

Trust, but Audit: Measuring an LLM Agent Pipeline that Extracts Software Capabilities from Documentation

Ahmed ElBatanony

Innopolis University

Innopolis, Russia

a.elbatanony@innopolis.ru

Manuel Mazzara

Innopolis University

Innopolis, Russia

m.mazzara@innopolis.ru

Abstract

Studies that build new software engineering datasets with large language models (LLMs) have no human gold standard against which to measure the models' errors. We report the design and measured reliability of an LLM agent pipeline that extracts user-observable capabilities from business-application documentation without a human annotator. Agents quote verbatim evidence, a script checks every quote, two blind runs extract independently, and a verifier and an adjudicator rule with recorded reasons. On 25 applications (2,021 statements), no quote failed the script check, and the two runs agreed on only 79.2% of capabilities. A blind audit of 150 random statements found 97.3% acceptable, but only 62.2% were supported by their excerpt alone. Agreement between the pipeline's own roles was near-perfect and told us neither. We give four lessons for using LLMs as annotators.

CCS Concepts

• **Software and its engineering** → **Requirements analysis**; • **Computing methodologies** → *Natural language processing*.

Keywords

large language models, data annotation, reliability, empirical software engineering, feature extraction, business applications

1 Introduction

LLMs now do work that empirical software engineering used to assign to trained human coders: labelling issues, classifying commits, extracting requirements. Studies that compare LLM labels with existing human labels find them competitive for many tasks [1, 7], and guidelines now ask authors to validate and report such use [10–12]. A study that *builds* a new dataset with an LLM has a harder problem: there is no gold standard, and the dataset's credibility rests on errors that are rarely measured. One recent study that did check found 17.38% of LLM-extracted software features inaccurate on manual review [9].

We met this problem while building a dataset of business applications and their functional capabilities. Low-code platforms, model-driven toolkits and LLM application generators all target “business applications”, yet no reference population of the class exists. Our dataset samples 165 applications from GitHub and from a commercial software directory, stratified by the 13 categories of the APQC Process Classification Framework [2], and describes each by capability statements taken from its own documentation. The dataset itself is the subject of a separate article. This paper is about the instrument: an LLM agent pipeline meant to stand without human annotators, and what we could and could not measure about it.

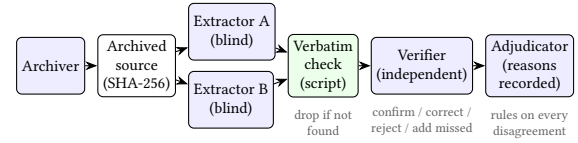


Figure 1: Extraction pipeline. Blue steps are LLM instances under fixed, published briefs; green is a deterministic script. The archiver chooses which documentation pages to fetch. Only items the adjudicator marks undecidable reach a human.

Our contributions are: (1) a pipeline that turns four reliability mechanisms into checkable artefacts (Section 3); (2) measurements from a pilot, a model calibration, a main run on 25 applications and a blind audit (Section 4); and (3) four lessons (Section 5).

2 Task and Setting

The unit of analysis is a *feature statement*: one user-observable capability in one of two forms, “The system lets ⟨actor⟩ ⟨action⟩ ⟨object⟩ [⟨condition⟩]” for user actions and “The system automatically ⟨action⟩ ⟨object⟩ when ⟨event⟩” for automations. Each statement carries a verbatim excerpt of its source, the source URL and the SHA-256 hash of the archived source text. Sources are first-party only. Testimonials, quality claims, pricing, compliance and deployment facts are never evidence. The task is feature mining from product descriptions [4, 8], here done by LLM agents.

The task is hard for an LLM in three ways. Marketing pages invite *inference beyond the text*. The same capability is *repeated* across sections. And statement counts follow writing style: in our pilot, one vendor’s long feature page yielded 98 statements while a product of comparable breadth yielded 16, so analyses should use capability presence, not counts.

3 The Pipeline

Figure 1 shows the pipeline. Each role is an agent with file access, running under a fixed, published brief.

M1. Evidence-bound prompt. The extraction prompt requires one capability per statement, keeps conditions, names the kind of integration rather than the vendor, and requires an excerpt “copied exactly from the source”. Excluded material must be listed under *skipped* with a reason, so exclusions are visible.

