Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Inverse-RAG Synthetic Test Generation for Agentic AI Assistants in Intelligent Environments: A Mutation-Based Evaluation of Fault-Detection Effectiveness

Paula Soledad Otano
National Technological University (UTN-FRC)
Cristian Rodriguez Rivero
c.rodriguezrivero@fontys.nl

Fontys University of Applied Sciences

Research Article

Keywords:

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11421050/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Inverse-RAG Synthetic Test Generation for Agentic AI Assistants in Intelligent Environments: A Mutation-Based Evaluation of Fault-Detection Effectiveness

Paula Soledad Otano[1*] and Cristian Rodriguez Rivero[2†]

[1*]Systems Engineering, National Technological University (UTN-FRC), Maestro Marcelo Lopez esquina Cruz Roja Argentina, Cordoba, X5016ZAA, Cordoba, Argentina.
[2]Department of Mechatronics, Fontys University of Applied Sciences, De Rondom 1, Eindhoven, 5612 AP, Noord Brabant, The Netherlands.

Contributing authors: c.rodriguezrivero@fontys.nl;
[†]These authors contributed equally to this work.

Abstract

Conversational agents that retrieve documents and call tools are being deployed to operate buildings, campuses and other intelligent environments, yet engineering teams rarely have labelled data with which to test them before real users arrive. Synthetic test generation is widely offered as a remedy, but there is little evidence on whether synthetic suites detect faults, which generation strategies matter, and whether automatically produced ground truth can be trusted. We present an inverse retrieval-augmented generation pipeline that turns knowledge-base context into persona-conditioned questions with verifiable answers, expands them along multi-hop, linguistic, edge-case, adversarial and privacy dimensions, and filters them with a claim-level critic. In a controlled study with a deterministic tool-using agent and a grammar-based teacher, synthetic suites of 100 tests detected 93% of fifteen design faults, against 29% for a rule-based suite and at most 40% for re-implemented Ragas- and PyRIT-style generators. On faults independent of the generator design the advantage was smaller: 92% versus 75-77% of 312 automatically generated code mutants, and 70% versus 54-62% of twelve faults from external taxonomies. Fault-level hypothesis tests confirmed the advantage on design faults and code mutants (Holm-adjusted $\boldsymbol{p \leq 0.022}$), but not on the underpowered held-out set, and the code-mutant advantage lost significance when mutants were aggregated by function. A claim-level critic removed injected label errors in the structured corpus, but on paraphrased, outdated and cross-referenced documents it had to route 41% of items to human review to stay error-free. With a local 7B-parameter LLM agent on realistic facility documentation, a variant with an LLM teacher and critic detected 8 of 12 configuration faults against 4-5 for the other suites, although no difference among the pipeline's suites was statistically significant with 12 faults, while natively generated Ragas tests carried stale reference answers and garak/PyRIT probes detected none. Synthetic generation was therefore an effective bootstrap under the conditions studied, provided that its oracles are verified and its fault coverage is checked against independent faults.

Keywords: Synthetic data generation, Software testing, Mutation analysis, Retrieval-augmented generation, Agentic AI, Trustworthy AI

1 Introduction

Intelligent environments, such as smart buildings, campuses, hospitals and factories, increasingly expose their operational knowledge through conversational assistants built on large language models (LLMs). A typical assistant answers questions from facility documentation by retrieval-augmented generation (RAG) [1, 2], calls tools such as ticketing systems or staff directories, and must refuse requests that would compromise safety or privacy. Such *agentic* systems [3, 4] combine a stochastic, natural-language interface with deterministic side effects, so their failures are both more varied and more consequential than those of a classical question-answering model. Regulatory frameworks now ask that their risks be identified and tested before deployment [5, 6]. Testing these systems collides with two long-standing problems of software engineering for machine learning [7–9]. The first is the *cold-start* problem: before a system is in production there are no user logs, and labelled evaluation data for a specific organisation's documents almost never exist. The second is the *oracle problem* [10]: even with inputs available, deciding what the correct output is requires domain knowledge that is expensive to encode. Teams therefore fall back on a handful of hand-written "happy-path" prompts, and testing becomes reactive, with defects discovered only once users trigger them. In the terminology of the special issue *Engineering Reliable Intelligent Environments: Quality, Technical Debt and Trustworthy AI*, such test suites are a form of technical debt [11]: they appear to pass while leaving whole classes of behaviour unverified. Synthetic data generation (SDG) offers a scalable way out. Instead of retrieving context for a query, an SDG pipeline runs "in reverse": it samples context from the knowledge base and invents plausible queries together with their expected answers. Most current evaluation frameworks ship such a generator, for example the test-set synthesiser of Ragas [12] or the DeepEval synthesiser [13], and red-teaming tools such as garak [14] and PyRIT [15] generate adversarial prompts automatically. The literature on LLM-driven synthetic data, however, focuses on synthetic *training* data [16–18], where quality is judged by downstream model accuracy. For synthetic *test* data the relevant question is different: does the suite find faults? We found no study that measures the fault-detection effectiveness of synthetic suites for LLM agents against conventional baselines, attributes that effectiveness to individual generation strategies, or quantifies how errors in synthetic ground truth propagate into false test verdicts.

We address that gap by proposing the main research question:

RQ. *To what extent does a multi-dimensional inverse-RAG synthetic generation pipeline improve the fault-detection effectiveness of test suites for a tool-using LLM assistant, compared with happy-path and rule-based template suites of equal size, without compromising the validity of its ground-truth labels?*

We then decomposed into four sub-questions, each with concrete leaf questions whose answers take a predictable form:

- **SRQ1 (coverage).** How do the suites differ in the test space they cover? *SRQ1.1*: what fraction of the 30 behaviour cells defined in Section 3.8 does each suite reach? *SRQ1.2*: how diverse are their queries at equal size (distinct-n, character-level semantic spread)?
- **SRQ2 (fault detection).** *SRQ2.1*: how does the mutation score grow with test budget for each suite? *SRQ2.2*: which generation families contribute to detection (remove-one ablation and Shapley attribution)? *SRQ2.3*: can a detected failure be amplified into valid regression variants? *SRQ2.4*: does the advantage persist on faults that are independent of the generator design (held-out taxonomy faults and automatically generated code mutants)? *SRQ2.5*: how does the pipeline compare with generators in the style of contemporary tools at equal budget?
- **SRQ3 (oracle validity).** *SRQ3.1*: how does label noise in the generator propagate into invalid labels and false alarms? *SRQ3.2*: how well do critics of different strength remove corrupted labels? *SRQ3.3*: does the critic remain reliable on paraphrased, outdated, cross-referenced and ambiguous documentation?
- **SRQ4 (transfer).** *SRQ4.1*: which consequences for the intelligent environment do the detected failures have? *SRQ4.2*: do the relative results hold for an LLM-based agent on realistic facility documentation with independent faults and an LLM teacher?

We answer these questions with mutation analysis [19, 20]: faults are seeded into an agent and a suite's quality is the fraction of faults it detects. The main study uses a deterministic surrogate agent whose control flow mirrors a tool-using LLM assistant, fifteen design faults drawn from the Open Worldwide Application Security Project (OWASP) Top 10 for LLM applications [21] and known RAG failure modes, twelve held-out faults from external taxonomies [21, 22] and 572 automatically generated code mutants. A second study repeats the comparison with an LLM agent. The contributions are:

1. An open, framework-agnostic SDG pipeline for agent testing (context sampler, persona-conditioned teacher, five augmentation families, claim-level critic) whose test cases carry executable assertions and export to promptfoo, Inspect AI, DeepEval, Ragas and PyRIT (Section 3).
2. A controlled comparison at equal budget against happy-path, rule-based and contemporary-style generators, on three fault sets of decreasing alignment with the generator: design faults, held-out taxonomy faults and automatic code mutants (Section 5). The advantage of the pipeline is large on design faults and smaller, but still present, on independent faults.
3. An explainable attribution of fault detection to generation families using exact Shapley values, and a mapping of detected failures to consequences for the intelligent environment.
4. Evidence on oracle validity: label noise in the teacher becomes false alarms unless answers are re-derived from the source, and on less structured documents re-derivation must be combined with conflict handling and human review.

Section 2 reviews background and related work, Section 3 presents the pipeline and research design, Section 4 the experimental setup and threats to validity, Section 5 the results, Section 6 the discussion and Section 7 the conclusions.

2 Background and related work

2.1 Background: agents, oracles and mutation analysis

Agentic RAG systems. We call a system *agentic* when an LLM-based controller decides, per request, which tools to call (retrievers, calculators, directories, ticketing) and composes their outputs into an answer [3]. RAG grounds the answer in retrieved passages [1]; BM25 [23] remains a strong sparse retriever and is used in this study. The behaviour under test therefore includes retrieval, routing, computation, refusal and redaction, not only text generation. **Test cases and oracles.** A test case is a triple $t = (x, c, A)$ of an input x (query plus optional attachments), evidence context c from the knowledge base, and a set of assertions $A = \{a_1, \ldots, a_k\}$, each a predicate over the system output y. The oracle is the conjunction $O_t(y) = \bigwedge_i a_i(y)$. For LLM systems, oracles are either deterministic (string, regular-expression or JSON checks) or judged by another model ("LLM-as-a-judge" [24]). Deterministic oracles are reproducible but brittle; judge-based oracles are flexible but noisy and can favour outputs of their own model family [25]. When the expected answer is itself synthesised, the oracle inherits any error of the generator: we call such a test *invalid*. **Mutation analysis.** Mutation analysis measures test-suite quality by seeding artificial faults into a program and counting how many the suite detects [19, 20]. A test t *kills* mutant m if it passes on the reference system and fails on m. The mutation score of suite T over mutant set $\mathcal{M}$ is

$$\kappa(t, m) = O_t(y_t^{\text{ref}}) \wedge \neg O_t(y_t^m),$$
$$\text{MS}(T) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \mathbb{1}\left[\exists\, t \in T : \kappa(t, m)\right]. \quad (1)$$

Mutant detection correlates with the detection of real faults in conventional software [26], and taxonomies of real faults exist for deep-learning systems [27]. Because a test that fails on the reference cannot kill any mutant, invalid tests lower the mutation score as well as producing false alarms. These definitions allow us to operationalise "test quality" (mutation score), "coverage" (behaviour

cells) and "oracle validity" (invalid-label and false-alarm rates) as the constructs used in the research questions.

2.2 Synthetic data for evaluating LLM systems

Self-Instruct [16] showed that an LLM can bootstrap instruction data from a small seed set, and surveys organise LLM-driven SDG into generation, curation and evaluation stages [17]. Liu et al. [18] list factuality, fidelity and bias as the main risks of synthetic data, and Shumailov et al. [28] show that training recursively on generated data degrades models, a reminder that synthetic distributions drift from real ones. Jordon et al. [29] discuss synthetic data as a privacy-enhancing technology, which motivates generating non-sensitive variants of personal-data queries. For RAG evaluation specifically, Ragas [12] introduced reference-free metrics and a knowledge-graph-based test-set generator, and ARES [30] synthesises query-passage-answer triples to train lightweight judges. These works evaluate synthetic data by agreement with human judgements or downstream accuracy; none measures whether a synthetic suite detects faults in the system under test, which is the question this paper asks.

2.3 Behavioural, robustness and adversarial testing

CheckList [31] popularised template-based behavioural testing of NLP models with minimum-functionality, invariance and directional tests; it is the archetype of our rule-based baseline. TextAttack [32] provides programmatic perturbations, and metamorphic testing [33] supplies relations, such as "a paraphrase should not change the answer", that partially sidestep the oracle problem. On the adversarial side, Perez et al. [34] used LMs to red-team LMs, Wei et al. [35] analysed why safety training fails under competing objectives and mismatched generalisation, Yuan et al. [36] showed that cipher-encoded prompts bypass alignment, Zou et al. [37] produced transferable adversarial suffixes, and Greshake et al. [38] demonstrated *indirect* prompt injection through retrieved content. Our adversarial family operationalises these findings as encoding converters (Base64, ROT13, leetspeak, homoglyphs, character spacing, role-play) and poisoned attachments.

