AI Cannot Review What Humans Catch: An Empirical Study of Agent Instruction File Review

Wenhao Yang
Peking University
Beijing, China
yangwh@stu.pku.edu.cn

Runzhi He
Peking University
Beijing, China
rzhe@pku.edu.cn

Minghui Zhou
Peking University
Beijing, China
zhmh@pku.edu.cn

***Abstract*—Agents increasingly shape software, and instruction files steer them. Files such as `AGENTS.md`, `CLAUDE.md`, `.cursorrules`, and `SKILL.md` act as executable policy: prose that a machine reads and acts on. Unlike ordinary documentation, a single edit to one of these files can silently redirect how an agent behaves across an entire repository. Natural-language rules have no per-change oracle, so they slip past the compilers, type-checkers, and test suites that guard code. Review is therefore the only quality gate, but no prior work has studied how that review happens.**

We analyzed 2,870 instruction-file pull requests across 2,246 repositories. The gate is largely broken: over 70% of instruction-file changes receive no substantive review. When an AI reviewer substitutes for a human, it is *structurally blind* to the concerns that most need human judgment. It catches surface structure but misses factual errors, necessity questions, and staleness problems. We contribute the first taxonomy of instruction-file review concerns (16 categories), which exposes this sharp human/AI divergence, and a benchmark of 1,962 human-checked concerns across 468 PRs. We ran three widely-used agents (Codex, OpenCode, and Claude Code) against the benchmark, giving each reviewer the full repository at review time. They recovered only 1.3–3.9% of the concerns humans raise. Even the strongest current agents cannot substitute for human attention on these files.

***Index Terms*—coding agents, agent instruction files, code review, large language models, empirical software engineering, open source software**

I. Introduction

Files such as `AGENTS.md`, `CLAUDE.md`, `.cursorrules`, and `SKILL.md` tell an agent how to behave inside a repository: which conventions to honor, which commands to run, which paths never to touch. In just a few years they have become a de-facto standard for agent-assisted development: among GitHub repositories that configure an agentic coding tool, 90.6% ship such a Markdown instruction file [1]. These files are **executable policy written in natural language**. They are ordinary prose, carrying none of the formal structure that makes code checkable. A machine reads them as instructions and acts on them. Because the agent acts on that prose literally, a single careless line changes what the agent *does*, not merely how the file reads. That change then propagates into everything the agent does. Such prose can be wrong in ways that matter: an `AGENTS.md` rule that directs the agent to build with a Go release that did not yet exist when the rule was written [2], or a checked-in agent-permission file that grants a shell tool whose `-exec` parameter a maintainer warned could be "leveraged by a prompt injection" [3].

Unlike the code they govern, agent instruction files escape the automated checks that guard software. Compilers, type-checkers, linters, and test suites encode a checkable definition of correctness and reject faulty changes before review. A natural-language rule has no comparable per-change oracle for whether it is true, necessary, current, or safe [4], [5]. Prompt- and agent-evaluation frameworks assess only what an instruction set *causes* an agent to do over a fixed task suite [6], not whether a given edit is sound. Review is therefore the only quality gate for these files, yet no one has studied how their changes are reviewed. Understanding that landscape is a first step toward governing and improving these AI practices.

We address this with a census of 2,870 instruction-file pull requests across 2,246 popular repositories. We find that this gate is largely broken in two ways: **human review is mostly absent**, and **where an AI reviewer substitutes for a human, it is structurally blind** to the concerns that most need human judgment. We ask four research questions.

RQ1 (Origin): Who authors instruction file changes, and how do they enter the repository? Finding: Instruction-file changes increasingly originate from AI and rarely arrive alone. Roughly one-third of the PRs are AI-authored, and 69% ride along inside a larger PR, dividing the reviewer's attention.

RQ2 (Coverage): To what extent and by whom are instruction file changes reviewed? Finding: 71.9% of changes receive no substantive review. Where review does occur, humans give rubber-stamp approvals 18.9% of the time and AI reviews are superficial (review-shaped walkthroughs that render no evaluative judgment) 39.5% of the time. Even among AI-authored changes, 78% still pull a human in to make the judgments the AI does not.

RQ3 (Divergence): what concerns do human and AI reviewers raise, and how do they diverge? Finding: Human and AI reviewers do not merely diverge: AI is *structurally blind* to the judgment-dependent concerns humans raise. Humans focus on completeness and factual correctness (24.6% of their concerns vs 5.4% of AI's), while AI gravitates to surface structure (19.0% vs 2.8%). The concerns only humans raise (unwarranted, stale, and hallucinated content) are almost never caught by AI.

RQ4 (Recovery): What share of human concerns can AI reviewers recover? Finding: Against a benchmark of 1,962 verified concerns across 468 PRs, three widely-used agents (Codex, OpenCode, and Claude Code) recover only **1.3–3.9%**. Each agent received the repository at review time, so a miss cannot be blamed on missing context. They miss precisely the judgment-dependent categories that humans raise. Because the benchmark captures exactly what AI misses, it is the grounding a future reviewer for these files would need; we sketch such a tool as future work.

This paper makes the following contributions.

- The first empirical census of how agent instruction files are reviewed (2,870 PRs), showing that review is largely absent and, where present, superficial;
- The first taxonomy of instruction-file review concerns (16 categories), which exposes that AI reviewers are structurally blind to the judgment-dependent concerns humans raise; and
- A benchmark of 1,962 manual validated concerns showing that even the strongest current agents recover only 1.3–3.9%, evidence that AI cannot substitute for human attention on reviewing agent instruction files.

II. Background and Related Work

A. Agent Instruction Files

Content and adoption. Prior work taxonomizes instruction file content across Agent READMEs, `CLAUDE.md`, Cursor rules, and agent manifests [7]–[11]. Parallel work studies prompt defects and system-prompt use [12]. Galster et al. report that 90.6% of agentic-tool-configured repositories ship instruction files [1]. Robbes et al. estimate 22–29% coding-agent adoption across 128k projects [13]. Arabat and Sayagh [14] treat instruction files as software artifacts that should be reviewed, measuring merge rate, code churn, and the number of review comments each PR draws.

Quality, security, and governance. Instruction file quality matters because defects alter what a non-deterministic agent does. Context length and prompt formatting influence model behavior [15], [16]. Gloaguen et al. [6] find that repository-level context files do not significantly improve task-success rates on SWE-bench or their own CTXBENCH benchmark. They also raise inference cost by over 20%, and LLM-generated files sometimes reduce success outright. Instructions also pose prompt-injection risks in agentic editors [17]. Security-smell work on infrastructure configuration provides the closest analogy for systematically reviewing privileged instructions [18]. Researchers studying prompt defects and governance frame these files as policy-bearing artifacts with specification, structure, and community-governance risks [12], [19].

B. Reviewing Software Artifacts

Code review. Classic studies sort review comments into functional defects and maintainability issues such as readability, naming, and structure. They find that maintainability concerns dominate, accounting for roughly 75% of comments [20], [21]. Turzo and Bosu [22] built a taxonomy of review comments. Other work studies why reviewers comment, which comments lead authors to make changes, and how review transfers knowledge between developers [23]–[25].

