocabulary of agency invites a particular diagnosis of failure, in which an unreliable system is one that needs better judgement, and it therefore invites prompt engineering as the remedy. Our data support a different diagnosis: most failures in our corpus were not lapses of judgement but consequences of a structure that permitted an invalid action to be taken, an error to be discarded, or a state to be revisited indefinitely. Those are properties of the harness.

The practical implication is a reallocation of engineering attention. In the deployments we observed, the ratio of effort spent on prompt revision to effort spent on the orchestration layer was, by the participants' own estimates, somewhere between five and twenty to one. Our results suggest that ratio is inverted with respect to the available returns. That is the single claim we would most like a reader to take from this article, and it is the claim we would most like to see tested somewhere other than here.

Appendix M. A checklist for reviewers of agent evaluation studies

We were asked during review to state what we would want to see in a comparable study, and the request produced a more useful artefact than we expected. The checklist below is offered to reviewers and editors handling submissions in this area. We fail two of our own items and say so.

Is the model configuration held fixed across compared conditions?
Comparisons that vary model and architecture together cannot attribute an effect to either. We hold model, temperature, top-p and retrieval index constant.

Is the prompt held as constant as the architectures permit, and is the residual measured?
Perfect parity is usually impossible. Section 11.2 measures our residual at roughly three percentage points of a nineteen-point effect.

Are step counts realistic for the claimed setting?
Enterprise workflows in our corpus run six to twenty-two steps. Results from three-step benchmarks should not be generalised to them, because the compounding mechanism scales with horizon.

Do any tasks have side effects, and were they executed rather than simulated?
Several failure modes concerning idempotence and ordering are invisible in read-only evaluation. Six of our twelve workflows write to external systems.

Is scoring human, automated, or model-based, and is the choice justified?
Model-based scoring rewards well-formed output and is unsafe where success depends on external state. We use two human raters.

Is inter-rater agreement reported per axis?
Unreported rater variance makes small differences uninterpretable. We report κ of 0.79, 0.84 and 0.61.

Is clustering in the task sample accounted for?
Tasks within a workflow share tools, corpus and context. We make workflow-level pairing primary, reducing our effective n to twelve.

Is cost reported per completed task as well as per attempt?
Per-attempt cost systematically flatters unreliable architectures. The two figures move in opposite directions in our data.

Is latency reported at percentiles rather than as a mean or median alone?
Median-only reporting concealed a 3.4-fold difference in our p95 results.

Are failure modes coded, and by whom?
We fail this item partially: our taxonomy was developed and applied by the authors, with only a 20% independent re-coding at $\kappa = 0.71$.

Are boundary conditions where the favoured design loses reported prominently?
W11 is our counterexample and appears in Section 1.2, Table 9 and the conclusion.

Is the sample of organisations characterised, and its unrepresentativeness stated?
Five organisations with mature staging practice. Absolute rates should not transfer; the ordering should.

Is a raw, execution-level dataset released with a data dictionary?
Appendix J. Aggregates alone prevent independent reanalysis.

Could a reader replicate on their own workflows with a stated effort budget?
Appendix G.3 estimates two to three weeks of one engineer's time.

Are conflicts of interest disclosed with a verifiable editorial record?
We fail this item in form and attempt to compensate in substance: an author is the journal's Editor-in-Chief, and the full editorial timeline is published in Appendix I.

Two general remarks. First, most items on this list are cheap; the expensive ones are human scoring, side-effect execution and independent failure coding, and of those the third is the cheapest to fix and the most commonly skipped, including by us. Second, a study that fails several items may still be worth publishing. The purpose of a checklist is to make the gaps visible in the text rather than to gate the literature, and a paper that states plainly where it is weak is more useful than one that leaves a reader to infer it.

Colophon and licence

Journal of Innovation, Product, Computing & Emerging Technologies (JIPCET)
ISSN 2998-4475 · Volume 3, Issue 2 · November 2026 · Published quarterly since 2024

JIPCET is an independent, open-access, peer-reviewed journal publishing research at the intersection of computing, product development and emerging technologies. All research articles undergo double-blind review by at least two independent reviewers. Editorial decisions are taken by human editors; where automated tools assist screening, metadata extraction or reviewer matching, their use is disclosed and no decision rests on them.

Persistent identifier. https://doi.org/10.5281/jipcet.2026.0042

Article history. Received 3 August 2026; revised 12 October 2026; accepted 28 October 2026; published online 14 November 2026. Time from submission to first decision: 47 days.

Corrections and retractions. Corrections are published as linked notices and the original record is never silently altered. Retraction notices state the reason and who requested the retraction. Readers wishing to raise a concern about this article should write to the editorial office, which will acknowledge within five working days.

Archiving. The published version of record, the supplementary material and the editorial history are deposited together. The article may be self-archived by its authors in any repository without embargo.

Citation. Paxton, D., Okonkwo, R., & Marsh, S. (2026). Evaluating agentic AI architectures for enterprise applications. *Journal of Innovation, Product, Computing & Emerging Technologies*, 3(2), 1–40. https://doi.org/10.5281/jipcet.2026.0042

Typeset in DejaVu Serif and DejaVu Sans by the JIPCET editorial production office. Set on A4 with a single 170 mm measure. This document contains 40 pages.