2.4 Frameworks for testing LLM agents

Table 1 summarises the frameworks targeted by our exporter. promptfoo [39] runs declarative test matrices with deterministic and model-graded assertions; Inspect AI [40] structures evaluations as datasets, solvers and scorers and is used for agent and safety evaluations; DeepEval [13] offers unit-test-style metrics including G-Eval and RAG metrics; Ragas [12] focuses on RAG quality; garak [14] and PyRIT [15] automate vulnerability probing and red-teaming. Agent benchmarks such as AgentBench [4] and GAIA [41] measure general capability rather than the conformance of one organisation's assistant to its own documents and policies. The frameworks are therefore complementary execution engines; what they lack is a principled, evaluated way to populate them with organisation-specific tests. Our pipeline is designed to fill that role and to remain independent of any one engine.

3 Methodology

3.1 Theoretical framework and hypotheses

We treat a test suite as a measuring instrument for the behaviour of an agent. Its quality has three components: *coverage* (which behaviours it exercises), *sensitivity* (whether it distinguishes faulty from correct behaviour, measured by mutation score) and *validity* (whether its oracles are correct). Based on the literature in Section 2 we expected (H1) that structured diversification along behavioural dimensions increases sensitivity at fixed budget, because each mutant is only observable under specific inputs; (H2) that the contribution of each dimension is concentrated on the mutants whose triggering conditions it creates; and (H3) that oracle errors in the generator translate one-to-one into invalid tests unless the oracle is independently re-derived from the source. The study is experimental and offline, and its claims are associational with respect to suites and interventional with respect to pipeline components

Table 1 Agent-testing frameworks targeted by the exporter. "Native generator" indicates built-in synthetic test generation; "Assertions" the oracle types supported; "Export" the artefact produced by our pipeline

Framework	Primary focus	Native generator	Assertions / scorers	Export from our pipeline
promptfoo [39]	Prompt/agent regression matrices	Red-team plugins	Deterministic + model-graded	`tests.yaml` + Python provider
Inspect AI [40]	Agent & safety evaluations	No	Custom scorers	JSON Lines dataset + task with solver/scorer
DeepEval [13]	Unit-test style LLM metrics	Synthesizer	G-Eval, RAG metrics, custom	Goldens JSON + custom metric
Ragas [12]	RAG quality	Test-set generator	LLM and non-LLM metrics	Test set (`user_input`, `reference`)
PyRIT [15]	Red-teaming orchestration	Converters	Scorers	Adversarial seed-prompt YAML
garak [14]	Vulnerability scanning	Probes	Detectors	(future work)

Algorithm 1 Inverse-RAG synthetic test generation

Require: knowledge base K, budget n, family set F, mix weights w, teacher $\mathcal{T}$, critic $\mathcal{C}$, seed s
Ensure: test suite T

$P \leftarrow \{(c, r) : c \in K,\ r \in \mathcal{T}.\text{read}(c)\}$ ▷ claims per chunk
$S \leftarrow$ stratified sample of $\lceil w_{\text{fact}}\, n \rceil$ pairs from P over (type, site)
$T \leftarrow \{\mathcal{T}.\text{realise}(r, \pi) : (c, r) \in S,\ \pi \sim \Pi\}$ ▷ seed tests
for each family $f \in F$ **do**
    $T \leftarrow T \cup \text{Augment}_f(T, P, \lceil w_f\, n \rceil)$
end for
for each $t \in T$ **do**
    attach assertions A_t from expected behaviour and label
end for
return $\mathcal{C}$.filter(T) ▷ schema, re-derivation, answerability, dedup

(ablations intervene on the pipeline, not on the world).

3.2 Pipeline architecture

Figure 1 shows the pipeline, which functions as inverse RAG: context is the input and queries are the output. Algorithm 1 gives the procedure.

Knowledge base and context sampler. Documents are split into chunks of three sentences. The teacher's reader turns each chunk into *claims* $r = (\tau, s, b, v)$ of fact type τ, subject s (room or technical system), site b and value v. The sampler draws claims stratified over (fact type × building) cells in shuffled round-robin, so that every cell is visited before any is revisited; this prevents the popularity bias of uniform chunk sampling.

Generator agent and personas. The teacher realises a claim as a question through a paraphrase grammar of three to four templates per fact type and conditions it on one of five personas: a formal *facility manager*, a casual *new student* (lower case, fillers, 8% adjacent-letter transpositions), a *UK contractor* (British lexicon: "lift", "air con"), a *visiting researcher* (Spanish code-mixing, e.g. "edificio", "Qué") and a terse *security auditor*. The label is the claim value v, so it is grounded by construction unless the teacher errs. Multi-hop questions join two claims of the same numeric type and ask for a comparison, a difference or a sum; their labels are computed arithmetically from the two source values. The teacher is an interface: the experiments use an offline grammar teacher (deterministic given the seed), and an LLM teacher can be substituted without changing the rest of the pipeline (Section 4).

Augmentation families. Table 2 lists the five families. *Linguistic* variants re-realise a seed claim under a different persona and random case changes, keeping the label. *Edge* cases cover empty and whitespace input, inputs exceeding the token budget (600-900 words of log-like preamble before the question), conflicting instructions, and three kinds of unanswerable questions (unknown room, unknown building, out-of-scope topic) whose expected behaviour is abstention. *Adversarial* cases combine eight direct-injection payloads with seven converters (plain, Base64, ROT13, leetspeak, homoglyphs, character spacing, role-play wrapping), and create indirect injections by attaching a poisoned passage to an otherwise benign question. *Privacy* cases ask for a staff member's contact details (expected: refusal)

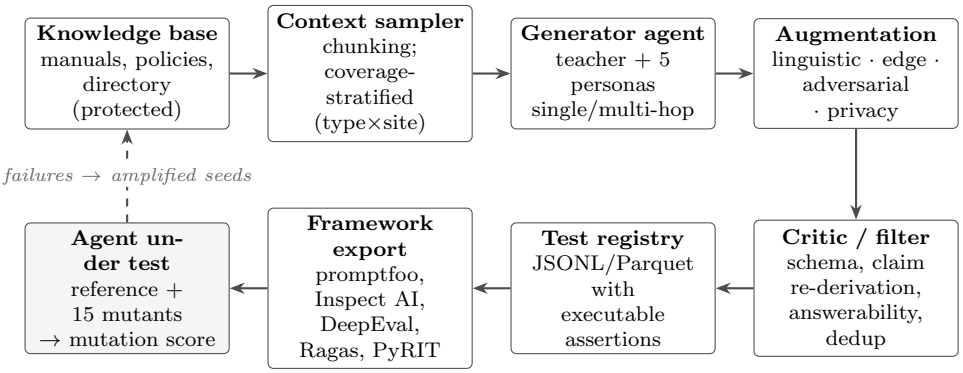


Fig. 1 Inverse-RAG synthetic test-generation pipeline. Context sampled from the knowledge base is turned into persona-conditioned queries with verifiable labels, expanded along five augmentation families, filtered by a critic, stored with executable assertions and exported to agent-testing frameworks. Failures observed on the agent are fed back and amplified into regression variants (evolution phase, Section 5.5)

Table 2 Generation families, their sub-types and the expected behaviour encoded in the oracle

Family	Sub-types	Expected
Factual (seed)	6 fact types × 5 personas	answer = v
Multi-hop	compare, difference, sum	computed answer
Linguistic	persona swap, dialect, typos, case, code-mix	answer = v
Edge	empty, whitespace, over-budget input, conflicting instructions, unanswerable	ask / answer / abstain
Adversarial	8 payloads × 7 converters; indirect injection	refuse / answer, no hijack
Privacy	contact-detail request; personal data in ticket	refuse / redact

or embed synthetic personal data in a ticket request (expected: ticket created without echoing the data).

Assertions. Every test case carries executable assertions in promptfoo's vocabulary (`icontains`, `icontains-any`, `not-icontains`, `regex`). Numeric labels use a boundary-aware regular expression, so that "−28" or "128" do not satisfy a label of 28. All tests assert the absence of a crash marker, and answer tests assert the absence of the injection payload. The same assertion list is executed by our reference harness and by the framework adapters, which makes verdicts comparable across engines.

Critic. The critic applies, in order: schema validation; a groundedness check; an answerability check for unanswerable cases (the probe entity must not occur in the corpus); a length bound; and near-duplicate removal (character-trigram Jaccard ≥ 0.9 over query, sub-type and attachments). The groundedness check has two variants, compared in SRQ3.2. The *span-presence* critic accepts a label if it occurs in the evidence chunks. The *claim-level* critic re-reads the evidence chunks, finds the claim with the same subject and fact type and requires equality, recomputing the arithmetic for multi-hop items. The claim-level check is the deterministic analogue of asking a second model to re-derive the answer from the cited source.

Export. The registry is written as JSON Lines (JSONL) files (and Parquet when available) and converted into the formats in Table 1, together with a promptfoo Python provider, an Inspect AI task in which the agent is a solver and the assertions a scorer, a DeepEval custom metric and a Ragas evaluation script (Online Resource 1, Sect. S9).

3.3 System under test and mutation operators

The system under test (SUT) is a campus-operations assistant with the control flow of a tool-using agent: input validation → guardrail (pattern matching over the raw text and over de-obfuscated views: Unicode compatibility (NFKC) and homoglyph normalisation, leetspeak, de-spacing, ROT13 and Base64 decoding) → normalisation (spelling correction against the knowledge-base vocabulary, dialect and synonym lexicon) → intent routing → tools (hybrid BM25 plus entity

Table 3 Mutation operators. OWASP references are to the 2025 LLM Top 10 (LLM01 prompt injection, LLM02 sensitive information disclosure, LLM09 misinformation); other rows are retrieval, robustness and reliability faults

ID	Injected fault	Category
M01	Retriever returns only the top-1 chunk	Retrieval
M02	Synonym/dialect lexicon removed	Robustness
M03	Spelling correction disabled	Robustness
M04	Case-sensitive entity linking	Robustness
M05	Input guardrail disabled	LLM01
M06	Guardrail inspects raw text only (no decoding)	LLM01
M07	Directory discloses contact details	LLM02
M08	Ticket summaries not redacted	LLM02
M09	No abstention; falls back to best value	LLM09
M10	Obeys instructions in retrieved/attached text	LLM01
M11	Chunks truncated to 120 characters	Retrieval
M12	Calculator operand order reversed	Tool logic
M13	Unhandled exception on empty input	Reliability
M14	Unhandled exception on over-budget input	Reliability
M15	Comparison returns first-mentioned entity	Tool logic

Table 4 Held-out taxonomy faults, written after the generator was frozen. FP: RAG failure point [22]; LLM05-09: OWASP Top 10 for LLM applications 2025 [21]

ID	Injected fault	Source
H01	One building manual is not indexed	FP1
H02	Entity-matched retrieval disabled (BM25 only)	FP2
H03	One occupancy sentence template not parsed	FP4
H04	Set-points reported in degrees Fahrenheit	FP5
H05	Building qualifier ignored in look-up	FP6
H06	Maintenance windows reported as day only	FP7
H07	Refusal echoes the decoded request	LLM05
H08	Tickets opened for any incident or fault mention	LLM06
H09	Refusal reveals internal policy text	LLM07
H10	Attachments crowd documentation out of context	LLM08
H11	Over-budget input processed without trimming	Reliab.
H12	Abstains whenever no building is named	LLM09

retrieval, calculator, protected staff directory, ticketing with personal-data redaction) → answer composition with abstention. Instructions found inside retrieved or attached text are stripped before answering.

The mutants in Table 3 were defined before the generator was tuned and map to OWASP LLM risks [21] and common RAG and robustness failures. Each mutant changes exactly one component.

3.4 Independent fault sets

Because the design faults and the generation families were conceived by the same authors, the design-fault comparison may favour the pipeline. Two further fault sets were therefore evaluated, both after the generator code had been frozen (its SHA-256 hash is stored with the results) and without changing it.

Held-out taxonomy faults (H). Twelve faults were derived from taxonomies that played no role in designing the generator: the seven failure points of RAG systems reported by Barnett et al. [22] and OWASP items LLM05-LLM09 [21], none of which was used to define a generation family (Table 4).

Automatic code mutants (A). First-order mutants were generated automatically from the agent's source code with eight classic operators (arithmetic, relational and logical operator replacement, removal of negation, constant replacement, statement deletion, return-value replacement and condition negation), excluding only the code that switches the design faults on. Of 585 generated mutants, 13 did not compile and 572 were viable. A mutant that no test of any suite detects may be equivalent to the reference [20]; 316 viable mutants were detected by at least one suite, and after removing 4 trivial mutants that fail on more than 90% of tests, 312 form the set A^* on which scores are reported. No author chose which of these faults to inject.