Documentation and configuration review. Previous work on documentation reviews studies issue types, outdated references, installation guidance, documentation-review comments, and README linting [26]–[30]. Work on configuration distinguishes two failure modes we borrow. *Misconfiguration* means wrong parameter values that break the system; Xu et al. argue this is a design problem rather than a user problem [31]. *Configuration smells* are maintainability and security anti-patterns in configuration scripts that a review can catch even when the values themselves are valid [32]–[34].

Substantive review. Another strand of studies focuses on the rubber-stamp effect of reviews: changes are approved without scrutiny. Look good to me (LGTM) review smell studies report 64.7% comment-free rate and 18% LGTM rate in pull requests [35], [36]. Similar work quantifies a shallow review rate of 14% [37]. Bot participation and differences between human and LLM comment categories provide additional baselines [38], [39].

AI reviewers and benchmarks. Both industry and academia have invested in automated review generation since the pre-LLM era [40]–[44]. GitHub reports 60M Copilot reviews in production [44], a vendor-reported scale signal. Evaluations of automated code review have moved from text-similarity metrics toward recall against human review comments [45]–[50].

C. The Knowledge Gap

Instruction files now steer much of what software agents do, and they are widely adopted and consequential. Yet they lack strong validation; review is the main quality gate for each change. Researchers have studied code review, documentation review, and configuration review extensively, but not changes to instruction files. These files have no per-change oracle, and no prior work provides the comment-level census, taxonomy, or AI benchmark we build here.

III. Methodology

Instruction files have no per-change oracle, and no existing dataset separates instruction-file review from ordinary code review. A large-scale, hand-validated census of real pull requests is therefore the only practical way to see how these files are reviewed. Surveys would burden already-overloaded maintainers and still miss scale; case studies risk project-specific bias and cannot capture the broader landscape. We mine the whole of GitHub and reconstruct the review lineage for thousands of agent-instruction-file PRs. The study has four stages (Figure 1): **(1) acquire** a census of reviewed instruction file PRs; **(2) classify** every actor as human or AI; **(3) hand-code** each relevant review act; and **(4) validate and benchmark** the human concerns against product-native AI reviewers.

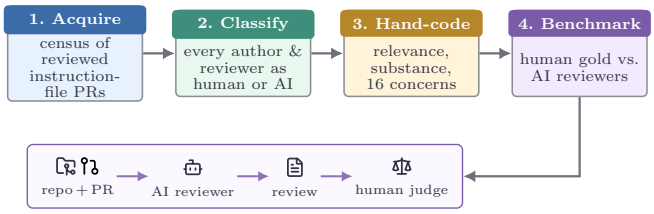

Fig. 1. Overview of the study pipeline.

TABLE I
DEMOGRAPHICS OF THE CORPUS REPOSITORIES.

Statistics	Mean	Median	Distribution
# of Stars	8,565.83	1,426.00	
# of Commits	8,738.32	2,198.50	
# of Contributors	246.27	56.00	
Age (years)	5.99	5.26	
Language	TS (27.4%), Py (18.7%), Go (8.4%), Other (45.5%)		
Owner type	Org (1,706), User (540)		

A. Data acquisition & the corpus

We assemble the corpus in three stages. **(1) Candidate discovery.** We first query the April 2026 snapshot of World of Code [51] for commits whose changed paths match our instruction-file patterns (≈460k commits). We then map these commits to GitHub pull requests using GHArchive event data [52] and retain only candidates from repositories with more than 100 stars, yielding 3,765 candidate PRs. **(2) PR retrieval.** Archival datasets contain noise and hollowing [53], so we cross-validate each candidate against the GitHub API. We keep only pull requests whose full metadata, changed files, reviews, and comments can be recovered. **(3) Static snapshot.** For each verified PR, we compile a static snapshot: PR metadata, changed files, review and discussion comments, review records, and the base and head versions of each changed instruction file. A PR enters the corpus only if it (i) modifies at least one instruction file, (ii) contains at least one non-author review comment in the recovered GitHub data, and (iii) can be consistently reconstructed at the reviewed revisions. The corpus contains 2,870 verified instruction-file PRs from 2,246 repositories. These PRs yield 5,475 instruction-file instances, where each instance is one changed instruction file within one PR.

Table I summarizes the 2,246 corpus repositories. The four metrics are heavily skewed, so we report mean and median and visualize each distribution.

B. Actor classification — author & reviewer

Before measuring anything else, we label every actor in the corpus.

Author origin (RQ1). For each change, we infer its author origin from structured signals in the PR artifact and from textual evidence in the PR description and discussion. We distinguish four mutually exclusive origins: (i) *pure-human*, (ii) *AI-generated-declared*, (iii) *AI-agent/bot*, and (iv) *self-admitted-AI-assisted*. AI-generated-declared and AI-agent/bot changes carry unambiguous machine-emitted markers in the PR metadata, specifically fixed co-author lines or generated-by trailers from tools such as Claude Code, Copilot, or Codex. Self-admitted-AI-assisted changes come from free-text self-reports in the PR description or comments (e.g., "vibe-coded" or "with help of …"). These self-reports are noisier but still explicit. Changes that show none of the above signals are labeled pure-human. To validate the labeling procedure, we manually audited a stratified random sample of 150 PRs. We found a missed-AI-attribution rate of 2.0% (3 errors). Because our method relies on explicit declarations and may miss unreported AI involvement, the AI-authored share we report is a conservative lower bound.

Reviewer role (RQ2–RQ4). We automatically classify each reviewer account as Human, AI, or Tool based on signals drawn from public account metadata. We exclude reviewer accounts whose self-reported status says they skipped or failed the review. We validate the classifier against a manually curated gold standard of 300 stratified reviewer accounts, independently coded by two annotators (Cohen's $\kappa = 0.983$). The classifier achieves 95.3% accuracy under account-level weighting, where each unique reviewer account contributes once. It achieves 98.5% accuracy under PR-instance-level weighting, where each appearance of a reviewer in a PR is counted separately.

C. Human coding

Relevance (automated + human). A comment is relevant if it addresses a changed instruction artifact. We first apply two deterministic rules. Together they cover 91.8% of all non-author review comments (21,710 of 23,651). Rule (1): an inline review comment left *on the changed instruction file*, such as a diff-anchored suggestion on a specific line. Rule (2): a bare approval in a *pure-instruction* PR that changes only instruction files, such as an "LGTM" with no inline comments at all. We also exclude empty bodies and status-only bot noise. The remaining 8.2% of comments are hand-coded and adjudicated. Inter-coder agreement was moderate ($\kappa = 0.630$), reflecting the inherent ambiguity of relevance at the boundary between instruction-file and non-instruction-file discussion. Aggregating to the PR level, the funnel over all 2,870 PRs is: 721 with no effective non-author review (no human or AI reviewer left a comment after we exclude bot skip/fail messages), 949 reviewed but not relevant, and 1,200 relevantly reviewed (Figure 2).

