PREDICTING USER-REPORTED PROBLEMS FROM CODING-AGENT TEXT

Anonymous authors
Paper under double-blind review

ABSTRACT

Can a coding agent's prose help predict whether the user's next message will report a problem? We study 34,046 work-blocks from real coding-agent sessions and predict next-user reports that something is broken, erroring, or not working. These reports are only weakly associated with detected execution errors, making them a distinct feedback target. Adding within-block prose to structured context, developer history, and work statistics raises average precision from 0.211 to 0.258. The additive gain persists after excluding blocks with specified lexical cues. In separate user and temporal holdouts, prose alone also outperforms the feature baseline. At a 10% review budget, the combined model captures 35.5% of subsequent reports versus 30.2% for the baseline. The results establish prose as an incremental signal for offline review allocation, without claiming code correctness or intervention benefit.

1 INTRODUCTION

Coding agents act between user messages (Baumann et al., 2026), and developer feedback identifies problems and directs further work (Tang et al., 2026). We ask whether the agent's prose can help anticipate that feedback: will the next user message report something broken, erroring, or not working in the ongoing coding task? We call this outcome a *next-user failure report*. When review capacity is limited, a ranking of blocks by predicted report risk could help a developer decide which work to inspect first. We evaluate that decision offline: how many next-user reports can a ranking capture within a fixed budget of selected blocks? The text comes from the preceding block; the history-feature timing qualifications are detailed in the Method section.

Our unit is a *work-block*: assistant messages and tool activity between consecutive user prompts. A block can contain editing, debugging, inspection, explanation, or planning; its end does not imply task completion. We refer to the assistant text within each block as *agent prose*. We use that text to predict the immediately following user message, such as a report that a test fails or an application still will not start.

Execution-error markers are a natural candidate signal, but they do not consistently coincide with user reports. Across the 34,046 work-blocks we study, 86.6% of reports follow blocks without a detected execution-error marker, while 90.7% of marker-positive blocks receive no report. Research on deployed dialogue systems likewise finds a mismatch between annotated system errors and expressed user dissatisfaction (See & Manning, 2021). We therefore study the next user report as an observed feedback outcome, without treating the report as an independent verdict on code correctness.

Agent prose is an observable output of the work-block: it can describe attempted actions, checks, plans, and claimed results. The agent composes that prose from the work it has just performed, so difficulty, unverified steps, and unresolved issues may be reflected in both the text and the likelihood of a subsequent user report. Those descriptions may contain information that tool counts and other activity summaries omit: for example, whether a check was attempted, what it found, or what remains unresolved. We test whether adding those descriptions improves prediction beyond the structured features. Prior studies forecast user grounding acts from prompts and developer acceptance from behavioral histories (Shaikh et al., 2025; Jiang et al., 2026). Other work improves satisfaction estimation with dialogue text (Bodigutla et al., 2020) and assesses completed agent work using learned detectors and critics (Advani, 2026; Wang et al., 2026). The reviewed comparisons do not isolate whether prose from the preceding agent work-block adds predictive information about the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

next failure report beyond structured summaries of preceding interaction and agent activity. We test that incremental value using prior-feedback summaries, developer history, and work statistics. A separate experiment compares the trained prose predictor with a zero-shot critic reading the same cached prose.

A paired exchange illustrates the target and a ranking disagreement:

> **Agent (excerpt):** "Verify the dev server actually starts — we just pushed the PrimeVue fix but you haven't confirmed it works yet."
> **Next user message:** "Starts, but I am getting a keycloak error of client not found."

The server starts, but the user reports a client-configuration error. In one held-out split, the prose model ranks the preceding block in its top 4%, while the feature baseline ranks it in its bottom 4%. The example illustrates report-risk ranking, not prediction of the specific error or proof that pending-verification language explains the score.

A natural alternative explanation is that the predictor recognizes explicit announcements of trouble. Among observed reports, 64.3% follow blocks with neither a specified error or apology term in the prose nor a detected execution marker. We therefore test prediction after excluding blocks containing the specified prose terms, with the cue definitions and filtering scope detailed in Method and Results.

The paper makes three contributions. First, we characterize next-user failure reports as an observable feedback target distinct from detected execution errors. Second, adding prose to the structured feature baseline raises average precision from 0.211 to 0.258, with an additive gain that persists after lexical exclusion. Separate experiments assess prose-only prediction under user and temporal holdouts and compare it with the tested critic. Third, the combined model captures 35.5% of subsequent reports at a 10% review budget, compared with 30.2% for the feature baseline. Together, these findings support prose as an additional input for offline review allocation and motivate testing whether acting on the ranking improves review outcomes.

2 Related Work

2.1 Predicting user feedback

In coding sessions, PULSE estimates user ratings from user-message, agent-behavior, and git features (Chen et al., 2026). Our task focuses on the immediately following message rather than a session-level rating, using prose from the preceding agent work-block. Developer-acceptance models predict acceptance of code suggestions from behavioral histories and deployment context (Jiang et al., 2026), while Shaikh et al. (2025) forecast subsequent grounding acts from the initial user prompt. These studies motivate accounting for prior interaction when assessing the predictive value of agent prose.

Bodigutla et al. (2020) improve dialogue satisfaction estimation with embeddings of user requests and system responses, and compare that model against one trained on 48 hand-crafted features (correlation 0.79 versus 0.77). Because both sides of the exchange enter as one representation, the comparison does not isolate the system response's own contribution. Deng et al. (2022) use a system-response selection objective, testing a pretraining objective rather than the incremental value of response text. We test whether within-block agent prose adds predictive value for next-user failure reports beyond structured prior-feedback, history, and work features.