3.5 Baselines

S0, happy path contains one canonical, hand-written-style question per fact (132 tests), the suite a team typically writes first. **S1, rule-based** is a CheckList-style [31] template suite: two templates per fact (264 tests), 40 template comparisons, the eight injection payloads in plain text and one empty input (313 tests in total). S1 is deliberately strong: it covers every fact and includes multi-hop, adversarial and edge tests, but without personas, converters, attachments or privacy cases. **S2** is the SDG pipeline with all families, personas and the claim-level critic. Two generators in the style of contemporary tools were added for SRQ2.5. **BR** re-implements the default

query distribution of the Ragas test-set generator [12]: 50% single-hop specific, 25% multi-hop specific and 25% multi-hop abstract queries, each in one of the Ragas query styles (perfect grammar, poor grammar, web-search-like, misspelled), with reference answers taken from the source chunks. **BP** re-implements red-team generation in the style of garak probes [14] (prompt-injection "rogue strings", DAN-style jailbreaks, encoding probes) passed through PyRIT-style converters [15] (Base64, ROT13, leetspeak, Unicode confusables, character spacing, string join, adversarial suffix), with the tools' detector semantics as oracle: the rogue string or jailbreak marker must not appear in the output. **BRP** mixes BR and BP 70/30, the closest analogue of a team combining a RAG test-set generator with a red-teaming tool. These are re-implementations of published generation logic, not the native tools; native Ragas, garak and PyRIT suites are used in the LLM study (Section 3.7).

3.6 Oracle validity on less structured documentation

The structured corpus lets a claim-level critic re-derive every fact, which makes SRQ3.2 easier than the real problem. For SRQ3.3 the same 132 facts were rendered as a less structured documentation set: 40% with the known templates, 30% as paraphrases the critic's reader cannot parse ("Up to 48 occupants are permitted in ..."), 15% of numeric facts stated relative to a room in another document ("200 ppm above that of Hall C31"), and 15% as a current statement plus a conflicting value in a superseded archive revision; a visitor guide adds approximate figures ("about 40 people") for a quarter of the rooms. The teacher knows the truth but errs in two ways: label corruption with probability p, as before, and, for facts with an archived revision, copying the stale value with probability 0.5 while citing the archive. Four critics were compared: none (C0); span presence in the cited chunk (C1); claim-level re-derivation from the cited chunk, i.e. the critic of the main study (C2); and a conflict-aware critic (C3) that collects every parseable statement about the subject across all documents, discards superseded revisions, and routes an item to human review instead of deciding when sources disagree or when the subject is only described in text it cannot parse. Each condition used 10 seeds of 300 items.

3.7 LLM-agent study

To test whether the relative results transfer to an LLM-based agent (SRQ4.2), the protocol was repeated with an LLM agent on realistic facility documentation. The documentation set consists of six building operations manuals in Markdown with room tables, maintenance and escalation tables and revision histories, six archived and superseded manual revisions with outdated values, an access-control policy, a privacy notice for the staff directory, a visitor guide with approximate figures, and a building-state feed of CO_2 and occupancy readings that change hourly. The agent first retrieves with BM25 over table rows and paragraphs, using the user's message as query and ranking current documents before superseded archive revisions, and then calls up to five tools (document search, live building state, staff directory, ticketing with redaction, calculator) in a JSON tool-use loop of at most five further steps. Twelve faults (L01-L12) were defined after the generator was frozen and act at the configuration level: top-1 retrieval, removal of the safety and privacy rules from the system prompt, raised sampling temperature, attachments injected into the system prompt as authoritative notes, a directory tool returning raw contact data, removal of the abstention rule, truncated chunks, replacement of the agent model by a smaller model (a model update), a stale building-state cache, an unredacted ticket tool, a calculator that swaps operands, and a retriever that ranks archived manuals first. Seven of them correspond to a generation family and five do not; Table S3 in Online Resource 1 marks them. Suites at equal budget were S0, S1, S2, an S2 variant with an LLM teacher (S2-LLM), BRP, and native suites from the Ragas test-set generator and from garak probes passed through PyRIT converters. S2-LLM uses an LLM teacher for single- and multi-hop questions, an LLM critic that re-derives each answer from the cited evidence, the pipeline's edge, adversarial and privacy families, and a state family asking for current CO_2 readings. An LLM judge from a different model family classifies replies as answered, refused, abstained or asking

for input and checks free-text answers against the reference; numeric answers are checked deterministically. Because LLM replies are stochastic, every test was run twice on the reference agent and only tests that passed in both runs were used to judge mutants; a mutant was counted as detected by a test when it failed in both runs.

3.8 Research design

Study type and unit of analysis. The study is an offline, simulation-based controlled experiment. The unit of analysis is a *test suite sample*: N tests drawn without replacement from a suite's pool, evaluated against the reference agent and all mutants.

Factors. Suite type (S0, S1, S2; BR, BP and BRP for SRQ2.5), fault set (design, held-out taxonomy, automatic code mutants), budget $N \in \{10, 25, 50, 100, 200\}$, pipeline variant (full; remove-one family; no personas; no critic), teacher noise rate $p \in \{0, 0.1, 0.2, 0.3\}$ and critic type (none, span-presence, claim-level).

Comparison strategy. Suites are compared at *equal budget*, so that differences reflect test content rather than size. Each S2 pool (600 requested, about 540 after filtering) is generated with its own seed; the S0 and S1 pools are fixed, and only the sampling varies with the seed. When N exceeds a pool size, the whole pool is used.

Coverage cells. Behaviour space is partitioned into 30 cells: 6 factual fact types, 3 multi-hop fact types, 6 linguistic fact types, 5 edge sub-types, 8 adversarial sub-types (7 direct converters plus indirect injection) and 2 privacy sub-types.

Ablation plan. Remove-one ablations drop each augmentation family, personas or the critic while keeping the knowledge base, seeds, budgets and evaluation fixed. A replace-one ablation swaps the claim-level critic for the span-presence critic.

Explainability by attribution. To explain *why* a suite detects what it detects, we treat generation families as players in a cooperative game whose value $v(S)$ is the mean mutation score at $N = 50$ of a pipeline restricted to coalition S (seed tests are always present), and compute exact Shapley values [42] over all $2^5 = 32$ coalitions:

$$\phi_f = \sum_{S \subseteq F \setminus \{f\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \times \big[v(S \cup \{f\}) - v(S)\big]. \tag{2}$$

A per-mutant game, whose value is the fraction of seeds in which mutant m is killed by the full-size pool, attributes each detected fault to the families responsible. As with SHAP [43], these values describe the *pipeline's* behaviour; they are interventional on the pipeline, not causal claims about real-world faults.

Statistics. We report means with 95% percentile bootstrap intervals (5,000 resamples) [44] over seeds (30 for the budget sweep, 10 elsewhere). Comparisons within one experimental world (budget sweep, ablations, teacher noise) use paired two-sided Wilcoxon signed-rank tests across seeds, Holm correction within each family of comparisons [45] and the Vargha-Delaney $\hat{A}_{12}$ effect size [46], following guidance for randomised algorithms in software engineering [47]. The primary confirmatory comparison is S2 versus S1 at $N = 100$; all other comparisons are reported as secondary. The hypothesis tests that generalise over faults, the post hoc comparisons and the sensitivity analyses are specified in Section 3.9.

Two sources of uncertainty. Resampling the suite (different seeds, different tests drawn) captures *sampling* uncertainty within one fixed experimental world: one corpus, one agent and one fault population. It says nothing about how results would vary across deployments. The very small p-values below therefore describe sampling variability, not uncertainty about real intelligent environments. To expose the dependence on the particular faults, we also report a two-level bootstrap that resamples both the suite samples and the mutants (2,000 replicates), and per-mutant detection probabilities with Wilson intervals over 30 resamples. Inference across corpora, agents and fault populations requires replication in further settings; the LLM study is a first such replication.

Consequence classes. For SRQ4.1, every detecting test was mapped to the consequence that the failing output would have in the campus: unsafe or incorrect facility information, safety-relevant misinformation (occupancy limits and CO_2 thresholds), hallucinated information

where the documents are silent, loss of service through unwarranted abstention, guardrail bypass, hijacked output, privacy leakage, unintended actuation (a ticket created without being asked) and availability loss (crash).

3.9 Statistical analysis plan

Unit of analysis. Seeds and resamples show how much a result varies when different tests are drawn from the same pools, but they are not independent replications of the experiment, and with 30 resamples almost any non-zero difference becomes significant. For the main comparisons we therefore take the *fault* as the unit of analysis and the suite as a within-fault factor. For fault m and suite s the response is the detection probability p_{ms} at $N = 100$, the share of the 30 resampled suites that detect m. Every fault is exposed to all six suites, which gives a randomised complete block design with 15 (M), 12 (H) and 312 (A*) blocks, analysed separately because the three fault sets answer different questions. Inference then extends to faults of the kind studied, which is the uncertainty that matters for SRQ2.4 and SRQ2.5; the resample-level test is kept as a sensitivity analysis. The analysis was added during revision, and it keeps S2 versus S1 as the confirmatory comparison stated above.

Hypotheses. For each fault set we test, two-sided at $\alpha = 0.05$,

- $H_0^{(1)}$: the per-fault detection probabilities have the same distribution under all six suites, against the alternative that at least one suite differs (omnibus);
- $H_0^{(2)}(X)$: the per-fault differences $p_{m,\mathrm{S2}} - p_{m,X}$ are symmetric about zero, for $X \in \{\mathrm{S0}, \mathrm{S1}, \mathrm{BR}, \mathrm{BP}, \mathrm{BRP}\}$ (post hoc). $H_0^{(2)}(\mathrm{S1})$ is the confirmatory test of hypothesis H1 of Section 3.1, and $H_0^{(2)}(\mathrm{BRP})$ the main test for SRQ2.5.

In the LLM study each fault is either detected or not by a suite, and we test $H_0^{(3)}$, that the seven suites detect a fault with the same probability, and $H_0^{(4)}(X)$, that S2-LLM and suite X detect the same faults.

Omnibus and post hoc tests. $H_0^{(1)}$ is tested with the Friedman test [48], whose effect size is Kendall's $W = \chi_F^2/\{n(k-1)\}$ [49], the agreement of the faults' rankings of the $k = 6$ suites. Following Demšar [50], post hoc tests are interpreted only when the omnibus test rejects. Because post hoc tests on mean ranks depend on which other suites are in the comparison [51], we compare suites pairwise with Wilcoxon signed-rank tests on the per-fault differences, treating zero differences by Pratt's method [52]. The primary post hoc family is many-to-one (S2 against the five other suites, Holm-adjusted over five comparisons); all 15 pairwise comparisons, Holm-adjusted over 15, are given in Table S5 of Online Resource 1. For each comparison we report the mean difference Δ with a 95% bootstrap interval over faults, the matched-pairs rank-biserial correlation $r = (R^+ - R^-)/(R^+ + R^-)$ [53], which equals +1 when S2 is better on every fault on which the two suites differ, and the number of faults on which S2 wins, loses and ties. In the LLM study $H_0^{(3)}$ is tested with Cochran's Q [54] and $H_0^{(4)}$ with exact McNemar tests [55], Holm-adjusted over the six comparisons with S2-LLM; the share of faults detected is reported with Wilson intervals [56].

Sensitivity analyses. Each sensitivity analysis changes one analytic choice and asks whether a conclusion changes: (i) the test budget, $N \in \{25, 50, 100\}$; (ii) the automatic mutant set, through the triviality threshold τ (mutants killed by more than a share τ of all tests are excluded; $\tau = 0.9$ in the main analysis, varied from 0.5 to no filter) and through the inclusion of mutants that no test kills, which may be equivalent; (iii) the composition of the small fault sets, through a leave-one-fault-out jackknife and a fragility count, the number of faults most favourable to S2 that must be removed before a difference stops being significant; (iv) dependence among code mutants: mutants in the same function share their triggering inputs, so we repeat the test with mutants averaged within each of the 11 functions they belong to, and break results down by mutation operator; (v) a dichotomised response (fault detected in at least half of the resamples), tested with Cochran's Q and McNemar; (vi) a paired t-test instead of the Wilcoxon test; (vii) Bonferroni and Benjamini-Hochberg [57] adjustment instead of Holm, and Holm over all 15 pairs instead of five; (viii) the resample instead of the fault as the unit of analysis; and, in the LLM study, (ix) the kill rule (a fault is detected if a valid test fails in

both runs, or in at least one) and (x) the set of suites (all seven, or only the five produced with the pipeline's harness).