M2. Script check. Every excerpt is matched, after whitespace normalisation, against the archived text, and every statement must begin with one of the two form prefixes; failures are dropped and counted. The script does not check the rest of the template, nor whether the excerpt suffices.

Table 1: Pilot statement accuracy, strong-model extractor (4 applications, 224 statements; verbatim check over all 8 pilot applications).

Measure	Value
Excerpts found verbatim in source	317 / 317
Verifier verdicts (confirmed / corrected / rejected)	185 / 33 / 6
After adjudication	186 / 33 / 5
of the corrections: form only / content	29 / 4
Supported share (not rejected)	97.8%
Unchanged or form-only correction	96.0%
Missed capabilities added (share of final set)	5 (2.2%)
Adjudicator overturns of reviewed verdicts	1 of 39

M3. Two blind extractions. Two instances extract from the same sources without seeing each other’s output, in the spirit of sampling several reasoning paths [13]; their statements are concatenated. Blindness is enforced by instruction and file layout, not by a sandbox.

M4. Verification with recorded adjudication. A third instance, which produced neither extraction, confirms, corrects or rejects each statement against its excerpt, marks duplicates between the runs, and proposes capabilities both runs missed. A stronger model rules on every correction, rejection, duplicate and proposal and records its reasons; confirmations are not sent to it. Only items it marks *undecidable* would reach a human.

The same pattern screened the sample: a screener model applied fixed inclusion criteria to every candidate application; every uncertain case was adjudicated and every exclusion reviewed (44 rulings recorded, with the screener’s original verdict).

Model roles follow cost: archiving, extraction, verification and screening use a mid-tier model, adjudication a stronger one. In the main run these were `claude-sonnet-5-5` and `claude-opus-5-5` at medium reasoning effort, run as Claude Code 2.1.304/2.1.308 subagents on 21.–28.10.2026. Temperature and seed are not exposed.

4 Measurements

4.1 Pilot

The pilot used the strong model as extractor on eight applications, two per source type. Four (two open-source, two commercial) went through verification and adjudication; the other four came from source types later dropped and were checked only by script.

Table 1 gives the results. Of 224 statements, 5 were rejected: two evidenced only by a customer story, one internal data-model claim, and two that inferred beyond their excerpt. Of the 33 corrections, 29 only rewrote a system-initiated behaviour into the automation form, a gap in the first prompt that the pilot exposed.

4.2 Calibration: can a cheaper extractor match?

For two pilot applications we adjudicated the full set of capabilities (109) and compared extraction setups by the share of these each setup covers, judged by meaning. The reference set grew out of the strong model’s own extraction, and coverage was judged by a strong-model session that knew which setup produced which list.

Table 2: Recall against 109 adjudicated capabilities.

Extraction setup	Covered	Recall
Strong model (Opus), one pass	108	99.1%
Mid-tier model (Sonnet), one pass	101	92.7%
Sonnet, one pass + Sonnet verifier	104	95.4%
Sonnet, two blind passes + verifier	107	98.2%

Table 3: Main run on the stratified subset (25 applications, 217 archived pages).

Measure	Value
Statements extracted (run A / run B)	1,879 / 1,940
Dropped by script (not verbatim / wrong form)	0 / 0
Verifier verdicts (confirmed / corrected / rejected)	1,890 / 76 / 1,853
After adjudication	1,895 / 77 / 1,847
duplicates / other rejections	1,787 / 60
Supported share of the 2,032 non-duplicates	97.0%
Missed capabilities accepted / proposed	49 / 56
Final statements	2,021
Overturned (other verdicts / duplicates)	1 of 136 / 6 of 1,793
Extracted final statements found by both runs	1,561
by run A only / run B only	236 / 175
Run-to-run overlap (Jaccard)	79.2%
Single-run coverage of the final set (A / B)	88.9% / 85.9%

The table therefore compares setups; it does not measure absolute recall.