Substantive & concern coding (inductive thematic analysis). From the 1,200 relevantly-reviewed PRs, a single coding scheme yields both the concern taxonomy and a substantive/non-substantive label. A comment is substantive iff the coding attaches at least one concern, i.e., a genuine evaluative point about the change's content, placement, naming, or governance. Two authors inductively open-coded approximately 300 PRs, iterated to a stable codebook, and

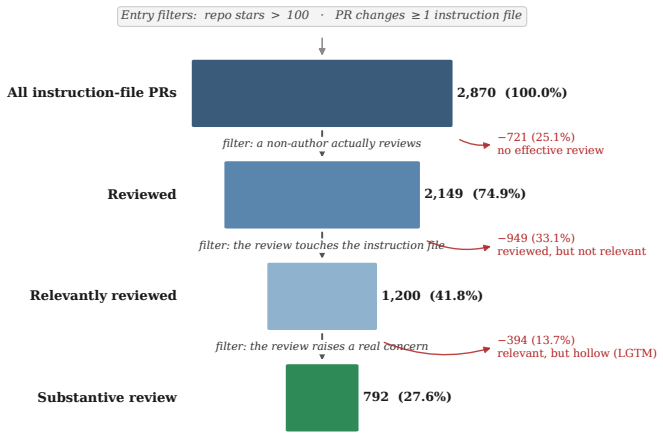

Fig. 2. The review funnel over 2,870 instruction-file PRs.

then double-coded 99.3% of the 1,200 PRs. The substantive-vs-non-substantive *tier-gate*, which marks whether a comment carries any concern, reached $\kappa = .918$. Per-concern code assignments agree at a macro-average $\kappa = .72$; most codes fall between .71 and .89, and a few judgment-heavy codes dip lower. Disagreements were adjudicated to gold, yielding 1,987 substantive comments across the 1,200 PRs.

Identity-blind coding. The substantive/concern pass is identity-blind. Coders never see reviewer login, class, or mix; they see only the instruction file diff and comment texts. Reviewer identity is joined back only after labels are frozen.

D. Validation & evaluation

Validate comments → the gold. The 1,987 substantive comments feed this step. Two coders validate each concern (KEEP/DROP/UNSURE) by reading the PR, inspecting the current repository, and searching the web. They judge each concern against three tiers of evidence, from strongest to weakest: *outcome* (did the author later change the flagged content?), *discourse* (a "good catch" or "fixed" reply vs. a correct author rebuttal), and *factual truth* (is the claim itself correct, checked against the code and docs?). Double-coding and adjudication trim 25 confirmed false positives (1.26% of the 1,987 substantive comments). The result is a gold set of 1,962 concerns across 468 PRs.

Benchmark (reviewer execution & coverage judging). We reconstruct each PR at its review-time state and invoke each product's native review on the full repository (base and head) with the archived PR title and body. The three subjects are Codex `review` (GPT-5.5 high), OpenCode `review` (DeepSeek-v4-pro max), and Claude Code `/review` (Claude Sonnet 4.6 high). We run each system on 446 of the 468 gold PRs that could be reconstructed at review time; the remaining 12 PRs were unreconstructable. We disabled web tools and trimmed git history to prevent leakage.

TABLE II
AUTHOR-ORIGIN CLASSES.

Author origin	PRs	of 2,870
Pure human	1,907	66.4%
AI-generated (declared)	403	14.0%
Human, self-reported AI use	192	6.7%
AI agent / bot	368	12.8%

We measure coverage as strict nugget-recall against the gold: each gold concern is a nugget, and we report what fraction the AI review surfaces. **Two human judges** independently judge each AI review. A `covered` verdict requires the judge to supply a contiguous verbatim quote from the AI review that raises the *same concern in the same direction* as the gold concern. A match counts even when the AI's wording or proposed fix differs from the gold; praise or neutral description does not count.

IV. RQ1: Origin

Before examining how instruction-file changes are reviewed, we characterize the changes themselves: who writes them, whether they travel alone or with other work, and whether they create a file or edit an existing one. These properties shape the task that later review faces. The full census covers 2,870 PRs across 2,246 repositories and contains 5,475 instruction-file instances. The unit of analysis throughout is the PR.

Authorship. We classify each PR by the origin of its instruction-file changes into four mutually exclusive author-origin classes, following the detection method of Section III. Roughly one third are explicitly generated or assisted by AI. Table II shows the breakdown: **963** of 2,870 PRs fall into one of the three AI-associated classes, while the rest are pure human. These counts capture only cases with *explicit* AI evidence (a self-declaration, a machine-emitted trailer, or a bot author account). Undisclosed AI use is not counted, so the true AI share is likely higher.

Authorship tells us who writes the changes. We now ask how they enter the repository: alone, or bundled with other work that competes for reviewer attention.

Ride-along. Instruction changes rarely arrive alone. We call a PR a *rider* if it changes at least one non-instruction file and *standalone* otherwise. **1,979 (69.0%)** are riders. Riders carry a median of 4 other files. The distribution has a heavy right tail: a handful of mega-PRs sweep up thousands of vendored, generated, or lockfile entries, and the largest touches 1.34M lines. Among riders, the instruction edit most often travels with code (68.7%), then docs (58.2%) and config (51.0%).

Change type. The dominant change is adding a new instruction file, not editing an existing one. By instances, 72.4% of the 5,475 instruction-file occurrences are added as new. By PRs, **1,925 (67.1%)** are *pure-add*: they only add instruction content, with no deletions to existing instructions. Most instruction files enter the repository as a brand-new addition. The reviewer has

TABLE III
CO-CHANGED FILE TYPES IN RIDER PRS.

type	PRs	share	type	PRs	share
code	1,360	68.7%	build/deps	778	39.3%
docs	1,152	58.2%	test	776	39.2%
config	1,010	51.0%	CI/workflow	503	25.4%
other	802	40.5%	lockfile	499	25.2%

TABLE IV
REVIEWER MIX ACROSS THE TWO REVIEW GATES.

Reviewer mix	PRs	Relevant (n/N)	Substantive (n/N)
silent	717	0/717	—
human-only	882	411/882	242/411
ai-only	484	337/484	194/337
human+ai	530	446/530	353/446
tool-only	257	6/257	3/6
Total	2,870	1,200/2,870	792/1,200

no prior version to diff against and must evaluate the content on its own terms.

Summary

Roughly a third of instruction-file changes are AI-authored (963/2,870), 69% ride along inside larger code PRs rather than arriving alone, and 67% add a brand-new file with no prior version to diff against, three conditions that together leave the reviewer no baseline to judge the change against.

V. RQ2: COVERAGE

RQ2 asks three questions: who reviews instruction-file changes, whether any review reaches the change, and whether that review offers a real evaluative judgment. We call the last failure *review hollowing*: review is present but never engages with or evaluates the instruction-file change.

A. Who reviews the changes

Table IV shows the reviewer mix for the 2,870 instruction-file PRs (classification in Section III).