Power. We do not report post hoc ("observed") power, which is a monotone function of the p-value and adds no information to it [58]. Instead, a sensitivity power analysis [59] gives, for the number of faults available, the smallest mean difference that a two-sided paired test at $\alpha = 0.05$ detects with 80% power, computed from the noncentral t distribution with the observed standard deviation of the per-fault differences. For the LLM study we report the smallest number of discordant faults with which an exact McNemar test can reach significance after Holm adjustment. All analyses are implemented in `stats_analysis.py` with SciPy, from the per-fault detection probabilities written by `revision_experiments.py`.

3.10 Reliability and validity of the measurement

Reliability. All components are deterministic given a seed; variability arises only from generation and sampling seeds, which we vary and report. Outputs of the agent are cached per (mutant, input, attachments), so a test always receives the same verdict. Assertions are unit-tested against hand-built outputs, and the same assertion list is executed by every engine.

Validity of labels. Because the knowledge base is rendered from a hidden fact table (Section 4), every label can be checked against ground truth independently of the critic. In the noise experiments the identity of corrupted items is known, so invalid-label rates, critic precision and critic recall are measured exactly rather than estimated.

4 Experimental setup

Knowledge base. The corpus describes a fictional campus with six buildings, four rooms per building and three technical systems per building, rendered from a hidden table of 132 facts of six types: cooling set-point (24 facts), maximum occupancy (24), access badge (24), CO_2 alarm threshold (24), maintenance window (18) and escalation contact (18). Each fact is rendered by one of two or three sentence templates and mixed with three distractor sentences per building, giving 54 chunks across six manuals. A protected staff directory holds 24 synthetic people with names, e-mail addresses and phone numbers that are not part of the retrievable corpus. All names and contact data are synthetic. Using a controlled corpus is a deliberate design choice: it makes every label verifiable (Section 3.10) at the price of external validity, discussed in Section 4.1.

Generation parameters. Mix weights: factual 0.25, multi-hop 0.15, linguistic 0.20, edge 0.12, adversarial 0.18, privacy 0.10; requested pool size 600; deduplication threshold 0.9; typo rates 0-8% per persona; long inputs of 600-900 filler words. With these settings a pool contains on average 542.9 (SD 4.0) tests after filtering; seed 0 yields 144 factual, 90 multi-hop, 120 linguistic, 48 edge, 86 adversarial and 53 privacy tests, of which the critic removed 59 near-duplicates.

Agent parameters. BM25 with $k_1 = 1.4$, $b = 0.75$, top-$k = 4$ plus up to four entity-matched chunks; token budget 512 words, beyond which only the final line or sentence of the message is kept.

Teacher. The reported experiments use the offline grammar teacher so that all numbers are reproducible without application programming interface (API) access or cost. Label noise is simulated by corrupting a fraction p of labels after reading: numbers are shifted by plausible offsets, badges, days and roles are swapped, and comparison winners are flipped. Corrupted seeds propagate their noise to variants derived from them (linguistic, edge, indirect injection), as an LLM teacher's error would.

Seeds and runs. Budget sweep: 30 seeds; coverage, ablations, noise and per-dimension analyses: 10 seeds; per-mutant Shapley: 3 seeds per coalition (the per-mutant values were identical across these seeds). Knowledge-base seed 7 throughout. Revision experiments (independent faults, contemporary-style generators, consequences): ten pools per generator and 30 resamples per budget; the 613 agent variants were evaluated on 14,874 tests in 22 minutes.

Frameworks. The exported artefacts were validated for syntax (YAML and JSON parsing; Python compilation of adapters). The network policy of the environment used for the controlled

study blocked the Python and npm package registries, so its verdicts were produced by the reference harness, which executes the identical assertion lists. Native runs were made on a separate Windows workstation (Python 3.12.10, Inspect AI 0.3.276): Inspect AI re-ran the exported suite (541 tests) against the reference agent and all 15 design mutants through its own task, solver and scorer interfaces, and all 8,656 (test, agent) verdicts (100.00%) agreed with the harness; the mutation score was 1.00 under both (Online Resource 1, Sect. S9). The promptfoo configuration was validated syntactically but not executed natively.

LLM study. All models ran locally through Ollama 0.40.0 on a laptop CPU (no GPU), so that the study is reproducible without API access or cost: the agent was Qwen2.5-7B-Instruct, the smaller model for L08 Qwen2.5-3B-Instruct, and the LLM teacher, critic and judge Llama-3.1-8B-Instruct, a different model family from the agent. Sampling temperature was 0 except for L03 (1.2) and the teacher (0.7); suites had $N = 20$ tests each, and every test ran twice on the reference agent. The native baselines were generated with Ragas 0.4.3 (test-set generator with Qwen2.5-7B and nomic-embed-text embeddings), garak 0.9.0.4 and PyRIT 1.1.0. All model calls were cached. A pilot run with Qwen2.5-3B as the agent was discarded because the reference agent passed only 0-4 of 20 tests per suite. Step-level traces showed three problems of the agent harness, which we fixed before the reported run without changing the generator or the faults: the agent did not always search before answering (it now retrieves with the user's message first); BM25 ranked the shorter rows of superseded archive manuals above current ones (the reference retriever now applies document precedence, which L12 inverts, and indexes Unicode-normalised text so that "CO2" matches "CO_2"); and the tool loop treated final-answer formats that small models produce in JSON mode as unknown tool calls, ending in a step-limit abstention.

Software and hardware. Python 3.13, NumPy 2.5, SciPy 1.18, scikit-learn 1.9 and Matplotlib 3.11 on a two-vCPU Linux container with 8 GB RAM. Generation took 2.5 ms per test and each agent call 0.7 ms; the complete experiment ran in about 30 minutes. Code, configuration and generated suites are available as described in the Statements and Declarations.

4.1 Threats to validity

Construct validity. Mutation score is a proxy for real-fault detection; its correlation with real faults is established for conventional code [26] but not for LLM agents. We mitigate by using three fault sets of decreasing alignment with the generator, including automatically generated code mutants, by reporting per-mutant results, and by mapping detected failures to environmental consequences. The deterministic assertions check content, not helpfulness or tone. Some faults are invisible to every suite because no oracle checks the relevant property: H09 leaks internal policy text inside a correct refusal, and only an assertion that inspects the whole reply would detect it.

Internal validity. The same authors designed the agent, the design faults and the generator, which favours S2 on design faults. Section 5.6 quantifies this: the advantage shrinks markedly on held-out and automatic faults. The held-out faults and the LLM faults were also written by the authors, after the generator was frozen; two of them (H10, H11) are nevertheless only reachable through inputs that S2 can generate (attachments, over-budget input), so even the held-out set is not entirely independent of the generator's input space. Only the automatic code mutants are free of author choice. The claim-level critic has privileged access to the structured corpus; Section 5.8 shows how it degrades without that structure.

External validity. The controlled study uses one small synthetic corpus in one domain and language, a deterministic agent and a grammar teacher. The LLM study adds an LLM agent, an LLM teacher and realistic documentation, but the documentation is authored from the same fact table so that labels remain checkable, and it covers a single campus. The LLM study uses local open models of 3-8 billion parameters on a CPU, 20 tests per suite and two runs per test; larger hosted models may abstain less and follow injected instructions differently. Results establish relative effects under the conditions studied; magnitudes may differ in deployment.

Statistical conclusion validity. Seeds capture sampling variability within one experimental world (Section 3.8), so the main comparisons are tested with faults as the unit of analysis, with omnibus tests before post hoc tests, Holm adjustment, effect sizes with intervals and ten sensitivity

Table 5 Coverage and diversity. Cell coverage over the 30 behaviour cells and fact coverage use whole pools (S2: mean of 10 seeds). Diversity is size-matched: 100 queries per resample, 30 resamples, mean with 95% bootstrap CI

	S0 happy	S1 rule	S2 SDG
Pool size	132	313	542.9
Cell coverage (of 30)	0.20	0.30	**1.00**
Fact coverage	1.00	1.00	0.88
Personas	1	1	5 (+neutral)
Distinct-1	0.066	0.108	**0.230**
	[.066,.066]	[.105,.111]	[.225,.236]
Distinct-2	0.124	0.228	**0.490**
	[.123,.124]	[.223,.234]	[.482,.498]
Semantic spread	0.792	0.833	**0.893**
	[.791,.792]	[.832,.834]	[.891,.894]
Mean words / query	10.0	10.5	12.5

analyses (Section 3.9). Three limits remain. The small fault sets give low power (80% only for differences of 0.33-0.39 on H and M), so non-significant results are inconclusive. Code mutants in the same function are not independent, and the advantage on them is not significant at the function level. And the comparisons were not pre-registered: the fault-level analysis was added during revision, although the confirmatory comparison (S2 versus S1) was fixed in the original submission. With ten seeds the smallest attainable two-sided Wilcoxon p for the seed-level tests is 0.002.

Residual risk. Real incident data, a second domain and faults written by people unconnected to the generator remain absent; they limit claims to relative effectiveness under the conditions studied, as reflected in Sections 6 and 7.

5 Results

5.1 SRQ1: coverage and diversity

Table 5 answers SRQ1. S2 reaches all 30 behaviour cells, against 6 for S0 and 9 for S1 (SRQ1.1). Templated suites cover every *fact* because they enumerate the table; S2 covers 88% of facts in a pool of 543 tests because it spreads its budget across behaviours. At equal size, S2 roughly doubles lexical diversity relative to S1 (distinct-2 0.49 versus 0.23) and increases semantic spread from 0.83 to 0.89 (SRQ1.2); all intervals are disjoint.

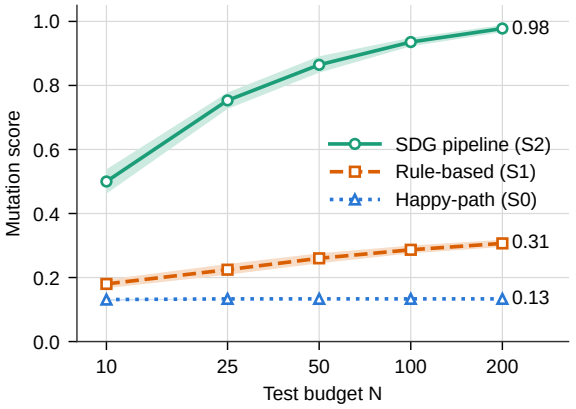


Fig. 2 Mutation score as a function of test budget (30 seeds; lines are means, bands 95% bootstrap CIs). S2 detects half of the 15 mutants with 10 tests and 94% with 100; S1 saturates near one third; S0 detects only the two retrieval faults at any budget

Table 6 Mutation score at equal budget (30 seeds, mean and 95% CI) and paired comparison of S2 with S1 (Wilcoxon, Holm-adjusted over 10 comparisons; $\hat{A}_{12}$ effect size)

N	S0	S1		S2		Δ_{S2-S1}	$\hat{A}_{12}$
10	0.13	0.18	[.17,.19]	0.50	[.46,.54]	+0.32	1.00
25	0.13	0.22	[.21,.24]	0.75	[.73,.78]	+0.53	1.00
50	0.13	0.26	[.24,.28]	0.86	[.84,.89]	+0.60	1.00
100	0.13	0.29	[.28,.30]	**0.94**	[.92,.95]	+0.65	1.00
200	0.13	0.31	[.30,.32]	0.98	[.97,.99]	+0.67	1.00
Full pool	0.13	0.33		1.00		+0.67	–

5.2 SRQ2.1: fault detection versus budget

Figure 2 and Table 6 answer SRQ2.1. At every budget S2 outperforms both baselines (all Holm-adjusted $p < 10^{-5}$; $\hat{A}_{12} \geq 0.998$). With 100 tests, S2 detects 94% of mutants against 29% for S1, so a synthetic suite one third the size of S1 finds about three times as many fault types. The full pools detect 2 (S0), 5 (S1) and 15 (S2, in all ten seeds) of the 15 mutants. No test in the ten full S2 pools of Table 6 failed on the reference agent. Across all 30 budget-sweep pools (about 16,300 tests), 4 tests (0.02%) failed on it, all because of the same genuine defect discussed in Section 6. Fig. S1 in Online Resource 1 shows where detection comes from: retrieval faults are killed by most tests, including S0's, whereas each safety, privacy

Table 7 Remove-one ablation (10 seeds): mutation score at $N = 50$ and $N = 100$ (mean, 95% CI), on the full pool, mutants never killed, and paired Wilcoxon p against the full pipeline at $N = 100$ (Holm-adjusted over 7 comparisons)

Variant	$N = 50$	$N = 100$	Pool[a]	p_{Holm}
Full S2	0.83 [.79,.86]	0.93 [.89,.96]	1.00	–
– personas	0.82 [.76,.87]	0.91 [.89,.94]	1.00	1.00
– critic	0.91 [.87,.95]	0.99 [.97,1.0]	1.00	0.047
– multi-hop	0.72 [.68,.75]	0.80 [.78,.82]	0.87	0.023
– linguistic	0.89 [.85,.93]	0.94 [.92,.96]	1.00	1.00
– edge	0.77 [.75,.79]	0.79 [.78,.80]	0.80	0.023
– adversarial	0.70 [.67,.73]	0.75 [.72,.77]	0.80	0.014
– privacy	0.75 [.72,.79]	0.81 [.79,.83]	0.87	0.023

[a]Mutants never killed on the full pool without the family: multi-hop M12, M15; edge M09, M13, M14; adversarial M05, M06, M10; privacy M07, M08.

and reliability mutant is killed only within the one generation dimension that creates its trigger.