One pass of the mid-tier model trailed the strong model by 7 capabilities, among them an audit log and an escalation rule (Table 2). The verifier recovered 3 and a second blind pass 3 more. On two applications these differences are a few items each; we took them as reason enough to adopt two blind passes plus a verifier, not as an effect estimate.

4.3 Main run

The main run applied the adopted setup to a subset fixed by rule before the run: per source type and PCF category, the first application in draw order with archived sources (12 open-source, 13 commercial; the open-source frame has no application in one category). The rule favours well-documented applications. The archiver fetched 241 pages, 217 of them non-empty. One application from the trial was adjudicated under an earlier brief that read whole sources; for the other 24 the adjudicator read a script-prepared file of the items to rule on.

No quote failed the script check (Table 3). 1,787 of the 1,847 rejections removed a capability already stated. Of the 2,032 statements not removed as duplicates, the verifier rejected 60; we call the remaining 97.0% the *verifier acceptance rate*, since 1,895 of the 2,021 final statements were checked by the verifier alone. The adjudicator overturned 1 of the 136 other verdicts and 6 of 1,793 duplicate rulings, but it sees the verifier’s verdict and reason.

Run overlap is computed from the verifier’s duplicate judgments: a final statement and its upheld duplicates form a group, found by A, B or both. The runs agreed on 79.2% of groups. One

Table 4: Blind audit of 150 random final statements by stratum (confirmed, corrected, added), weighted to the final set.

Check	Conf.	Corr.	Added	Weighted
Excerpt alone supports	57/90	15/30	11/30	62.2%
Supported in context	90/90	28/30	29/30	99.7%
One capability	86/90	29/30	30/30	95.7%
Actor not invented	83/90	28/30	26/30	92.1%
Acceptable as is	88/90	26/30	29/30	97.3%

run alone covered 85.9%–88.9% of the final set, and the verifier and adjudicator added 49 that both missed. Single-run groups mix true misses with differences in granularity and with duplicates the verifier missed (we found at least two in the release), so these are relative figures. Single-run coverage here is lower than in the calibration, but model, prompt version and applications all differ between the two.

4.4 Blind audit

Every figure so far is the pipeline judging itself. We therefore drew a stratified random sample of 150 final statements (90 confirmed, 30 corrected, 30 added) and gave it to a fresh strong-model instance that saw only statement, excerpt and source page, with no pipeline file or verdict. It judged each row on seven checks; Table 4 gives the main ones, weighted to the final set.

97.3% of statements were acceptable as is (95% CI 94.4%–100.0%). The 7 rejected rows included a partner programme presented as a capability, an invented “supervisor” role, and two corrections that added detail the page does not state. The striking result is excerpt sufficiency: only 62.2% of excerpts support their statement on their own, although 99.7% are supported once the surrounding text is read. Our own brief likely contributed, by asking for “the shortest contiguous span”, and the verifier accepted such excerpts. The script guarantees that a quote exists, not that it is evidence. Corrected statements were the weakest stratum (26/30 acceptable).

4.5 What went wrong

- **Wrong-genre evidence** (customer stories as proof). Caught by the verifier and excluded by the prompt.
- **Insufficient excerpts and inference beyond them.** Not caught by the script; partly caught by the verifier; measured only by the audit.
- **Form drift** (rules written outside both forms). 12 prefix drops in two trial applications, 11 of them in one, under the first brief; none in the 24 applications extracted after a one-line reminder. Too few trial applications to credit the reminder.
- **Misses** of secondary views and administrative features. Reduced by the second pass and the verifier.
- **Contaminated frames.** 70% of 392 GitHub repositories with business topics were not business applications, including 42 AI prompt packs; the screener made each exclusion visible.
- **Infrastructure failure.** A first attempt with about twenty parallel agents hit the provider’s usage limit; most agents

stopped mid-task. Stage files checked by script kept partial output out of the data.