AI both writes and reviews. AI authors about one-third of these changes (RQ1) and reviews 1,014 of the 2,870 PRs (35.3%; the ai-only and human+ai splits appear in Table IV). These two roles overlap. Among the 2,675 PRs jointly covered by the authorship and reviewer analyses (93% of the census), 417 (15.6%) are both AI-written and AI-reviewed. In 148 (5.5%), an AI reviewer is the only reviewer present, with no human in the loop; we examine that configuration in RQ4.

B. Relevance: did any review touch the instruction change?

The first gate asks whether any non-author comment touched the instruction-file change. The relevance screen splits the 2,870 PRs into three groups: 721 with no effective non-author review, 949 reviewed-but-not-relevant, and 1,200 relevantly reviewed. Overall, 58.2% of instruction-file PRs receive no relevant comment at all, and only 1,200 (41.8%) are relevantly reviewed. AI reviewers reach the instruction file more often than humans do, and human+ai pairs reach it most often (Table IV). Silent and tool-only PRs draw essentially no relevant comments.

Reviewed-but-not-relevant is a failure mode prior work cannot see. The 949 PRs (33.1%) draw review on the PR's code but not on its instruction file. Prior shallow-review work would count these simply as "reviewed", because it tracks whether a PR received any review, not whether that review touched the instruction file. This matches RQ1: many instruction edits ride along on code-centric PRs, so reviewers focus on the code and ignore the instruction change.

C. Substantive: did the review raise a real concern?

We now examine individual comments. Reviewers left 4,789 comments on the 1,200 relevant PRs. We classify each comment as *substantive* if it raises at least one evaluative concern, or *hollow* if it touches the file without rendering an evaluative judgment. Overall, 66.9% are substantive and 33.1% are hollow.

The human/AI divergence. Humans are substantive **73.9%** of the time; AI reviewers are substantive only **57.4%**. The two also hollow out in different ways.

The two hollow failure modes are qualitatively distinct. Human hollow review is usually a rubber-stamp approval (18.9% of human comments), an "LGTM"-style sign-off. AI hollow review is almost always a walkthrough / PR-summary (39.5%). It looks like review, a tidy file-by-file recap, but renders no evaluative judgment.

The review funnel. Combining both gates at the PR level gives the funnel in Figure 2. Across all 2,870 instruction-file PRs, 71.9% receive no substantive review of the instruction change; only 792/2,870 (27.6%) do.

Summary

Only 41.8% (1,200/2,870) of instruction-file changes draw a comment that even touches the instruction file, and just 27.6% (792) draw a substantive one; over 70% clear the gate with no real evaluation. Most of the shortfall is silence or code-only review; even among reviews that do reach the file, a third of the comments are hollow "LGTM" or walkthrough sign-offs that render no judgment.

VI. RQ3: DIVERGENCE

We hand-code the substantive review comments and identify **16 concerns** (Table V), the quality issues a reviewer raises about an instruction-file change. Human and AI reviewers distribute across these concerns very differently. Instruction files have *no per-change oracle*: no automated ground truth that says whether a given directive is true, necessary, current, or safe. Because nothing forces reviewers to converge on a single "should catch" list, human and AI reviewers act as non-redundant checkers with systematically different attention patterns. Humans flag what only a project insider can judge

TABLE V
THE INSTRUCTION-FILE REVIEW-CONCERN TAXONOMY.

Concern	What the reviewer questions	Example	n	% of human	% of AI
Presentation					
Structure / formatting	Spelling, grammar, Markdown, or layout.	AI at `primer/react#6071`: "there's a typo in 'multpile'; it should be 'multiple'."	489	2.8	**19.0**
Correctness					
Executability	A command, path, link, or reference will not resolve or run as written.	AI at `tambo-ai/tambo#1116`: "Read Docs/AGENTS.md" uses the wrong casing and is not a valid link.	311	5.1	**8.2**
Consistency	Conflicts with another rule, a named in-project source, or its own example.	AI at `k3nsei/ng-in-viewport#1556`: "update the Node.js requirement… to match `package.json` engines."	359	4.5	**11.6**
Factual accuracy	A stated fact about the project, toolchain, versions, or workflow is wrong as written.	Human at `openfoodfacts/openfoodfacts-server#12271`: "the `checks` target runs a full frontend build and frontend lint, too"—correcting what the file claims the command does.	262	**7.8**	2.5
Stale / outdated	Once valid, the content has aged out as versions, APIs, or practices changed.	Human at `decidim/decidim#15187`: "this shouldn't happen anymore after the rbenv introduction."	78	**2.8**	0.2
Hallucinated / fabricated	Refers to a file, command, or practice that never existed here.	Human at `PrivateBin/PrivateBin#1698`: "the repo doesn't contain a public directory—I assume this got picked up from our setup hardening instructions?"	46	**1.5**	0.3
Coverage					
Conciseness	Valid content that is duplicated or verbose.	AI at `kornia/kornia#3620`: the section "duplicates the development setup and task list already documented in CONTRIBUTING.md."	103	**2.5**	1.8
Specificity / actionability	Present but too vague or underspecified for an agent to act on.	Human at `dotnet/diagnostics#5609`: "is 'Native component issue' a concept you recognize? … could you give an example?"	201	**4.6**	3.6
Completeness	A rule, command, prerequisite, or example the agent needs is missing.	Human at `OWASP/mastg#3447`: "list prerequisites needed to execute or evaluate the test… existing files are in `prerequisites/`."	775	**24.6**	5.4
Efficacy					
Discoverability / loading	Whether the file or rule will actually be found, loaded, or scoped by the agent tooling.	Human at `Pipelex/pipelex#105`: "please make this an on-demand rule, not `alwaysApply`."	140	**3.2**	2.5
Wrong mechanism	The behavior should be a hook, linter, or CI check, not natural-language prose.	Human at `langchain-ai/langchain#32075`: "let's avoid using copilot as a replacement for a linter… the linter is already doing this."	23	**0.8**	0.1
Security	Secrets, over-broad permissions, prompt injection, or other unsafe agent behavior.	Human at `langfuse/langfuse-python#1268`: "we are allowing all `cat` commands… claude might decide to read `.env`… and send to LLM."	79	1.6	**1.8**
Warrant					
Scope / placement	Fine content in the wrong file, directory, or layer.	Human at `buildbuddy-io/buildbuddy#9822`: put general guidance "in CONTRIBUTING.md instead of adding a bunch of new files specific to only some IDEs."	111	**3.6**	0.6
Audience fit	May be true, but aimed at the wrong reader—a human maintainer, not the agent.	Human at `dashpay/dash#6741`: "it's not for Claude, it's for the maintainer."	39	**1.2**	0.3
Triviality / low-value	Has no actionable value on its own—obvious or empty.	Human at `jitsi/jitsi-meet#16459`: "I don't think this does anything useful, it just lists dependencies."	159	**5.6**	0.4
Overreach / unwarranted	Whether the rule or file should exist at all, even if accurate and well formed.	Human at `quarkusio/quarkus#53038`: "are we *certain* we want AIs to be using `onlyIf`?"	109	**3.9**	0.2

(whether a directive is true, complete, warranted, or out of date), while AI reviewers disproportionately stay at the surface of formatting and embedded-code syntax. That contrast is the paper's central empirical claim.