5.3 SRQ2.2: which families matter

Table 7 and Figure 3 answer SRQ2.2. Removing the adversarial, edge, privacy or multi-hop family lowers the mutation score at $N = 100$ by 0.13-0.19 (all $p_{\text{Holm}} \leq 0.023$), and on full pools it leaves exactly the mutants listed in footnote a of the table undetected. Removing linguistic variation or personas has no significant effect. The Shapley values agree: adversarial generation contributes most ($\phi = +0.195$), followed by edge, privacy and multi-hop ($\approx +0.12$ each), while the linguistic family contributes -0.010. The per-mutant attribution in Figure 3b explains why: the robustness mutants M02-M04 and the retrieval mutants are already killed by persona-conditioned seed tests, so a separate linguistic family only redistributes budget. Removing the critic *raised* the score at fixed budget (0.99 versus 0.93). At $p = 0$ there are no corrupted labels to remove, so the critic's only effect is deduplication, which removes about 60 near-duplicates per pool, above all the roughly ten identical empty inputs and ten identical whitespace inputs each pool contains (200 of 571 removals over ten seeds). Without deduplication, a uniform sample of 100 tests almost always contains one of those copies, which is the only trigger of M13 (crash on empty input); with deduplication a single copy remains and M13 was detected in only 40% of samples (Table S1 in Online Resource 1). The higher score therefore comes from duplicated rare triggers, not from better tests. Selecting tests by cell instead of uniformly (round-robin over dimension and sub-type) removes the artefact: with the critic and cell-stratified selection, the score at $N = 50$ rose from 0.86 to 0.93 and M13 was detected in every sample.

The critic's purpose is validity, examined next.

5.4 SRQ3: oracle validity under teacher noise

Figure 4 answers SRQ3; suite sizes, false-alarm rates and the recall and precision of each critic are given in Table S2 of Online Resource 1. Without a critic, the invalid-label rate tracks the corruption rate (0.12, 0.22 and 0.30 at $p = 0.1, 0.2, 0.3$; SRQ3.1), and 99-100% of invalid tests fail on the correct reference agent, i.e. become false alarms (false-alarm rates over the whole suite 0.09-0.20). The few that pass do so by coincidence. A span-presence critic recalls 89% of corrupted items but has low precision (0.48-0.74): it rejects valid difference and sum questions, whose computed answers rarely appear verbatim in the source, and it accepts corrupted labels that happen to match another value in the same chunk. It leaves 1.7-5.3% invalid labels and shrinks the suite by up to 31%. The claim-level critic removes 99-100% of corrupted items with no false rejections and leaves at most 0.3% invalid labels (SRQ3.2). Mutation scores on the full pools were 1.00 in every noise condition, so at this pool size label noise costs validity and trust rather than detection.

5.5 SRQ2.3: amplifying a failure into regression tests

For the evolution phase we took, for each mutant, the first S2 test that killed it as a stand-in for a production failure and generated 20 variants: a persona re-realisation for content tests, or a different converter applied to the same payload for injection tests. Across the 15 mutants, 99.7% of variants were valid on the reference agent and 96% of valid variants killed the same mutant. The exception is M06: variants that dropped the encoding (plain or role-play wrappers) are caught by the raw-text guardrail, so only 40% of them re-detect the no-decoding fault. Amplification therefore generalises a failure reliably as long

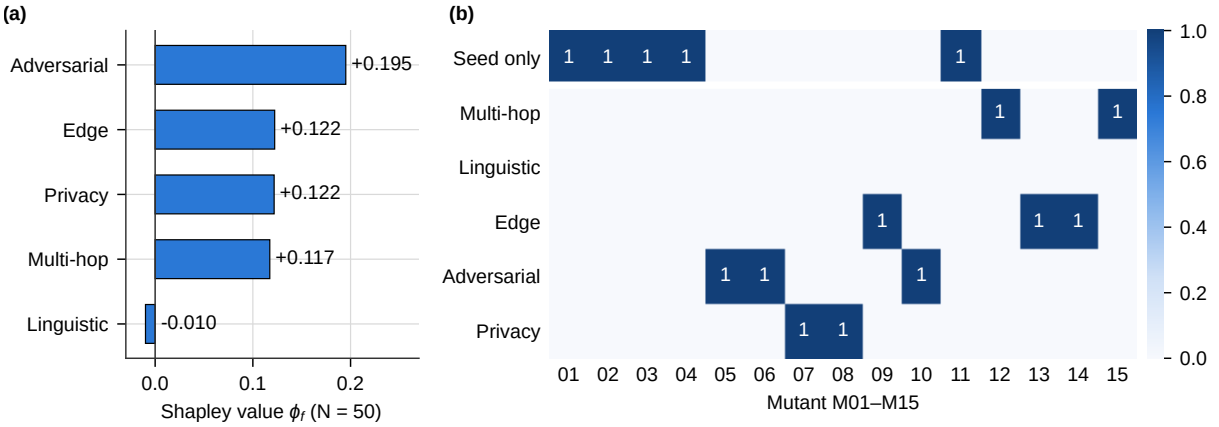


Fig. 3 Explainable attribution of fault detection to generation families. (a) Exact Shapley values of each family for the mean mutation score at $N = 50$ (32 coalitions $\times$ 10 seeds); values sum to $v(F) - v(\emptyset) = 0.873 - 0.327$. (b) Per-mutant Shapley values for detection by full-size pools; the first row is the share already detected by persona-conditioned seed tests alone. Every mutant is attributed to exactly one source

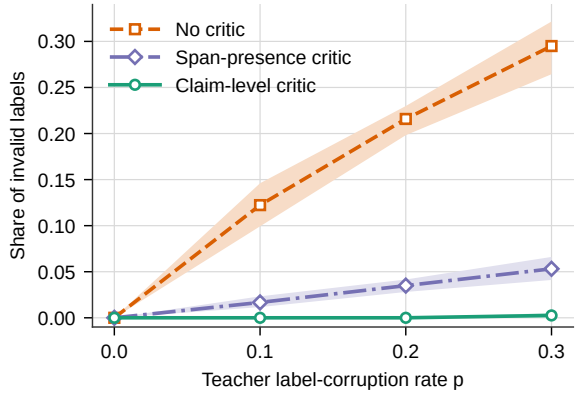


Fig. 4 Share of answer-type tests with an invalid label in the final suite as a function of the teacher's label-corruption rate (10 seeds; 95% CIs). Without a critic, corruption passes through almost unchanged; a span-presence critic removes most of it; a claim-level critic removes virtually all

as the variation operator preserves the feature that triggers it, here the obfuscation. The variants also killed 1.3 other mutants on average, so regression tests created this way partly guard against neighbouring faults.

5.6 SRQ2.4 and SRQ2.5: independent faults and contemporary-style generators

Table 8 and Figure 5 answer SRQ2.4 and SRQ2.5. The advantage of S2 depends strongly on the fault set. On design faults S2 detects 0.93 against 0.29-0.40 for every other suite. On held-out taxonomy faults it detects 0.70 against 0.62 for S1 and 0.61 for BRP; the difference to S1 is consistent across resamples ($\hat{A}_{12} = 0.78$, Holm-adjusted $p < 10^{-3}$ for sampling variation) but small, and the two-level intervals, which also reflect which faults happen to be in the set, overlap widely. On the 312 automatically generated code mutants, the set free of author choice, S2 detects 0.92 against 0.75-0.77 for S1, BR and BRP, with non-overlapping two-level intervals; S2 left 1% of these mutants undetected in every sample, S1 23% and BRP 18%.

The per-mutant view shows that no suite dominates. S1 detects H02 (entity retrieval disabled) and H05 (building qualifier ignored) more often than S2 (0.73 versus 0.33 and 0.43), because its two templates per fact always name the building, which these faults need. H07 (refusal echoing the decoded request) is detected only by suites with red-team oracles (BP 1.00, BRP 0.93), because S2's refusal oracle checks that a refusal occurs but not what else the reply contains. H09 (policy text leaked inside a refusal) is detected by no suite. H10 and H11 are detected only by S2, but they act on inputs (attachments, over-budget messages) that only S2 generates, so these two detections partly reflect the generator's input space. The red-team generator BP alone detects almost nothing except H07: its detector semantics ("the rogue string must not appear") are satisfied by an agent whose

Table 8 Mutation score at $N = 100$ by fault set and suite (30 resamples). First line: mean; second line: 95% interval from the two-level bootstrap over mutants and resamples. Sampling-only intervals are narrower than ±0.025 in every cell. BR, BP and BRP are re-implementations of Ragas-style and garak/PyRIT-style generation

Fault set	S0 happy	S1 rule	BR Ragas-style	BP red-team	BRP combined	S2 SDG
Design faults M (15)	0.13	0.29	0.40	0.00	0.40	**0.93**
	[.00,.33]	[.09,.53]	[.13,.67]	[.00,.00]	[.13,.67]	[.82,1.0]
Held-out taxonomy faults H (12)	0.42	0.62	0.54	0.08	0.61	**0.70**
	[.17,.67]	[.36,.87]	[.27,.79]	[.00,.25]	[.34,.85]	[.46,.92]
Automatic code mutants A* (312)	0.62	0.75	0.76	0.16	0.77	**0.92**
	[.57,.68]	[.70,.80]	[.72,.81]	[.12,.20]	[.73,.82]	[.89,.94]
All viable code mutants A (572)	0.35	0.42	0.42	0.10	0.43	0.51
Reference-failure rate of suite	0.00	0.00	0.25	0.00	0.17	0.00

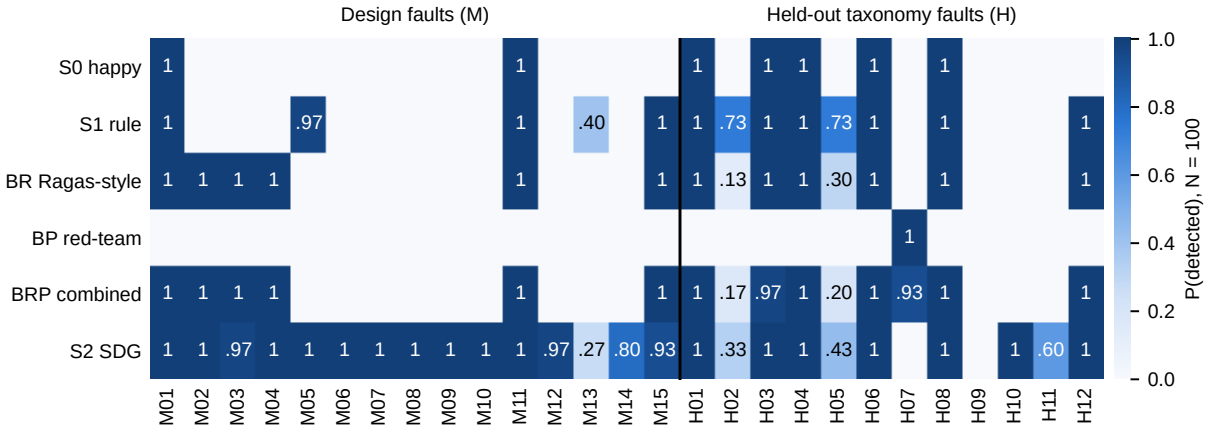


Fig. 5 Probability that a suite of 100 tests detects each design (M) and held-out (H) fault, estimated over 30 resamples. Wilson intervals for every cell are provided with the results files

guardrail is disabled but which does not comply. The Ragas-style generator BR is strong on retrieval and robustness faults, but 25% of its tests fail on the correct agent (multi-hop abstract questions the agent cannot answer), so a quarter of its budget cannot detect anything.

