The AI Suggests, the Code Decides: Deterministic Grounding and Fail-Safe Tool Permissions in a Local-First Research Assistant

Surena Karimpour Ghannadi and Behrad Azimi

Immortal Horizon Technologies
suren.karimp@gmail.com, Behradazimi1@gmail.com

Abstract. Large language models (LLMs) now help researchers with literature reviews, but they still produce sources and quotations that do not exist, and LLM agents can take actions that nobody approved. Better prompts, human review and larger models make these failures rarer, but they do not bound them. This paper describes Agatha, a local-first desktop research assistant built on one principle: the model proposes, and deterministic code decides. When Agatha digests a paper, the model proposes claims together with character offsets into the article text. A deterministic normaliser then checks every offset against the article, copies the quotation from the article itself, assigns identifiers, and drops any claim whose quotation cannot be found. A fabricated quotation therefore becomes a missing record rather than a convincing false one. The same principle governs actions. Each of Agatha's 112 agent tools declares its effect, the resource it touches and whether its output can be trusted, and a missing declaration is treated as the most dangerous case. Content from untrusted sources marks the run so that later writes are checked, and sub-agents cannot write to their parent's state. We describe the design, report the engineering record behind it (152,032 source lines and 4,031 automated tests), and state the main open problem: the drafting stage still relies on prompt instructions, and extending deterministic checks from reading to writing is future work.

Keywords: Large Language Models, Hallucination, Citation Grounding, AI Agents, Fail-Safe Defaults, Local-First Software.

1 Introduction

Large language models (LLMs) are now part of everyday research work. They summarise papers, suggest related work and draft text. They also make things up. A model can cite a paper that does not exist, report a number that nobody measured, or present a paraphrase as an exact quotation, and the text reads just as fluently when it is wrong [8]. In one study of short literature reviews written by ChatGPT, 55% of the citations produced by GPT-3.5 and 18% of those produced by GPT-4 were fabricated [12]. For a researcher, a confident false citation is worse than no citation, because fluency no longer signals that anything was checked.

A second failure appears when the model becomes an agent: a program that uses the model to choose and run tools, such as web search or writing a file. An agent can call a tool it was never meant to use, or follow instructions hidden in a web page it has just read [7]. When nothing records which request allowed which action, there is nothing to audit afterwards.

The usual remedies are better prompts, human review and larger models. Each helps, but none sets a limit. A prompt is a request to the model, not a rule that the system enforces, and its effect can change silently when the model version changes. A human reviewer catches the errors they happen to recognise and misses the rest. A larger model makes the same kind of mistake, only less often.

This paper takes a structural view: both failures come from trusting the output of a probabilistic text generator without a check. Our answer is a division of labour. The model proposes; deterministic code, which always gives the same answer for the same input, decides. If a claim's evidence and identity are decided by code rather than by the model, a fabrication produces an absence rather than an error. If what an agent may do is decided by declarations that code checks, an undeclared action is refused rather than merely discouraged.

We describe how this principle is applied in Agatha, a local-first desktop application for scholarly research, and in Horizon core, the agent engine it is built on. The contributions are:

- **Deterministic grounding.** A claim normaliser in which the model supplies only positions in the source text. Code checks them, copies the quotation from the source and assigns identifiers, and claims that cannot be resolved are dropped (Sect. 4).
- **Fail-safe tool permissions.** Every tool declares its effect, its resource and the trust level of its output. A missing declaration defaults to the most dangerous case, untrusted content marks the run, and sub-agents are isolated from their parent (Sect. 5).
- **An honest engineering record.** The testing and verification practices behind the system, and a plain statement of what code does not yet check (Sects. 6 and 7).

Section 2 reviews related work, Sect. 3 gives an overview of the system, and Sect. 8 concludes.

2 Related Work

Hallucination and Citation. Ji et al. survey hallucination in natural language generation, where generated text is fluent but unfaithful to its source or to the facts [8]. Retrieval-augmented generation conditions the model on retrieved documents [10], and the ALCE benchmark measures whether generated statements are supported by the passages they cite; on its ELI5 dataset, even the best models tested lacked complete citation support about half of the time [6]. These approaches give the model better evidence, or measure how well it uses that evidence. Agatha is complementary: it does not ask the model to quote correctly, because the quotation text is never taken from the model at all.