A. The concern taxonomy

We code each substantive comment into 16 concerns. Table V groups them by the kind of quality questioned and orders them from surface-level, machine-checkable concerns to context-dependent, judgment-heavy ones. The grouping is expository, not a coding axis; a comment may carry up to two concerns. Most of the taxonomy is shared with code and documentation review. Five concerns have no equivalent in prior code-, documentation-, or prompt-review taxonomies: warrant (*Triviality*, *Overreach*), agent efficacy (*Audience fit*, *Discoverability / loading*), and a new AI-authorship subcategory of correctness (*Hallucinated*). These five are the taxonomy's distinctive contribution. The percentage columns show each concern's share *within that reviewer type's own comments* (2,687 human / 2,042 AI).

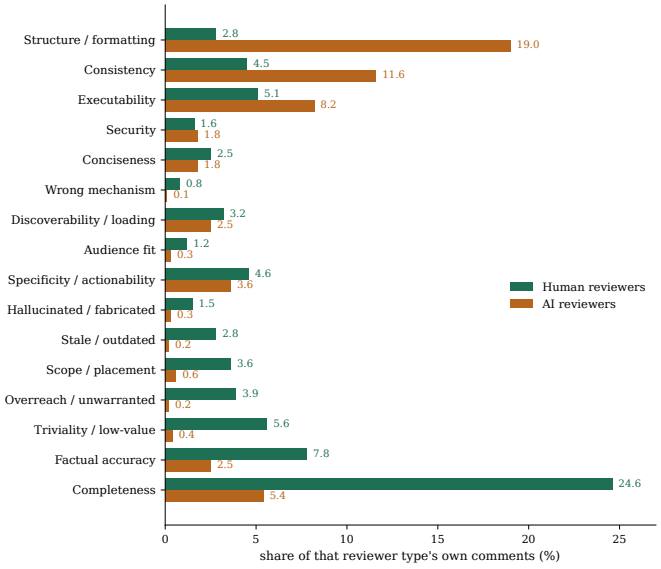

Fig. 3. Each concern's share of human vs. AI substantive comments.

B. The human/AI divergence

The distribution reveals the divergence immediately. The three largest concerns are Completeness (n=775), Structure (n=489), and Consistency (n=359); together they account for the bulk of substantive comments. The long tail contains the judgment and AI-risk concerns: Triviality, Overreach, Stale, Hallucinated, Audience fit, and Wrong-mechanism. These appear rarely, but almost entirely in human comments. Their rarity does not make them unimportant; it means no reviewer or tool is currently raising them.

Human comments are substantive **73.9%** of the time and AI comments **57.4%** (the RQ2 split). But *which* concerns each side raises diverges sharply (Table V and Figure 3). The concerns fall into three clusters:

AI leads on the machine-checkable surface. Structure / formatting (19.0% of AI's comments vs. 2.8% of humans'), Consistency with a declared spec (11.6 vs. 4.5), and Executability / broken references (8.2 vs. 5.1) can all be pattern-matched or run to verify.

Humans lead on the judgments that lack an oracle. They catch what is missing (Completeness, 24.6 vs. 5.4), whether it is true (Factual accuracy, 7.8 vs. 2.5), whether it adds value (Triviality), whether it oversteps (Overreach), whether it has gone stale (Stale), whether the AI invented it (Hallucinated), and whether the placement, audience, or mechanism is right. These judgments require knowing the project, the intent, and the domain; no automatic checker can make them.

Security, Discoverability, Specificity, and Conciseness form a shared band. Both sides raise boundaries, loading, clarity, and cost at similar rates.

The human is the backstop. The gap is concrete, not hypothetical: when humans and AI review the same file, humans supply the judgments AI misses. Among the 446 PRs reviewed by both, 139 contained an AI-authored instruction file. In **78%** of those (109/139) the AI review missed at least one substantive concern that a human then caught. Humans also overrode the AI outright in 13 PRs, making the Overreach and Triviality judgments that AI does not make. On these files AI review is advisory, not authoritative.

Attention is not capability. The patterns above describe what AI *does* raise in the wild, not what a *dedicated* AI *could* raise. RQ4 turns that distinction into a capability test.

Summary

Human and AI reviewers check different things, not the same things at different rates. AI concentrates on surface structure: formatting (19.0% of its comments vs. 2.8% of humans'), spec consistency, and broken references, while humans dominate the judgments that lack an oracle: what is missing (Completeness, 24.6% vs. 5.4%), untrue, stale, or unwarranted. Where both review the same AI-authored file, humans catch a concern the AI missed 78% of the time (109/139); AI judges well-formedness, but whether a rule should exist is invisible to it.

VII. RQ4: Recovery

RQ3's observational distribution cannot settle what AI reviewers *can* raise. We therefore test whether a dedicated AI reviewer, given the same change and full repository, reproduces a human concern (benchmark mechanism in Section III).

A. How much do they recover?

We score the three reviewers against the 1,912 expert-validated human gold concerns (446 PRs); two human judges scored coverage (Section III).

OpenCode recovers 3.9% (75/1,912), Claude 3.2% (62/1,912), and Codex 1.3% (25/1,912); every reviewer misses **≈96–99%**. Coverage is scored strictly—the same concern in the same direction, backed by a verbatim quote (nugget recall, Section III)—and the ranking is stable under a stricter independent re-judging (3.5%, 2.9%, 1.2%).

Table VI gives per-category coverage of the gold, grouped from surface-level, machine-checkable concerns (top) to context-dependent, judgment-heavy ones (bottom).

Even a perfect three-way ensemble reaches only 6.9% (132/1,912); the products miss largely *different* concerns (triple-intersection = 3 concerns), and 77.8% of the 446 PRs (347) receive zero coverage from all three, so **the ceiling is a property of the *task*, not of any one model.**

B. Where the blindness falls

The pattern in Table VI is a floor drop as the concerns become more judgment-heavy. Reviewers reach 5–7% on what current model knowledge can *verify* (stale facts, executable errors, factual mistakes), but the judgment-heavy Warrant category at the bottom is **a near-total blind spot** (best 2.2%, OpenCode; the others are below 2%). The dividing axis is not hard-versus-easy; it is "checkable against an external ground truth" versus "requires a normative judgment about whether the instruction should exist." This is the capability leg of RQ3's

TABLE VI
PER-CATEGORY COVERAGE OF THE 1,912-CONCERN HUMAN GOLD.