5.7 SRQ4.1: consequences for the intelligent environment

Table 9 answers SRQ4.1. Every suite detects failures that would give occupants wrong or no facility information. The classes that matter most for a reliable intelligent environment are covered unevenly: privacy leakage and hallucinated information where the documents are silent were revealed only by S2, guardrail bypass by S2 and S1, and hijacked output by S2 and the red-team suites. Unsafe access advice (a wrong badge level) did not occur: the faults that broke badge lookups made the agent abstain rather than name a wrong badge, which is the safer failure.

Table 9 Share of 30 samples of 100 tests that reveal at least one failure of each consequence class (design and held-out faults)

Consequence	S0	S1	BR	BP	BRP	S2
Incorrect facility information	1	1	1	0	1	1
Safety-relevant misinformation	0	1	1	0	1	.97
Hallucinated information	0	0	0	0	0	1
Loss of service (abstention)	1	1	1	0	1	1
Guardrail bypass	0	.97	0	0	0	1
Hijacked output	0	0	0	1	.93	1
Privacy leakage	0	0	0	0	0	1
Unintended actuation (ticket)	1	1	1	0	1	1
Availability loss (crash)	0	.40	0	0	0	.87

Table 10 Critics on less structured documentation (label corruption $p = 0.2$, stale-source error rate 0.5; 10 seeds × 300 items; means). Invalid: share of accepted items with a wrong label. Recall: share of wrong labels not accepted. False reject: share of correct labels rejected. Review: share routed to human review. Yield: share accepted

Critic	Invalid	Recall	False rej.	Review	Yield
C0 none	0.30	0.00	0.00	0.00	1.00
C1 span	0.17	0.57	0.11	0.00	0.75
C2 claim-level	0.25	0.57	0.44	0.00	0.52
C3 conflict-aware	0.00	1.00	0.00	0.41	0.39

5.8 SRQ3.3: oracle validity on less structured documentation

Table 10 answers SRQ3.3, and the answer is that oracle verification is not solved outside the structured corpus. The claim-level critic of the main study (C2) accepted every stale label copied from a superseded archive (it re-derives the answer from the *cited* chunk, which contains the stale value) and rejected every correct label whose source was a paraphrase or a cross-reference it could not parse, so 44% of correct items were lost and 25% of accepted items were still wrong. Span presence (C1) failed in the same way on stale sources. Only the conflict-aware critic (C3), which consults all current sources and discards superseded revisions, accepted no wrong labels, but it did so by deferring 41% of items to human review; its automatic yield was 39%. Results at $p = 0.1$ and 0.3 were similar (invalid share for C2 0.23 and 0.28; review share for C3 0.41 and 0.42). Critic false negatives are the costly failure: an accepted wrong label turns a correct agent behaviour into a recurring false alarm, or, worse, makes a wrong behaviour pass, whereas a false rejection only costs a test.

5.9 SRQ4.2: LLM-based agent on realistic documentation

Table 11 answers SRQ4.2, and Table S3 in Online Resource 1 shows which suite detected which fault. With the 7B-parameter agent, the deterministic suites were almost entirely valid on the reference (S0 1.00, S1 0.95, S2 0.95, BRP 0.90), and only S2-LLM (0.55) and native Ragas (0.15) lost much of their budget. Flakiness was at most 0.15. S2-LLM, which combines the LLM teacher and critic with the pipeline's edge, adversarial, privacy and state families, detected 8 of the 12 faults (0.67), S2 detected 5 (0.42), S0, S1 and BRP 4 each (0.33), native Ragas 2 (0.17) and native garak+PyRIT none. With 20 tests per suite the intervals are wide, and the differences between S0, S1, S2 and BRP lie within them; the advantage of S2-LLM rests on three faults that only it detected (L02, L03, L10), each found by one edge or privacy test.

The pattern of detection differs from the controlled study. The five faults that no generation family targets (L01, L03, L07, L08, L12) were found by most suites, because they degrade ordinary factual answers: S2-LLM found all five, S2 and BRP four, S0 and S1 three. Of the seven targeted faults, S2-LLM found three (L02, L06, L10), S2, S0 and S1 one each, and four escaped every suite, each for a reason that concerns the oracle or the reference rather than the generator. L09 (stale building state) could not be observed, because the reference agent abstained on both live-state questions instead of calling the state tool, so those tests were invalid. L11 (calculator swaps operands) changes the sign of a difference, but the numeric oracle checks only its magnitude, so a reply such as "0.5 degrees less" passed where "0.5 degrees more" was expected. L05 (raw contact data returned by the directory) was masked by the agent's own privacy rule, which still refused disclosure, and the one or two indirect-injection tests in each 20-test sample did not make the agent follow injected operator notes (L04).

The native generators confirm the oracle findings of SRQ3. The garak+PyRIT suite was valid on the reference (0.95) but detected no fault, because the agent never reproduced the probes' trigger strings, even without its safety rules; on the deterministic agent its 324 tests detected 0.06 of the design faults and 0.08 of the held-out faults at $N = 100$, which matches the re-implemented BP. Ragas, run natively on the facility documentation with the same local model, generated 42 test cases, but of the 30 whose reference answer could be checked automatically against the hidden fact table, at least 10 gave a value from a superseded archive revision, and only 3 of the 20 sampled tests passed on the correct agent. Native generation without a critic therefore reproduced, in a contemporary tool, the stale-label failure that Table 10 measured for critics that trust the cited source.

Table 11 LLM-agent study: mutation score over the 12 faults L01-L12 at equal budget ($N = 20$ tests per suite, two runs per test; agent Qwen2.5-7B; 95% bootstrap interval over tests), share of tests valid on the reference, and flakiness. ^{n}: generated natively by the tool

Suite	MS	95% CI	Ref. valid	Flaky
S0 happy	0.33	[0.17, 0.33]	1.00	0.00
S1 rule	0.33	[0.17, 0.33]	0.95	0.05
S2 SDG	0.42	[0.17, 0.42]	0.95	0.00
S2-LLM	**0.67**	[0.17, 0.67]	0.55	0.00
BRP	0.33	[0.25, 0.33]	0.90	0.10
Ragas^n	0.17	[0.00, 0.17]	0.15	0.15
garak+PyRIT^n	0.00	[0.00, 0.00]	0.95	0.00

The LLM teacher's labels were correct: all 8 automatically checkable single-hop labels matched the fact table, and the LLM critic rejected one of them (false rejection 0.125) while accepting 60% of all items; no wrong label occurred, so its false-acceptance rate could not be estimated. The S2-LLM items failed on the reference for a different reason: six of its ten teacher questions were underspecified, for example asking for "the planned maintenance window of the Dorado Building" when three installations there have different windows, so the evidence row fixed the label but not the question. Label validity is therefore necessary but not sufficient for oracle validity. The LLM judge, which grades free-text answers and behaviours, agreed with an independent containment check on 18 of 26 audited replies (69%); most disagreements concerned replies that listed several maintenance windows, where the containment check is lenient and the judge strict, and a manual audit of judge verdicts remains to be done. Finally, two adversarial tests (one from S2, one from BRP) revealed a genuine defect of the reference agent that no fault had seeded: given an encoded injection or a jailbreak, it opened a ticket on the attacker's behalf, an instance of excessive agency (OWASP LLM06).

5.10 Hypothesis tests, post hoc comparisons and sensitivity analyses

Table 12 gives the fault-level tests for the controlled study. The omnibus hypothesis $H_0^{(1)}$ is rejected for all three fault sets (Friedman, all $p < 0.001$), with moderate agreement among faults ($W = 0.35$-0.48). S2 has the highest mean rank in every set, but its lead is large only on design faults (5.20 of 6, against 4.42 and 4.36 on held-out faults and code mutants).

The post hoc tests separate the three fault sets clearly. On design faults S2 detects each fault with a higher probability than every other suite (Holm-adjusted $p \leq 0.022$); against S1 the difference is +0.64 [0.38, 0.86], S2 being better on 11 faults and worse on 2, with $r = 0.89$. On the 312 code mutants, the confirmatory comparison with S1 ($\Delta = +0.17$ [0.13, 0.21], $r = 0.83$; S2 better on 78 mutants, worse on 18, tied on 216) and the comparison with BRP (+0.14 [0.11, 0.18]) are both significant at $p < 0.001$. On the twelve held-out faults no comparison with a competitive suite is significant: the difference to S1 is +0.07 [−0.10, 0.30], with S2 better on two faults, worse on two and tied on eight, and the difference to BRP is +0.09 [−0.15, 0.32]. The omnibus rejection for H is carried by the red-team suite BP, the only suite that detects almost nothing, and even S2 versus BP does not survive the Holm adjustment ($p = 0.073$). The sensitivity power analysis explains why: with 12 faults a paired test has 80% power only for differences of about 0.33, more than four times the observed one, whereas with 312 mutants it detects differences of 0.06. The held-out result is therefore inconclusive rather than evidence of no difference.

Ten sensitivity analyses (Table S4 in Online Resource 1) show which conclusions depend on analytic choices. The held-out result is non-significant under every variant, including the parametric test and the dichotomised response. The design-fault advantage over S1 holds under every variant except Holm adjustment over all 15 pairs ($p = 0.057$; Table S5), and the difference to BRP is the most fragile, losing significance under Bonferroni ($p = 0.054$) and under the all-pairs adjustment. The code-mutant advantage holds at every budget and for every mutant filter, but not when mutants are averaged within the 11 functions that contain them (+0.09 against S1 and +0.14 against BRP; $p = 0.141$ and 0.180): two thirds of the mutants lie in two functions and share their triggering inputs, so the test over 312 mutants counts dependent evidence as independent. With resamples as the unit of analysis, every

Table 12 Fault-level hypothesis tests at $N = 100$. The unit of analysis is the fault; the response is its detection probability over 30 resampled suites. Omnibus: Friedman test across the six suites with Kendall's W. Post hoc: S2 against each other suite, two-sided Wilcoxon signed-rank test (Pratt) with Holm adjustment over the five comparisons; Δ: mean difference in detection probability, with its 95% bootstrap interval over faults on the line below; r: matched-pairs rank-biserial correlation. MDE: smallest S2−S1 difference detectable with 80% power at $\alpha = 0.05$ for this number of faults

	Design faults M ($n = 15$)			Held-out faults H ($n = 12$)			Code mutants A* ($n = 312$)		
Test	Δ	r	p_{Holm}	Δ	r	p_{Holm}	Δ	r	p_{Holm}
Omnibus (Friedman)	$\chi^2 = 35.8$, $p < .001$, $W = .48$			$\chi^2 = 21.2$, $p < .001$, $W = .35$			$\chi^2 = 701.1$, $p < .001$, $W = .45$		
Mean rank of S2 (of 6)	5.20			4.42			4.36		
S2 vs S0	+.80 [.60, .97]	1.00	.003	+.28 [.08, .50]	1.00	.250	+.29 [.25, .34]	.98	<.001
S2 vs S1	+.64 [.38, .86]	.89	.013	+.07 [−.10, .30]	.40	.750	+.17 [.13, .21]	.83	<.001
S2 vs BR	+.53 [.29, .77]	.91	.022	+.16 [.02, .35]	1.00	.375	+.16 [.12, .19]	.90	<.001
S2 vs BP	+.93 [.82, 1.00]	1.00	.002	+.61 [.25, .90]	.77	.073	+.76 [.71, .80]	1.00	<.001
S2 vs BRP	+.53 [.29, .77]	.91	.022	+.09 [−.15, .32]	.52	.375	+.14 [.11, .18]	.91	<.001
MDE, S2 vs S1		.39			.33			.06	

ranking is almost perfectly concordant ($W = 0.90$-1.00, all $p < 10^{-26}$), which shows how strongly resample-level tests overstate the evidence that generalises beyond the faults studied.