Agent Security. Greshake et al. showed that instructions planted in content that an LLM-integrated application retrieves can take control of its behaviour, an attack they call indirect prompt injection [7]. Recent defences move protection out of the model and into the surrounding system. CaMeL separates the control flow derived from the user's request from untrusted data, and attaches capabilities to values so that policies are enforced when tools are called [3]. Beurer-Kellner et al. describe design patterns that constrain an agent once it has processed untrusted input [1]. These systems build on long-standing security principles: capabilities as tokens of authority [5], fail-safe defaults, under which access is denied unless it is explicitly permitted [11], and information-flow control, which tracks where data came from [4]. Agatha applies these ideas at the level of individual tool declarations in a shipping desktop application.

Local-First Software. Kleppmann et al. argue for software in which the user's own device holds the primary copy of their data and keeps working without a network connection [9]. For research tools this matters, because unpublished results, interview data and drafts are often sensitive.

3 System Overview

Agatha is a desktop application for scholarly research; version 9.0.0 is shipping. It is built on Horizon core, a portable agent kernel that contains the agent runtime, the tool framework, the model providers and the multi-agent orchestration. The kernel was extracted from Agatha with its history preserved, and a second product, a headless coding service, also uses it. An automated dependency check in continuous integration forbids the kernel from depending on any product, so a product can change without a change to the kernel. Table 1 summarises the application.

Table 1. Agatha at a glance (version 9.0.0).

Dimension	Count	Detail
Scholarly sources	12	arXiv, PubMed, Crossref, Semantic Scholar, OpenAlex, CORE, DOAJ, DOAB, plus web, news, encyclopedia and general web search
Agent tools	112	Each declares its effect, resource and provenance
Analysis passes	20	Evidence tables, contradiction detection, topic modelling, trends, novelty, plagiarism, citation chasing, corpus extraction
Model providers	11	Three run fully locally, with no external service

Agatha works in three stages (Fig. 1):

- **Acquire.** Agatha searches twelve sources with boolean query operators, imports papers by DOI, removes duplicates, detects paywalls and bot guards, and extracts the

article text, using OCR and vision-based table capture where needed. It also synchronises with the Zotero reference manager.

- **Digest.** For each paper, one model pass proposes an anatomy of the article: its segments, claims and methods, each with character offsets into the article text. A deterministic normaliser then decides what survives (Sect. 4). A second pass checks the publication metadata against the article and writes corrections back to the record.
- **Synthesise.** Claims are packed into context blocks with a fixed size budget and passed to drafting agents, one per section. An independent reviewer agent checks each section and is instructed to open the cited sources itself, and the drafting agent revises until the reviewer is satisfied (Sect. 5). A deterministic linter gates assembly. The output is LaTeX, and citations and the bibliography are attached without a model.

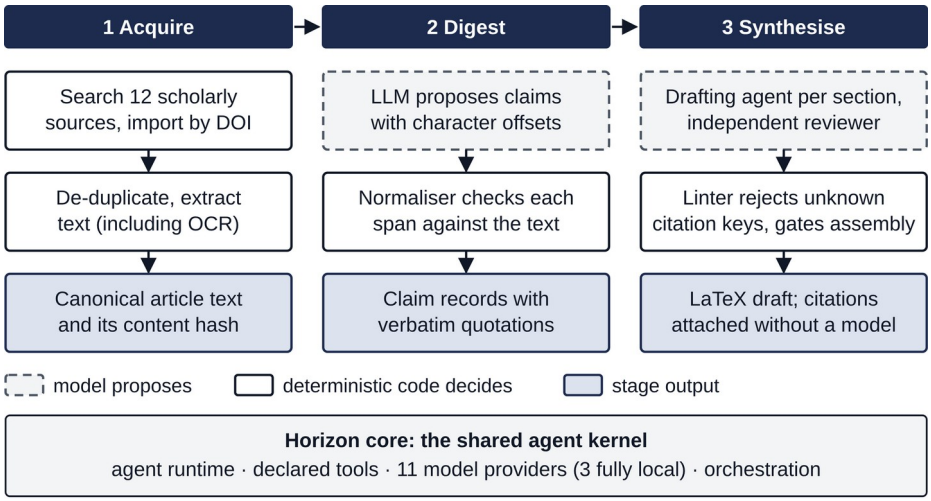


Fig. 1. Agatha's three stages on top of Horizon core. Dashed boxes are steps in which a model proposes; solid boxes are steps in which deterministic code decides.