5 Lessons

- (1) **Check quotes by script, and check that they suffice.** Agents with file access never fabricated a quote (0 of 4,136), so the check is a cheap invariant rather than a filter. The real risk is a genuine quote too short to carry the statement. Ask for excerpts that stand alone and have the verifier judge the excerpt alone.
- (2) **Repeat before you upgrade.** Two blind runs disagreed on a fifth of capabilities, and a second pass plus a verifier recovered most of what one pass of the cheaper model missed in calibration. Evidence is observational, but repetition is cheap.
- (3) **Pilot the prompt, not just the model.** 29 of 33 pilot corrections came from one gap in the statement forms.
- (4) **Measure from outside the pipeline.** Roles that see each other’s verdicts agree: the adjudicator overturned 1 of 136 reviewed verdicts, and a κ [3] over all statements would count unreviewed confirmations as agreement. Only the blind audit found the excerpt problem and the invented actors.

6 Threats to Validity

All roles, including the auditor, are models from one vendor, and the auditor is the same model as the adjudicator; shared blind spots stay invisible, and the audit is not a human audit. It covers 150 rows from 25 applications. Calibration rests on two applications and a reference set built from the strong model’s output. Source texts are not redistributed; a reader can re-fetch each URL and compare the hash, but pages change. Agents cannot be run with a fixed seed, so a re-run will not reproduce the statements; the run overlap measures that variation.

7 Related Work

Ahmed et al. [1] and Gilardi et al. [7] compare LLM labels with existing human labels; Pangakis et al. [10] argue that every automated annotation task needs its own validation. We had no human labels, so we built validation into the pipeline and added a blind audit. LLM-as-a-judge work [14] reports position and self-preference biases that motivate separate instances; citation-grounded generation [6] checks claims against quoted evidence, as our script does. Onagh and Nayebi [9] extract software features with LLMs and report a 17.38% error rate on manual review, a reference point for our audit, though task and models differ. Feature mining from product descriptions [4, 8] motivates the task, and an earlier study of feature concentration in consumer applications [5] used a single human coder. Our reporting follows recent guidelines for LLM use in SE studies [11, 12].

8 Conclusion

An LLM agent pipeline can produce an auditable dataset without a human in every loop: quote evidence, check quotes by script, extract twice blind, and adjudicate with recorded reasons. But agreement inside the pipeline says little. Only a blind audit showed that 97.3% of our statements are acceptable while more than a third of their

excerpts do not stand alone. Prompts, scripts, per-agent provenance, adjudication records and the audit are released with the dataset; for internal review every record is browsable at <https://pub10.vps.elbatanony.com>.

References

- [1] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. 2025. Can LLMs Replace Manual Annotation of Software Engineering Artifacts?. In *Proceedings of the 22nd International Conference on Mining Software Repositories (MSR)*.
- [2] APQC. 2024. Process Classification Framework, Cross-Industry, Version 7.4. APQC, Houston, TX.
- [3] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.
- [4] Jean-Marc Davril, Edouard Delfosse, Negar Hariiri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. 2013. Feature Model Extraction from Large Collections of Informal Product Descriptions. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 290–300.
- [5] Ahmed ElBatanony and Giancarlo Succi. 2021. The Pareto Distribution of Software Features and No-Code. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Beyond Code: No Code (BCNC '21)*. ACM, 18–22. doi:10.1145/3486949.3486964
- [6] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling Large Language Models to Generate Text with Citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6465–6488.
- [7] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. ChatGPT Outperforms Crowd Workers for Text-Annotation Tasks. *Proceedings of the National Academy of Sciences* 120, 30 (2023).
- [8] Negar Hariiri, Carlos Castro-Herrera, Mehdi Mirakhorli, Jane Cleland-Huang, and Bamshad Mobasher. 2013. Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1736–1752.
- [9] Elmira Onagh and Maleknaz Nayebi. 2025. Extension Decisions in Open Source Software Ecosystem. arXiv:2507.23168 [cs.SE]
- [10] Nicholas Pangakis, Samuel Wolken, and Neil Fasching. 2023. Automated Annotation with Generative AI Requires Validation. arXiv:2306.00176
- [11] June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the Silence: the Threats of Using LLMs in Software Engineering. In *Proceedings of the 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*.
- [12] Stefan Wagner, Marvin Muñoz Barón, Davide Falessi, and Sebastian Baltes. 2025. Towards Evaluation Guidelines for Empirical Studies Involving LLMs. In *IEEE/ACM International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE)*.
- [13] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations (ICLR)*.
- [14] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.