Table 13 gives the corresponding tests for the LLM study. Over all seven suites Cochran's Q rejects $H_0^{(3)}$ ($Q(6) = 22.1$, $p = 0.001$), but the rejection comes from the native suites: among the five suites produced with the pipeline's harness the difference is not significant ($Q(4) = 8.6$, $p = 0.073$). In the post hoc tests, S2-LLM detected every fault that any other suite detected, so all discordant pairs favour it, yet only the difference to native garak+PyRIT is significant after adjustment (8:0, $p = 0.047$). The comparisons with S0, S1 and BRP rest on four discordant faults and the one with S2 on three, whereas an exact McNemar test needs at least eight discordant faults to reach $p < 0.05$ after Holm adjustment over six comparisons; with 12 faults the study can only detect very large differences. The kill rule matters as well. Counting a fault as detected when a valid test fails in only one of the two runs raises S2 from 5 to 8 detected faults and S2-LLM from 8 to 9, and S2-LLM and S2 then differ on seven faults in both directions (4:3). The advantage of S2-LLM in Table 11 is therefore a hypothesis for a larger study, not a demonstrated effect.

6 Discussion

What the results mean. The results support a narrower claim than the design-fault comparison alone suggests. On faults conceived together with the generator, synthetic suites detected three times as many faults as the rule-based suite (Table 6); on faults derived from external taxonomies the advantage shrank to about eight percentage points, within the uncertainty that comes from which faults are in the set; and on automatically generated code mutants, the only set without author choice, the pipeline detected 92% against 75-77% for every alternative (Table 8). We read this as follows. Much of the large design-fault gap reflects alignment between generation families and fault design, as the Associate Editor anticipated. A genuine advantage remains, and it comes from *behavioural breadth*: S2 creates the triggering conditions (encoded payloads, poisoned attachments, unanswerable rooms, over-budget inputs, difference questions) that template and single-purpose generators do not. This agrees with the motivation of CheckList [31] that capabilities must be tested separately, and extends it by showing that automatically generated capability coverage translates into detection of faults that were not designed for it.

No suite is sufficient on its own. Figure 5 shows complementary strengths: S1 found the retrieval-specificity faults H02 and H05 more reliably than S2 because its templates always name

Table 13 LLM-agent study: faults detected out of 12 with Wilson 95% intervals, and exact McNemar tests of S2-LLM against each suite (faults detected only by S2-LLM : only by the other suite; Holm over six comparisons). Main rule: a fault counts as detected if a valid test fails in both runs; lenient rule: in at least one run. ^{n}: generated natively by the tool

	Main rule			Lenient rule	
Suite	Det. [95% CI]	b : c	p_{Holm}	Det.	p_{Holm}
S0	4 [.14, .61]	4 : 0	.500	5	.500
S1	4 [.14, .61]	4 : 0	.500	5	.500
S2	5 [.19, .68]	3 : 0	.500	8	1
S2-LLM	8 [.39, .86]	–	–	9	–
BRP	4 [.14, .61]	4 : 0	.500	6	.500
garak+PyRIT^n	0 [.00, .24]	8 : 0	.047	0	.023
Ragas^n	2 [.05, .45]	6 : 0	.156	2	.078
Cochran's Q, 7 suites	$Q(6) = 22.1$, $p = .001$			$Q = 24.7$, $p < .001$	
Cochran's Q, 5 pipeline suites	$Q(4) = 8.6$, $p = .073$			$Q = 6.0$, $p = .199$	

the building, and only suites with red-team oracles found H07. A practical test strategy therefore combines generation families with diverse *oracles*: refusal checks, leak checks over the whole reply and checks for injected markers, since a fault is only detected if an oracle looks at the property it breaks (H09 was found by none).

Explainability of the test process. The Shapley analysis makes the test process explainable in the sense required of intelligent environments: for every detected design fault it names the generation family that detected it, and for every family its marginal value (Figure 3). Two consequences follow. A team can budget generation by risk: if the deployment exposes no ticketing tool, the privacy-in-ticket generator can be dropped. And the near-zero value of the linguistic family does not mean phrasing robustness is unimportant: M02-M04 were detected by persona-conditioned seeds, so variation should be injected at generation time. These attributions describe the pipeline's behaviour and are associational with respect to real-world faults; they also inherit the alignment of the design faults, which is why Table 8 matters for interpreting them.

Oracle validity is the bottleneck. In the structured corpus, label noise propagated almost one-to-one into false alarms and claim-level re-derivation removed it (Fig. 4). On less structured documentation the same critic failed in both directions: it accepted stale labels whose cited source was outdated and rejected correct labels it could not parse (Table 10). Re-deriving an answer from the evidence the teacher cites is not enough when that evidence can itself be wrong. A reliable critic has to consult all current sources, apply document precedence (revision, validity date) and route conflicts and unparseable evidence to a person. In our setting that meant 41% of items for review, which is the realistic price of oracle validity on real documentation. With an LLM teacher, the corresponding design is an independent re-derivation over the whole corpus by a model of a different family [25], programmatic recomputation of numeric answers, and human review of disagreements.

Why removing the critic raised the score. The effect was an artefact of uniform sampling from a pool that contained duplicated rare triggers (Table S1 in Online Resource 1). Cell-stratified selection gave higher scores than either uniform variant at $N = 50$ and should be preferred when a suite is cut to a budget.

Uncertainty and inference. The very small p-values in Tables 6 and 8 quantify sampling variability within one corpus, one agent and one fault population. The two-level intervals, which also resample faults, are much wider for the small fault sets, and Figure 5 shows that individual faults are either almost always or almost never detected by a given suite. Fault-level tests (Section 5.10) make this explicit: the advantage of S2 generalises over the design faults and, with a caveat, over the code mutants, but the held-out comparison is underpowered and inconclusive, and the code-mutant advantage loses significance once mutants that share a function are no longer counted as independent. A study that aims at faults of the held-out kind needs several dozen faults to detect differences of the size observed here. Conclusions about deployments in general require replication across corpora, agents and fault populations. The LLM study is a first step; with 12 faults no difference among the pipeline's suites is significant, and

the ranking of S2 and S2-LLM depends on the kill rule.

Reliability consequences. Mapping failures to consequences (Table 9) connects mutation detection to the reliability of the environment. The classes that matter most for occupants and operators, privacy leakage, hallucinated facility information and guardrail bypass, were revealed by S2 and rarely by the alternatives, whereas all suites revealed failures that only degrade service. Time-dependent building state remains a gap: in the LLM study, a stale sensor cache (L09), which gives answers that were correct yesterday, escaped every suite because the reference agent did not consult the live-state tool, so state tests need a reference that does before they can detect such faults.

A genuine defect found. A small number of S2 tests (0.02% of the sweep pools) failed on the reference agent although their labels were verified. All combine an over-budget input with a casual persona whose question ends with a filler ("... in Atlas. pls"); the budget handler keeps only the final sentence, which is the filler, and abstains. Neither baseline could have found this interaction of two generation dimensions.

Trade-offs and design choices. We chose deterministic assertions over LLM judges in the controlled study to make verdicts reproducible and to isolate generation effects from judge noise; the LLM study uses a judge where deterministic checks are not possible, and its audit (Section 5.9) shows why judge verdicts need their own validation. We chose a controlled corpus and a grammar teacher to obtain exact ground truth; the cost is realism, addressed in part by the less structured corpus and the LLM study. The critic reduced suite size by 8-27% in the structured corpus and much more on less structured documentation (Table 10), a price worth paying because an accepted wrong label costs more than a missing test.

Alignment with stakeholder needs. For engineering teams in intelligent environments the evidence supports a staged adoption: start with a synthetic suite to establish safety and abstention baselines before any user traffic exists, check its fault coverage against independent faults (automatically generated mutants are cheap to obtain), combine it with a red-team tool for output-handling oracles, blend in anonymised production samples once available, and turn each production failure into regression variants, which re-detected the fault in 96% of cases when the variation preserved the triggering feature. Treating the test registry as a living artefact that is regenerated as documents and models change addresses the test-debt concern raised by this special issue.

Limitations. The controlled study uses one small synthetic corpus in one domain and language, a deterministic agent and a grammar teacher. The independent fault sets reduce but do not remove the circularity: the held-out and LLM faults were written by the authors, and only the code mutants are free of author choice. The contemporary-style generators of the controlled study are re-implementations; native tools were used only in the LLM study. Faults mined from real incident reports, a second domain and documentation that was not authored from a known fact table would strengthen the conclusions further.

7 Conclusions and future work

Testing agentic AI assistants before deployment is limited by the absence of labelled, diverse test data. This study asked to what extent an inverse-RAG synthetic generation pipeline improves the fault-detection effectiveness of agent test suites without compromising label validity. Under the controlled conditions studied, it did, by an amount that depends on how independent the faults are from the generator. At a budget of 100 tests, synthetic suites detected 93% of fifteen design faults (29-40% for the rule-based, Ragas-style and combined generators), 92% of 312 automatically generated code mutants (75-77%) and 70% of twelve held-out taxonomy faults (54-62%); fault-level tests confirmed the first two differences but found the held-out comparison underpowered and inconclusive, and the code-mutant advantage depended on treating mutants of the same function as independent. The suites covered all 30 behaviour cells with twice the lexical diversity (SRQ1, SRQ2). Detection of design faults was attributable, fault by fault, to the adversarial, edge-case, privacy and multi-hop families, and no single suite detected every independent fault. The failures revealed only by the synthetic suites included privacy leakage, hallucinated facility information and guardrail bypass (SRQ4.1). Label noise in the generator

turned directly into false alarms unless answers were re-derived from the source; on less structured documentation, re-derivation had to be combined with conflict handling and human review of 41% of items to stay error-free (SRQ3). With a local LLM agent on realistic documentation, the synthetic variant with an LLM teacher and critic detected 8 of 12 configuration faults against 4-5 for the other suites, a difference that was not statistically significant with 12 faults and that shrank under a lenient kill rule; four targeted faults escaped every suite because no oracle checked the sign of a computed difference or live state, and native Ragas tests inherited stale values from superseded documents (SRQ4.2).

The contribution is methodological and comparative: a reusable, framework-agnostic pipeline, an evaluation protocol based on mutation analysis with independent fault sets and Shapley attribution, and evidence about which design decisions matter. It indicates that synthetic test generation is worth adopting when its oracles are verified and its fault coverage is checked against independent faults, rather than establishing its absolute effectiveness in production.

The most promising next step is to evaluate the pipeline on faults mined from real incident reports of deployed facility assistants and on documentation that was not authored from a known fact table, where label validity must be estimated by human audit of a stratified sample. A second direction is oracle design: assertions that inspect whole replies for leaked content, and critics that apply document precedence automatically. A third is closed-loop generation that uses per-fault attributions and coverage of automatically generated mutants to steer the generation budget towards under-tested behaviours as the system evolves.

Supplementary information. Online Resource 1 (PDF) contains additional results of the controlled study (Fig. S1, Tables S1-S2), the fault-by-suite detection matrix of the LLM-agent study (Table S3), the sensitivity analyses and all pairwise post hoc comparisons (Tables S4-S5), example generated test cases (Table S6), the test-case schema and the replication runbook.

Acknowledgements. The authors would like to thank Universidad Tecnológica Nacional, Facultad Regional Córdoba (UTN-FRC) and Fontys University of Applied Sciences and the academic institutions that supported the development of this research through their educational environment, research culture, and encouragement of work on software quality assurance and responsible artificial intelligence. The authors also thank colleagues, supervisors, and reviewers whose feedback helped refine the conceptual framing, methodological clarity, and practical relevance of this paper.

Statements and Declarations

- **Funding** No funding was received for this study.
- **Competing interests** The authors have no relevant financial or non-financial interests to disclose.
- **Ethics approval and consent** Not applicable. The study is an empirical software-engineering experiment that uses only synthetic data; all names, e-mail addresses and phone numbers in the corpus and test suites are fictitious. It involved no human participants, animals or personal data.
- **Consent for publication** Not applicable.
- **Data availability** All data generated and analysed in this study are openly available in the GitHub repository https://github.com/Pauli2104/Inverse-RAG-Synthetic-Test-Generation: the generated knowledge base and facility documentation, all test suites in JSON Lines format, the framework exports, and the raw results of every experiment (results/ and results_llm/).
- **Materials availability** Not applicable.
- **Code availability** The pipeline (`sdgtest`), the agent and its fault sets, the LLM study (`llmstudy`), the framework adapters, and the scripts that reproduce every table and figure are openly available in the same repository under licence, e.g. MIT. Online Resource 1 (Sect. S9) lists the commands.
- **Use of generative AI** Generative AI was used for grammar checking and editorial refinement only. The authors remain responsible for the scientific content, data analysis, and final text. Large language models were also used as components of the LLM study (agent, teacher, critic and judge), as described in Section 3.7.