Local-First by Design. Retrieval runs inside the application process. A hybrid index combines statistical ranking over a Unicode-aware tokenizer with local transformer embeddings, merges the two result lists with reciprocal rank fusion [2], and can rescore the top results with a local cross-encoder. Embeddings are cached in SQLite under a key that includes the model name, so changing the embedding model cannot return stale vectors. The library, the index and the checks described below run on the user's computer, three of the eleven model providers run fully offline, and the application contains no usage telemetry.

4 Grounding: How Agatha Checks a Claim

A claim record carries an identifier, a sentence, a classification, a confidence value supplied by the model, and zero or more quote spans. A quote span is a pair of character offsets [*s*, *e*) that marks where the supporting quotation begins and ends in the canonical article text: the single stored version of the article from which all offsets are counted. The model proposes these records. A deterministic core then decides what is kept:

1. every offset is clamped into the canonical text, and an inverted or empty span is rejected;
2. the quotation text is re-sliced from the canonical text and is never taken from the model;
3. a claim whose quotation cannot be resolved is dropped entirely;
4. the segment a claim belongs to is derived from span containment, so the model cannot file a claim under the wrong section;
5. identifiers and cross-references are assigned by the normaliser, so the model cannot invent one;
6. an unrecognised classification falls back to a fixed default instead of passing through as free text;
7. a content hash of the canonical text is stored, so that later changes to the text can be detected.

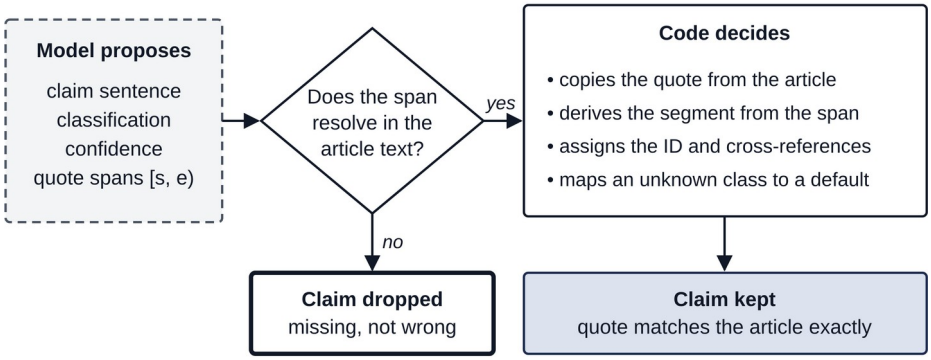


Fig. 2. How the normaliser treats a proposed claim. A claim whose span does not resolve is dropped; a claim that survives carries a quotation copied from the article by code. The model never supplies the quotation text, the identifier or the segment.

The result is a guarantee about structure (Fig. 2). A hallucinated offset cannot produce a claim, because the claim is only built once the offset resolves. A hallucinated quotation cannot survive, because its text is overwritten with the source text. Fabrication therefore yields a missing record rather than a plausible wrong one, and every later stage can treat the surviving claims as anchored to the source without checking them again. This is neither prompt engineering nor a filter applied after the fact: the model is simply never given the parts of the record it could get wrong.

What the normaliser does not guarantee is that the claim sentence is a faithful reading of its quotation. A model can still attach a real quotation to an overstated claim. Such an error is easy to check, because every surviving claim points to exact text in the source, but the design does not remove it (Sect. 7).

Honest Degradation. The same attitude extends to analyses that cannot run. An empty list of contradictions would read as “no contradictions found”, which is a finding; an analysis that did not execute is not a finding. Agatha’s reports therefore carry an explicit status and a note, and they say when the examined window did not cover the whole corpus. Analysis outputs that name articles outside the corpus the model was shown are dropped, because they used to appear as findings. In retrieval, the availability of each stage is reported separately, so a failed first-run download of the embedding model can be told apart from a deliberate keyword-only configuration.

Non-Latin Scripts. The retrieval tokenizer was originally limited to ASCII characters, which made multi-language text unsearchable, uncitable and unextractable. The same assumption was found in four separate code paths, and all four are fixed with regression tests. Researchers read and write in many languages, so correctness for non-Latin scripts is treated as a requirement rather than a localisation detail.