Concern category	N	OpenCode	Claude Code	Codex
Presentation				
Structure / formatting	76	**2**	0	0
Correctness				
Executability	134	6	**9**	**9**
Consistency	111	**8**	2	0
Factual accuracy	199	8	**10**	6
Stale / outdated	69	**5**	4	4
Hallucinated / fabricated	38	0	0	0
Coverage				
Conciseness	64	**4**	1	0
Specificity / actionability	123	4	**5**	0
Completeness	648	**27**	22	5
Efficacy				
Discoverability / loading	79	3	**4**	1
Wrong mechanism	20	**1**	0	0
Security	38	**1**	**1**	0
Warrant				
Scope / placement	94	**3**	0	0
Audience fit	32	0	0	0
Triviality / low-value	146	0	**1**	0
Overreach / unwarranted	98	**5**	**5**	1
Headline (unique concern)	**1,912**	**75 (3.9%)**	**62 (3.2%)**	**25 (1.3%)**

divergence, now confirmed under controlled effort: even a *dedicated* reviewer, given the full repository, is blind precisely where human review is distinctive.

The few hits cluster in Executability, Factual accuracy, and Stale / outdated—AI succeeds mainly when review degenerates into ordinary code-or-fact checking, the competence cross-domain guardrails credit it with [46], [47] (see Discussion).

The misses are consequential, not overcaution. Of the concerns all three reviewers missed, 71% (1,173/1,653 with a recorded outcome) were ones the PR author subsequently acted on—rising to 77% for the human-judgment categories—versus only 55% of the concerns the AI *did* catch. The AI does not preferentially recover the concerns that mattered most; if anything it misses them. (The gold's validation used author action as one well-foundedness signal, so the ≈70% base rate is partly by construction; the load-bearing claim is the missed-71% versus covered-55% contrast.)

C. The ranking is effort, not ceiling

The recall ranking across products tracks **engagement, not capability**: Codex is last because 32% of its reviews (141/446) are under 600 characters, single-sentence dismissals that never engage the instruction content. OpenCode and Claude write substantive, instruction-focused reviews and still land at 3.9% versus 3.2%.

D. When the reviewer is confidently wrong

Low recall would be benign if the reviewers stayed silent where unsure. They do not: a non-trivial share of the misses are not silence but active error, where the reviewer endorses the flaw, recommends the opposite of the human, or fabricates its own concern. Three hand-verified cases illustrate the modes.

False affirmation. On `Azure/azure-sdk-for-go#25394` the instruction file referenced a nonexistent legacy `services/` directory—the exact Hallucinated / fabricated concern the human flagged—yet OpenCode listed it as a positive finding ("Correct folder structure — uses `sdk/` ... rather than the legacy `services/` directory"), praising the file on the very dimension it was wrong.

Contradiction. On `NVIDIA-NeMo/Curator#982` the author's explicit decisions were to focus on Python 3.12 and ignore GPU for now; Codex recommended the opposite on both, keeping 3.10 support and adding the GPU install index.

Fabrication. On `Azure/azure-sdk-for-java#44735` the human confirmed the file was correct (Java 21 is the current LTS), yet both Codex and Claude independently asserted "Java 25 is the latest LTS" and told the author to update—two model families converging on the same invented version.

These are illustrative, hand-verified cases, not a precise rate (Section III). The direction matters: a reviewer that endorses flaws and fabricates facts does not merely fail to help—automated review of this kind would *inflate* a file's apparent quality, a point we take up in the Discussion.

Summary

Three state-of-the-art agents recover only 1.3–3.9% of validated human concerns even with full repository context, a task ceiling, not a model ceiling. The deeper problem is contamination: the reviewer loads the instruction file as its own context and, in several cases, obeys the very rules it is asked to audit.

VIII. DISCUSSION

Our results describe a situation where most instruction files skip through their only quality gate twice. In most of the cases, instruction files don't receive reviews; even in cases they do, the reviews possibly leaving out the significant concerns. This double skip reflects a deeper mismatch between the emerging artifacts and the existing review process.

A. The no-oracle gap as the structural blindness

Instruction files lack an external oracle: unlike code, their correctness cannot be settled by compilers, linters, or test suites. This forces reviewers onto judgment-grounded terrain, where humans raise completeness, factual, and judgment-heavy concerns while AI gravitates toward surface structure (Figure 3). The result reveals a systematical blind spot: AI misses precisely the concerns that make human review indispensable (Section VI). The blindness is structural, not informational. Today's agentic reviewers

Worse, the file under review is self-executing: the design of the current agentic reviewers configures agents towards their review target (head ref of GitHub PRs), possibly introducing

bias and undesirable behaviors. The to-be-reviewed agent instruction file acts as the agent's constitution, and a constitution cannot coherently question itself from an external standpoint. The convergence of independent products on the same blind categories, together with the contamination effects observed in our controlled trial, favors this structural reading over a pure knowledge-gap account (Section VII).

B. AI reviewers on instruction files verses code

Our results points out a major gap in the agents' reviewing capability in instruction files. The three reviewer agents we tested, given full repository context, covered only **1.3%**, **3.2%**, and **3.9%** of validated human concerns, in other words, missing out the majority. The category-level pattern reinforces the interpretation: Warrant concerns were barely touched (best 2.2%), hallucinated or fabricated content was detected 0% of the time, and Triviality / low-value necessity concerns were caught only once out of 146 instances. Staleness emerged as the lone relative strength (6–7%), suggesting that AI reviewers recover what can be checked against explicit, verifiable signals and miss what requires background knowledge typically preserved in developers' brains.

Yet, agents do know how to review code. Comparable evaluations on reviewing source code report far higher recall: single tools in c-CRAB reach 20–32% and a four-tool ensemble reaches 41.5% [54]; Code Review Bench scores the strongest tool at 73–77% [45]; even calibrated against CR-Bench, which sits near 5% precision at high recall [46]. Statistics from industrial tools agree—BitsAI-CR reports ∼75% precision [43], AutoCommenter reports ∼40% resolution ratio [55], and Copilot comments are 71% actionable [44] (though reception is mixed [56] and vendor benchmarks can be self-serving [57]).

Taken together, the evidence indicates that, for agentic reviewers with production-level capabilities, reviewing instruction files remains a distinct and underserved challenge. Whether the gap reflects a fundamental limitation or merely an under-represented target in current LLM training and evaluation is a question for future work; either answer would reshape how we design AI-assisted review.

C. Towards instruction file review automation

The status of current reviews and the performance gap of current agents render reviewing instruction an emerging, unique, and significant workload. Derived from the insights of our study, we sketch three core characteristics for future designs on automating instruction file reviews.

Tuned for instructions. The reviewer should treat instruction files as executable policy, separate from documentation and configurations. That means recognizing defect classes specific to these files: hallucinated rules that do not match the repository, stale constraints inherited from earlier versions, trivial or low-value rules that add noise and cost, and unwarranted rules that over-reach the agent's scope. It also means resisting the structural bias toward surface well-formedness (completeness, structure, and style) that dominates current AI review (Section VI). Training or prompting should be grounded in the human-validated concern taxonomy in Section III, with special attention to the judgment-heavy categories that human reviewers currently monopolize.