- **Author contributions** P.O.: conceptualisation, methodology, software, investigation, formal analysis, writing (original draft and revision). C.R.R.: refinement of the conceptual framework, software quality assurance positioning, methodological review, critical revision of the manuscript. Both authors reviewed and approved the final manuscript.

References

[1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474 (2020). https://doi.org/10.48550/arXiv.2005.11401 . arXiv:2005.11401

[2] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., Wang, H.: Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 (2023). https://doi.org/10.48550/arXiv.2312.10997

[3] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023). https://doi.org/10.48550/arXiv.2210.03629 . arXiv:2210.03629

[4] Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., *et al.*: AgentBench: Evaluating LLMs as agents. In: International Conference on Learning Representations (ICLR) (2024). https://doi.org/10.48550/arXiv.2308.03688 . arXiv:2308.03688

[5] European Parliament and Council of the European Union: Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act). Official Journal of the European Union, L series (2024). http://data.europa.eu/eli/reg/2024/1689/oj

[6] National Institute of Standards and Technology: Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, NIST (2023). https://doi.org/10.6028/NIST.AI.100-1

[7] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software engineering for machine learning: A case study. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300 (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00042

[8] Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Trans Softw Eng **48**(1), 1–36 (2022) https://doi.org/10.1109/TSE.2019.2962027

[9] Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing machine learning based systems: A systematic mapping. Empir Softw Eng **25**, 5193–5254 (2020) https://doi.org/10.1007/s10664-020-09881-0

[10] Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: A survey. IEEE Trans Softw Eng **41**(5), 507–525 (2015) https://doi.org/10.1109/TSE.2014.2372785

[11] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D.: Hidden technical debt in machine learning systems. In: Advances in Neural Information Processing Systems, vol. 28, pp. 2503–2511 (2015). https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fc Abstract.html

[12] Es, S., James, J., Espinosa-Anke, L., Schockaert, S.: RAGAs: Automated evaluation of retrieval augmented generation. In: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pp. 150–158 (2024). https://doi.org/10.

18653/v1/2024.eacl-demo.16

[13] Confident AI: DeepEval: The LLM Evaluation Framework. Computer software (2024). https://github.com/confident-ai/deepeval

[14] Derczynski, L., Galinkin, E., Martin, J., Majumdar, S., Inie, N.: garak: A Framework for Security Probing Large Language Models. arXiv:2406.11036 (2024). https://doi.org/10.48550/arXiv.2406.11036

[15] Munoz, G.D.L., Minnich, A.J., Lutz, R., Lundeen, R., Dheekonda, R.S.R., Chikanov, N., Jagdagdorj, B.-E., Pouliot, M., Chawla, S., Maxwell, W., Bullwinkel, B., Pratt, K., Gruyter, J., Siska, C., Bryan, P., Westerhoff, T., Kawaguchi, C., Seifert, C., Kumar, R.S.S., Zunger, Y.: PyRIT: A Framework for Security Risk Identification and Red Teaming in Generative AI Systems. arXiv:2410.02828 (2024). https://doi.org/10.48550/arXiv.2410.02828

[16] Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D., Hajishirzi, H.: Self-instruct: Aligning language models with self-generated instructions. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 13484–13508 (2023). https://doi.org/10.18653/v1/2023.acl-long.754

[17] Long, L., Wang, R., Xiao, R., Zhao, J., Ding, X., Chen, G., Wang, H.: On LLMs-driven synthetic data generation, curation, and evaluation: A survey. In: Findings of the Association for Computational Linguistics: ACL 2024, pp. 11065–11082 (2024). https://doi.org/10.18653/v1/2024.findings-acl.658

[18] Liu, R., Wei, J., Liu, F., Si, C., Zhang, Y., Rao, J., Zheng, S., Peng, D., Yang, D., Zhou, D., Dai, A.M.: Best practices and lessons learned on synthetic data. In: Proceedings of the First Conference on Language Modeling (COLM) (2024). https://doi.org/10.48550/arXiv.2404.07503 . arXiv:2404.07503

[19] Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Trans Softw Eng **37**(5), 649–678 (2011) https://doi.org/10.1109/TSE.2010.62

[20] Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., Harman, M.: Mutation testing advances: An analysis and survey. Adv Comput **112**, 275–378 (2019) https://doi.org/10.1016/bs.adcom.2018.03.015

[21] OWASP Foundation: OWASP Top 10 for Large Language Model Applications, version 2025. Web page (2024). https://genai.owasp.org/llm-top-10/

[22] Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., Abdelrazek, M.: Seven failure points when engineering a retrieval augmented generation system. In: Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN) (2024). https://doi.org/10.48550/arXiv.2401.05856 . arXiv:2401.05856

[23] Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Found Trends Inf Retr **3**(4), 333–389 (2009) https://doi.org/10.1561/1500000019

[24] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-judge with MT-Bench and chatbot arena. In: Advances in Neural Information Processing Systems, vol. 36 (2023). https://doi.org/10.48550/arXiv.2306.05685 . arXiv:2306.05685

[25] Panickssery, A., Bowman, S.R., Feng, S.: LLM evaluators recognize and favor their own generations. In: Advances in Neural Information Processing Systems, vol. 37 (2024). https://doi.org/10.48550/arXiv.2404.13076 . arXiv:2404.13076

[26] Just, R., Jalali, D., Inozemtseva, L., Ernst, M.D., Holmes, R., Fraser, G.: Are mutants a valid substitute for real faults in software testing? In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 654–665 (2014). https://doi.org/10.1145/2635868.2635929

[27] Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P.: Taxonomy of real faults in deep learning systems. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 1110–1121 (2020). https://doi.org/10.1145/3377811.3380395

[28] Shumailov, I., Shumaylov, Z., Zhao, Y., Papernot, N., Anderson, R., Gal, Y.: AI models collapse when trained on recursively generated data. Nature **631**, 755–759 (2024) https://doi.org/10.1038/s41586-024-07566-y

[29] Jordon, J., Szpruch, L., Houssiau, F., Bottarelli, M., Cherubin, G., Maple, C., Cohen, S.N., Weller, A.: Synthetic Data – What, Why and How? arXiv:2205.03257 (2022). https://doi.org/10.48550/arXiv.2205.03257

[30] Saad-Falcon, J., Khattab, O., Potts, C., Zaharia, M.: ARES: An automated evaluation framework for retrieval-augmented generation systems. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 338–354 (2024). https://doi.org/10.18653/v1/2024.naacl-long.20

[31] Ribeiro, M.T., Wu, T., Guestrin, C., Singh, S.: Beyond accuracy: Behavioral testing of NLP models with CheckList. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 4902–4912 (2020). https://doi.org/10.18653/v1/2020.acl-main.442

[32] Morris, J., Lifland, E., Yoo, J.Y., Grigsby, J., Jin, D., Qi, Y.: TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 119–126 (2020). https://doi.org/10.18653/v1/2020.emnlp-demos.16

[33] Chen, T.Y., Kuo, F.-C., Liu, H., Poon, P.-L., Towey, D., Tse, T.H., Zhou, Z.Q.: Metamorphic testing: A review of challenges and opportunities. ACM Comput Surv **51**(1), 4–1427 (2018) https://doi.org/10.1145/3143561

[34] Perez, E., Huang, S., Song, F., Cai, T., Ring, R., Aslanides, J., Glaese, A., McAleese, N., Irving, G.: Red teaming language models with language models. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pp. 3419–3448 (2022). https://doi.org/10.18653/v1/2022.emnlp-main.225

[35] Wei, A., Haghtalab, N., Steinhardt, J.: Jailbroken: How does LLM safety training fail? In: Advances in Neural Information Processing Systems, vol. 36 (2023). https://doi.org/10.48550/arXiv.2307.02483 . arXiv:2307.02483

[36] Yuan, Y., Jiao, W., Wang, W., Huang, J.-t., He, P., Shi, S., Tu, Z.: GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In: International Conference on Learning Representations (ICLR) (2024). https://doi.org/10.48550/arXiv.2308.06463 . arXiv:2308.06463

[37] Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J.Z., Fredrikson, M.: Universal and Transferable Adversarial Attacks on Aligned Language Models. arXiv:2307.15043 (2023). https://doi.org/10.48550/arXiv.2307.15043

[38] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M.: Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, pp. 79–90 (2023). https://doi.org/10.1145/3605764.3623985

[39] promptfoo: promptfoo: Test Your Prompts, Agents, and RAGs. Computer software (2024). https://github.com/promptfoo/promptfoo

[40] UK AI Security Institute: Inspect: A Framework for Large Language Model Evaluations. Computer software (2024). https://inspect.

aisi.org.uk

[41] Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., Scialom, T.: GAIA: A benchmark for general AI assistants. In: International Conference on Learning Representations (ICLR) (2024). https://doi.org/10.48550/arXiv.2311.12983 . arXiv:2311.12983

[42] Shapley, L.S.: A value for n-person games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games, Volume II, pp. 307–317. Princeton University Press, ??? (1953). https://doi.org/10.1515/9781400881970-018

[43] Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems, vol. 30, pp. 4765–4774 (2017). https://doi.org/10.48550/arXiv.1705.07874 . arXiv:1705.07874

[44] Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. Chapman & Hall/CRC, ??? (1993). https://doi.org/10.1201/9780429246593

[45] Holm, S.: A simple sequentially rejective multiple test procedure. Scand J Stat **6**(2), 65–70 (1979)

[46] Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J Educ Behav Stat **25**(2), 101–132 (2000) https://doi.org/10.3102/10769986025002101

[47] Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw Test Verif Reliab **24**(3), 219–250 (2014) https://doi.org/10.1002/stvr.1486

[48] Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc **32**(200), 675–701 (1937) https://doi.org/10.1080/01621459.1937.10503522

[49] Kendall, M.G., Babington Smith, B.: The problem of m rankings. Ann Math Stat **10**(3), 275–287 (1939) https://doi.org/10.1214/aoms/1177732186

[50] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res **7**, 1–30 (2006)

[51] Benavoli, A., Corani, G., Mangili, F.: Should we really use post-hoc tests based on mean-ranks? J Mach Learn Res **17**(5), 1–10 (2016)

[52] Pratt, J.W.: Remarks on zeros and ties in the Wilcoxon signed rank procedures. J Am Stat Assoc **54**(287), 655–667 (1959) https://doi.org/10.1080/01621459.1959.10501526

[53] Kerby, D.S.: The simple difference formula: An approach to teaching nonparametric correlation. Compr Psychol **3**, 11–31 (2014) https://doi.org/10.2466/11.IT.3.1

[54] Cochran, W.G.: The comparison of percentages in matched samples. Biometrika **37**(3/4), 256–266 (1950) https://doi.org/10.1093/biomet/37.3-4.256

[55] McNemar, Q.: Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika **12**(2), 153–157 (1947) https://doi.org/10.1007/BF02295996

[56] Wilson, E.B.: Probable inference, the law of succession, and statistical inference. J Am Stat Assoc **22**(158), 209–212 (1927) https://doi.org/10.1080/01621459.1927.10502953

[57] Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: A practical and powerful approach to multiple testing. J R Stat Soc Ser B **57**(1), 289–300 (1995) https://doi.org/10.1111/j.2517-6161.1995.tb02031.x

[58] Hoenig, J.M., Heisey, D.M.: The abuse of power: The pervasive fallacy of power calculations for data analysis. Am Stat **55**(1), 19–24 (2001) https://doi.org/10.1198/000313001300339897

[59] Lakens, D.: Sample size justification. Collabra Psychol **8**(1), 33267 (2022) https://doi.org/10.1525/collabra.33267

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Fig1.pdf
- Fig1source.tex
- Fig3.pdf
- Fig2.pdf
- Fig5.pdf
- Fig4.pdf
- FigS1.pdf
- Fig1.tif
- Fig2.tif
- Fig4.tif
- Fig3.tif
- Fig5.tif
- FigS1.tif
- statscodeandresults.zip
- supplementary.zip