5 Authority: What an Agent May Do

Grounding limits what the model may assert. The second half of the design limits what an agent may do. Every tool that an agent can reach declares three properties (Table 2), and each one fails safe in the sense of Saltzer and Schroeder [11]: if a declaration is missing, the system assumes the most dangerous case.

Table 2. Tool declarations and their fail-safe defaults.

Property	What the tool declares	If the declaration is missing
Effect	Whether it only reads or also writes	The call is treated as a write
Resource	A typed key for what it touches	The call conflicts with every other call
Provenance	Whether its output can be trusted	The output is treated as untrusted

The resource declaration is a typed key rather than a string, and a container key and a leaf key are distinct types. This makes it impossible for an opaque identifier to be read as a namespace, a defect that existed in the first version. A concurrency planner uses the declarations to run non-conflicting tool calls together, in waves. All 112 tool specifications in Agatha are declared. Their tests run through the planner itself and count the resulting execution waves, rather than inspecting a list of declarations, because a test that only inspects a list can pass while the protection is broken.

Taint Propagation. Content from an untrusted source, such as a fetched web page, is presented to the model as data and marks the run as tainted, an idea that goes back to information-flow control [4]. The mark lasts for the rest of the turn, so a write that follows a fetch is still checked however many trusted reads come in between. The mark is re-derived from the saved transcript when a run resumes, so a restart cannot clear it, and a configuration option makes the check blocking rather than advisory. This targets the indirect prompt injection pattern, in which instructions hidden in retrieved content try to trigger a write [7].

Delegation and Review. A sub-agent is an agent started by another agent to do part of its task. When an agent takes on a large task, it divides the work among sub-agents. Every sub-agent that produces a result is paired with its own review sub-agent, and the two work in a loop: the worker revises its result until the reviewer is satisfied. Sub-agents are isolated from one another and do not share their context, so each one sees only its own part of the task, and content that one sub-agent reads does not enter another's context. The reviewer is itself a model, so this loop raises quality but is not a guarantee; the deterministic checks of Sect. 4 remain the hard boundary.

Sub-Agent Isolation. Isolated context has to be matched by isolated authority. This work began with a real finding: a sub-agent that advertised a restricted tool list was able to call a different tool and save its result. The advertised list only shaped which tool descriptions the model was shown, while dispatch resolved names against the whole registry. The fix has two parts, and each is insufficient alone. The sub-agent now receives a genuinely filtered registry, so a tool that is absent from it is refused. In addition, the persistence hooks are removed for the nested run, because those hooks save by project identifier and would otherwise overwrite the parent's state, even from a read-only copy. The general lesson is that what a model is shown is not the same as what it can do.

6 Engineering Record

There has been no formal user evaluation of Agatha yet, and we report no measured fabrication rates. The evidence we can offer today is the engineering record of the code base, measured from the working repositories on 5 October 2026.

Table 3. Automated tests by suite, measured on 5 October 2026.

Suite	Files	Tests	Note
Research application	138	1,597	Integration suites that drive real routes
Renderer and packaging	57	582	Includes the Electron shell
Kernel	100	1,458	Per-module coverage floors
Shared, models, protocol	4	83	Contract level
Coding service	22	311	Includes protocol conformance
Total	321	4,031	0.7 test lines per source line

Across the four shipping repositories there are 152,032 source lines and 4,031 tests in 321 files, or 0.7 lines of test code per line of source (Table 3). All 351 defects tracked for Agatha are closed. Both mature repositories run their full lint, type-check and test suites on Linux and Windows in continuous integration; the Windows job exists because a whole class of defects turned out to be POSIX assumptions that a Linux runner hides. Unit tests never touch the network or real user data: a fixture harness starts a fake model server with stubbed search and fetch, so paths that depend on a model can be tested deterministically.

Four ratchets stop quality from sliding back: coverage floors per module, raised six times so far, with a guard that fails when a module drops below its floor; type checking of the test code itself, which surfaced 395 errors in the product and 211 in the kernel when it was first enabled, and whose product baseline is now zero; a checked-in baseline for component complexity; and contract tests that pin the public surface and the protocol.