Context-aware review. Many of the human-only concerns cannot be judged from the diff alone. Whether a rule is stale, whether a convention is project-specific, and whether a restriction is warranted all require understanding the project's evolution and norms. The reviewer should proactively mine project context (pull-request history, issue discussions, comments, commit messages, and the current repository state) and derive project-specific quality checks from it. For example, a rule that mandates an outdated workflow step can be flagged by cross-referencing recent CI changes. A rule that duplicates an existing convention can be checked against prior agent-instruction PRs. This shifts the reviewer from a pattern matcher to a grounded investigator.

Autonomous with escalation. A useful reviewer should save developer effort without sacrificing quality. Surface defects such as formatting, broken internal links, duplicated rules, or simple factual mismatches can be fixed directly or explained in detailed, actionable comments. Hard-to-judge issues such as necessity, over-reach, hallucination, and project-specific convention should be escalated to a human, but not as a blank request. The agent should first mine the relevant context and then interview the maintainer in a structured way, to elicit the judgment needed for an informed decision rather than hallucinating a verdict. This division of labor matches what we observe: even on AI-authored instruction-file changes, 78% still draw a human in for the final judgment calls (Section V).

IX. Threats to Validity

Construct validity. The 16-concern taxonomy was derived through inductive, identity-blind double-coding of 99.3% of the 1,200 relevantly-reviewed PRs. The substantive tier-gate—whether a comment carries any concern—reached $\kappa = .918$ (Section III); per-concern agreement is lower (macro-average $\kappa = .72$, most codes .71–.89, with a few judgment-heavy codes lower). The judgment-heavy concerns, i.e. Triviality, Overreach, Stale, Hallucinated, rest on few cases, so we anchor them to examples and interpret them cautiously (Section VI); the human/AI divergence, however, is driven by the high-κ, high-N codes (e.g., Completeness, Factual accuracy), not these. Coverage and substance are reported on separate axes, so high AI coverage is never read as high substance (Section V). Finally, the RQ4 benchmark measures *recall* against a human gold: agents earn no credit for valid concerns raised outside it, so our claim is that they fail to *recover human judgment*, not that they add no value; benchmark precision is left to future work.

Internal validity. Every reviewer-stratified result depends on labelling each account human, AI, or tool; the classifier validates against blind-coded gold at 95.3% account-weighted and 98.5% instance-weighted accuracy ($\kappa = .983$) (Section III). Author-origin and the ride-along split are descriptive lower bounds: undisclosed AI use is uncounted, so the ≈1/3

AI-authored share is a floor, not a ceiling (Section III). RQ3's observational gap could also reflect *tasking* rather than *capability*—wild AI accounts may never be asked to review substantively; RQ4 removes this confound by explicitly tasking agents with the full repository, and they still recover only 1.3–3.9% (Section VII).

External validity. The corpus is gated to popular repositories (`star>100`, 2,246 repos) on a single English-dominant platform; the long tail and other forges are excluded. Since review is scarcer there, the 71.9% no-substantive-review rate is a lower bound. Agent instruction files are a moving convention—our snapshot spans February 2021 to April 2026—so we frame the taxonomy and benchmark as exploratory rather than a stable base rate (Section III; Section VI; Section VII).

Reliability and reproducibility. All coding disagreements were adjudicated to a single agreed gold label, so the labels we analyze are reconciled, not disputed. The gold set holds 1,962 validated concerns across 468 PRs (1.26% false positive); the RQ4 benchmark scores the 1,912 concerns across the 446 PRs reconstructable at review time. Coverage is scored by two human judges, and the product ranking is stable under a stricter independent re-judging (Section VII); we treat the point estimates as indicative, not precise, given a *single, nondeterministic* run per product. Two residual threats cut in our favor: both identity-blind *leakage* and *training-cutoff contamination* could only *inflate* recovery, so our figures upper-bound true capability—and RQ4's fabricated-fact cases (e.g., an invented "Java 25 LTS") show agents guessing, not recalling.

X. Conclusion

Agent instruction files are executable policy with no per-change oracle: they slip past compilers, type-checkers, and test suites. Review is their only quality gate. Across 2,870 pull requests from 2,246 repositories, we found that gate is largely broken, with over 70% of instruction-file changes receiving no substantive review. When AI reviewers fill the gap, they are structurally blind to the concerns that most need human judgment: they catch surface structure but miss factual errors, necessity, staleness, and hallucinated content. This is not a generic "AI cannot review" finding; the same agents review source code competently, but their blindness here is specific to executable policy.

We contribute the first taxonomy of instruction-file review concerns (16 categories), which exposes the human/AI divergence, and a benchmark of 1,962 human-validated concerns across 468 PRs against which three widely-used agents recover only 1.3–3.9%. The no-oracle gap drives this divergence: without ground truth for correctness, human and AI reviewers attend to non-overlapping concerns, and because an agent-native reviewer loads the file as its own context, the artifact under review can shape the very agent that audits it. A constitution cannot coherently question itself. An instruction-file change is a change to agent behavior, not a documentation edit. Whether the gap reflects a fundamental limitation or an under-represented training target remains open; either answer would reshape how we design review for executable policy.

References

[1] M. Galster, S. Mohsenimofidi, J. L. Lulla, M. A. Abubakar, C. Treude, and S. Baltes, "Configuring agentic ai coding tools: An exploratory study," in *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26)*, 2026, adoption: 90.6% of 2,853 agentic-tool-configured repositories ship a Markdown context/instruction file.

[2] diggerhq/digger contributors, "Opentaco POC," GitHub pull request `diggerhq/digger#2110`, https://github.com/diggerhq/digger/pull/2110, 2025, opened 2025-08-23; accessed 2026-06-30.

[3] keycloak/keycloak contributors, "AI migration tool—command with rules, based on the specs generated by the spec-kit with manually clarified migration patterns," GitHub pull request `keycloak/keycloak#47024`, https://github.com/keycloak/keycloak/pull/47024, 2026, opened 2026-03-10; accessed 2026-06-30.

[4] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[5] S. Schulhoff *et al.*, "The prompt report: A systematic survey of prompt engineering techniques," 2024.

[6] T. Gloaguen, N. Mündler, M. N. Mueller, V. Raychev, and M. Vechev, "Evaluating agents.md: Are repository-level context files helpful for coding agents?" in *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems (MemAgents)*, 2026, oral presentation; verify final venue metadata before camera-ready.

[7] W. Chatlatanagulchai *et al.*, "Agent readmes: An empirical study of context files for agentic coding," 2025.

[8] ——, "On the use of agentic coding manifests: An empirical study of claude code," in *Product-Focused Software Process Improvement*. Springer Nature Switzerland, 2025, pp. 543–551.

[9] S. Mohsenimofidi, M. Galster, C. Treude, and S. Baltes, "Context engineering for ai agents in open-source software," 2025.

[10] S. Jiang and D. Nam, "Beyond the prompt: An empirical study of cursor rules," in *Proceedings of the 23rd International Conference on Mining Software Repositories*, 2026.

[11] H. V. F. Santos, V. Costa, J. E. Montandon, and M. T. Valente, "Decoding the configuration of ai coding agents: Insights from claude code projects," 2025.