Verification by Removal. The project's standing rule is that every protection must be verified by removal: revert the fix and watch a specific test fail. In the project's own words, "a test that cannot fail is a comment". The register that tracks this rule also records where it was broken, including review findings that turned out to be wrong on re-verification and are marked as such.

Failures that Report Success. Several defects found this way produced no error at all. The sub-agent dispatch gap in Sect. 5 is one. In another, a duplicate type library made schema introspection miss schemas built by a consumer, so structured list extraction silently returned empty results. In a third, found in an internal audit and fixed, a blocked folder nested inside a workspace was fully readable, because directory enumeration checked only the base directory. Such failures are invisible to code reading, to tests on a clean tree and to type checking; they were found only by systematically auditing the places where the architecture could fail quietly. We believe this class of defect deserves more attention in LLM application engineering, because a model-driven system keeps producing plausible output even when one of its components is broken.

7 Limitations

We state the limits plainly, because a grounding claim that cannot be falsified is a claim, not a control.

From Reading to Writing. The checks are strongest at extraction. In the drafting stage, the requirement that an agent open a claim's source before using it is a prompt rule, and nothing yet records whether the agent complied: the tests assert that the prompts contain the instruction, not that the tool was called. The nearest mechanical check is the linter, which rejects any citation key that does not resolve to a digested article, as well as malformed LaTeX and duplicated paragraphs, and assembly waits for an independent review verdict. Even so, a drafting agent that never verifies can still produce a section that is accepted.

Testing the Guarantee End to End. The integration harness cannot yet detect citation fabrication on its own: a run in which every citation was invented would still pass. Adding a negative test, in which a deliberately fabricated citation key must be rejected, is our first planned step, because until it exists further improvements to grounding cannot be measured.

Faithfulness and Scope. As noted in Sect. 4, the normaliser guarantees that a quotation exists in the source, not that the claim built on it is a fair reading. There is no user study yet. The capability model described here covers Agatha's 112 agent tools; a fuller derivation model is specified but not yet implemented. Agatha is tested on Windows and Linux only.

8 Conclusion and Future Work

Agatha applies one principle to two problems. For what a model asserts, the model proposes positions in the source and code decides what the quotation says, so fabrication produces absence instead of error. For what an agent does, every tool declares its effect, its resource and its provenance, and anything left unsaid is treated as dangerous. Both mechanisms ship in a working local-first application.

Three steps come next. First, teach the integration harness to fail on fabricated citations, so that improvements to grounding become measurable. Second, implement the first phases of the specified capability model, in which every principal carries its lineage and a derived grant is never larger than its parent [5]. Third, and most open, apply the same derivation to assertion: derive what a drafting agent may state from the evidence it has actually opened, so that today's prompt-enforced verification becomes an invariant that code checks. We also plan to pilot Agatha with university researchers and to measure, with them, how many fabricated citations it stops and how much reviewing time it saves.

Disclosure of Interests. The authors are affiliated with Immortal Horizon Technologies, which develops Agatha and Horizon core, the software described in this paper.

References

1. Beurer-Kellner, L., et al.: Design patterns for securing LLM agents against prompt injections. arXiv preprint arXiv:2506.08837 (2025)
2. Cormack, G.V., Clarke, C.L.A., Büttcher, S.: Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 758–759. ACM, New York (2009)
3. Debenedetti, E., et al.: Defeating prompt injections by design. arXiv preprint arXiv:2503.18813 (2025)
4. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
5. Dennis, J.B., Van Horn, E.C.: Programming semantics for multiprogrammed computations. Commun. ACM **9**(3), 143–155 (1966)
6. Gao, T., Yen, H., Yu, J., Chen, D.: Enabling large language models to generate text with citations. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 6465–6488. Association for Computational Linguistics, Singapore (2023)
7. 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. ACM, New York (2023)
8. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12), 248:1–248:38 (2023)
9. Kleppmann, M., Wiggins, A., van Hardenberg, P., McGranaghan, M.: Local-first software: you own your data, in spite of the cloud. In: Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2019), pp. 154–178. ACM, New York (2019)
10. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems 33 (NeurIPS 2020), pp. 9459–9474. Curran Associates, Red Hook (2020)
11. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proc. IEEE **63**(9), 1278–1308 (1975)
12. Walters, W.H., Wilder, E.I.: Fabrication and errors in the bibliographic citations generated by ChatGPT. Sci. Rep. **13**, 14045 (2023)