[12] H. Tian, C. Wang, B. Yang, L. Zhang, and Y. Liu, "A taxonomy of prompt defects in llm systems," 2025.

[13] R. Robbes, T. Matricon, T. Degueule, A. Hora, and S. Zacchiroli, "Agentic much? adoption of coding agents on github," 2026, arXiv preprint; recent work.

[14] A. Arabat and M. Sayagh, "Toward instructions-as-code: Understanding the impact of instruction files on agentic pull requests," in *Proceedings of the 23rd International Conference on Mining Software Repositories, Mining Challenge Track*, 2026.

[15] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, "Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting," in *The Twelfth International Conference on Learning Representations*, 2024.

[16] N. F. Liu *et al.*, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[17] Y. Liu, Y. Zhao, Y. Lyu, T. Zhang, H. Wang, and D. Lo, ""your ai, my shell": Demystifying prompt injection attacks on agentic ai coding editors," 2025.

[18] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *2019 IEEE/ACM 41st International Conference on Software Engineering*. IEEE, 2019, pp. 164–175.

[19] W. Yang, R. He, and M. Zhou, "Beyond banning ai: A first look at genai governance in open source software communities," 2026.

[20] M. V. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 430–448, 2009.

[21] M. Beller, A. Bacchelli, A. Zaidman, and E. Jürgens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 202–211.

[22] A. K. Turzo and A. Bosu, "What makes a code review useful to opendev developers? an empirical investigation," *Empirical Software Engineering*, vol. 29, no. 1, p. 6, 2024.

[23] Z. Li, Y. Yu, G. Yin, T. Wang, Q. Fan, and H. Wang, "Automatic classification of review comments in pull-based development model," in *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering*, 2017, pp. 572–577.

[24] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering*. IEEE, 2013, pp. 712–721.

[25] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.

[26] E. Aghajani *et al.*, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering*. IEEE, 2019, pp. 1199–1210.

[27] W. S. Tan, M. Wagner, and C. Treude, "Detecting outdated code element references in software repository documentation," *Empirical Software Engineering*, vol. 29, no. 1, p. 5, 2024.

[28] H. Gao, C. Treude, and M. Zahedi, "Adapting installation instructions in rapidly evolving software ecosystems," *IEEE Transactions on Software Engineering*, vol. 51, no. 4, pp. 1334–1357, 2025.

[29] N. Rao, J. Tsay, M. Hirzel, and V. J. Hellendoorn, "Comments on comments: Where code review and documentation meet," in *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, 2022, pp. 18–22.

[30] H. Mynampaty, N. Josephine, K. E. Isaacs, and A. M. McNutt, "Linting style and substance in readmes," in *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. ACM, 2026, pp. 1–23.

[31] T. Xu *et al.*, "Do not blame users for misconfigurations," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 244–259.

[32] Y. Zhang, H. He, O. Legunsen, S. Li, W. Dong, and T. Xu, "An evolutionary study of configuration design and implementation in cloud systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 2021, pp. 188–200.

[33] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 189–200.

[34] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams, "Security smells in ansible and chef scripts: A replication study," *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 1, pp. 3:1–3:31, 2021.

[35] E. Doğan and E. Tüzün, "Towards a taxonomy of code review smells," *Information and Software Technology*, vol. 142, p. 106737, 2022.

[36] M. F. Gön, B. Yetiştiren, and E. Tüzün, "Towards unmasking lgtm smells in code reviews: A comparative study of comment-free and commented reviews," in *2024 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2024, pp. 163–174.

[37] M. Chouchen *et al.*, "Anti-patterns in modern code review: Symptoms and prevalence," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2021, pp. 531–535.

[38] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Quality gatekeepers: Investigating the effects of code review bots on pull request activities," *Empirical Software Engineering*, vol. 27, no. 5, p. 108, 2022.

[39] S. Goldman *et al.*, "What types of code review comments do developers most frequently resolve?" in *2025 40th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2025, pp. 3760–3765.

[40] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Towards automating code review activities," in *2021 IEEE/ACM 43rd International Conference on Software Engineering*. IEEE, 2021, pp. 163–174.

[41] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "LLaMA-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning," in *2023 IEEE 34th International Symposium on Software Reliability Engineering*. IEEE, 2023, pp. 647–658.

[42] H. Hong and J. Baik, "Retrieval-augmented code review comment generation," 2025.

[43] T. Sun *et al.*, "BitsAI-CR: Automated code review via llm in practice," 2025.

[44] GitHub, "60 million copilot code reviews and counting," 2026, vendor self-reported blog; accessed 2026-06-25.

[45] A. Zverianskii, A. Zhang, J. Clyne, A. Garcia, F. Barez, and S. Upadhyay, "Code review bench," https://github.com/withmartian/code-review-benchmark, 2026, open-source benchmark (Martian); accessed 2026-06-25.

[46] K. Pereira, N. Sinha, R. Ghosh, and D. Dutta, "CR-Bench: Evaluating the real-world utility of ai code review agents," 2026, iCLR 2026 Workshop on Agents in the Wild.

[47] A. Naik, M. Alenius, D. Fried, and C. Rosé, "CRScore: Grounding automated evaluation of code review comments in code claims and smells," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2025, pp. 9049–9076.

[48] J. Lu *et al.*, "DeepCRCEval: Revisiting the evaluation of code review comment generation," in *Fundamental Approaches to Software Engineering*. Springer, 2025, pp. 43–64.

[49] E. M. Voorhees, "Overview of the TREC 2003 question answering track," in *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*. NIST, 2003.

[50] R. Pradeep *et al.*, "The great nugget recall: Automating fact extraction and RAG evaluation with large language models," in *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2025, pp. 180–190.

[51] Y. Ma *et al.*, "World of code: Enabling a research workflow for mining and analyzing the universe of open source VCS data," *Empirical Software Engineering*, vol. 26, no. 2, 2021.

[52] GH Archive, "GH Archive: Public GitHub timeline data," https://www.gharchive.org/, 2026, accessed 2026-07-01.

[53] E. Kalliamvakou, L. Singer, G. Gousios, D. M. German, K. Blincoe, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 92–101.

[54] Y. Zhang, Z. Pan, I. N. B. Yusuf, H. Ruan, R. Shariffdeen, and A. Roychoudhury, "Code review agent benchmark," 2026.

[55] M. Vijayvergiya, G. Petrovic, Z. Chen, M. Salawa, M. Ivankovic, and R. Just, "Ai-assisted assessment of coding practices in modern code review," in *Proceedings of the 1st International Workshop on Large Language Models for Code*. ACM, 2024.

[56] M. Watanabe, Y. Kashiwa, B. Lin, T. Hirao, K. Yamaguchi, and H. Iida, "On the use of chatgpt for code review: Do developers like reviews by chatgpt?" in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2024, pp. 375–380.

[57] DeepSource, "Every ai code review vendor benchmarks itself, and wins," 2026, industry critique; accessed 2026-06-25.