2.2 Assessing agent work through language

Advani (2026) detects benchmark false success from agent trajectory text, including on held-out tasks, and compares learned detectors with prompted judges; restricting that input to the closing message performs comparably. We study a different outcome, the next user's failure report, on logged sessions rather than benchmark tasks with known success labels. Difficulty and hedging terms in model reasoning also correlate with answer correctness (Vanhoyweghen et al., 2026), suggesting that language can carry outcome-related information without establishing internal awareness. On the same corpus we study, Kraishan & Jitkajornwanich (2026) find that an agent's self-report refers to roughly one executed action in eleven and that a reader recovers about a fifth of the action log from the report alone. That work establishes prose as an incomplete record of the work, and it does not test whether the record predicts what the user does next.

A rubric-supervised critic evaluates completed agent segments against sparse PR-merge and code-survival outcomes (Wang et al., 2026). That critic learns from segment traces; our separate critic experiment uses a zero-shot prompt on cached prose and does not evaluate the trained trajectory model.

2.3 Allocating limited review

Effort-aware defect prediction ranks source components and measures defects captured within a fixed inspection budget (Guo et al., 2024). Agent monitoring reports detection rates at operating points chosen to limit false-positive escalations (Kale et al., 2026). These studies motivate evaluating prediction under a resource constraint. We measure report capture at a fixed budget of selected work-blocks, rather than a measured inspection-time budget.

3 Task, Data, and Evaluation

3.1 Unit, target, and population

Unit and target. We use logged command-line coding-agent interactions from SWE-chat (Baumann et al., 2026). A work-block contains assistant messages and tool activity between consecutive user prompts.

We define a **next-user failure report** as an immediately following user message that reports something broken, erroring, or not working in the ongoing coding task. The report may concern code, application behavior, or an execution attempt, and may describe a symptom or supply an error log without prescribing a fix.

Operationally, the target is positive when that immediately following message has the corpus-supplied label `failure_report`. "Next" refers to the next user message, not the next failure report found by looking farther ahead in the conversation. Corrections, rejections, takeovers, and other labels are negative for this binary task.

The report is associated with the preceding work-block by conversational adjacency. The label does not establish that the block introduced the problem: the user may be reporting a persistent problem or an unsuccessful attempt to resolve it. We therefore predict an observed report about the ongoing task, not a verified defect caused by the agent. Execution-error markers are a separate measurement (Section 4.1), and absence of a failure report establishes neither correctness nor user satisfaction.

Retained population. The builder joins a following user prompt to the preceding block, so terminal blocks without a following user prompt do not enter the evaluation. A heuristic filter removes prompts matching system-notification patterns and some long formatted or tool-output-like messages. The filtered population contains 34,046 blocks and 2,736 positive labels (8.04%). Eligibility thus depends on the observed follow-up; the results do not estimate failure among all agent work-blocks. The filter can also discard genuine user-pasted errors.

Label reliability. The corpus labels are automated. Blind relabelings yield binary agreement of $\kappa = 0.645$ with an independent language model on 150 stratified turns and $\kappa = 0.650$ with one human coder on 50 stratified turns (88.0% raw agreement). The human check provides no inter-human agreement estimate. Both samples predate the current filter and therefore provide preliminary label checks, not validation of the retained population. Appendix A gives the sampling details and multiclass results.

3.2 Inputs and model comparisons

We distinguish three feature groups. **Context** comprises counts of earlier pushback categories and block position, including log position. Context does not include a semantic encoding of the preceding conversation. **History** comprises a developer report rate and labeled-block count constructed from training rows as described in Section 3.3. **Work statistics** comprise tool counts, action/research-category counts, edited-file counts, a test-command indicator, assistant-message count, message

length, and hedge counts. The *feature baseline* combines all three groups, including the linguistic summary features of message length and hedge counts.

Prose is assistant-message content concatenated within the block and truncated to the first 2,000 characters. Prose is one observable output of the block, not the underlying work process or its result. The representation uses assistant text from within the block, rather than only a final completion message, and excludes the subsequent user message. TF-IDF unigrams and bigrams are reduced to 200 SVD components and supplied to a class-weighted gradient-boosting classifier. Vocabulary and SVD are fitted on each training split. The same classifier settings are used for feature-only and combined models (Appendix B). Missing prose in the main comparisons is replaced by a blank string.

Prose-only versus the feature baseline compares alternative representations. Adding prose to the full feature baseline tests whether language improves a model that already has context, history, and work statistics. We use the combined model for the AP comparison in Section 4.2 and the review allocation in Section 4.4. Adding prose to context and history alone provides a separate ablation against a narrower baseline.

3.3 Timing and historical-feature construction

The intended prediction point is the end of the block, before the next user message. Assistant text and tool features exclude that message, and prior-feedback counts exclude its label. Developer histories are constructed separately within each split after ordering rows by developer, session creation time, and block position.

A training row receives the developer's expanding report rate over preceding training rows. A test row receives the rate and count over all of that developer's training rows; an unseen developer receives the overall training rate and a zero count. Held-out labels are excluded from history construction. This split-level separation does not guarantee chronological availability: in random session splits, training rows can postdate a test block. The temporal regime orders sessions, but overlapping sessions still prevent a guarantee of event-level ordering. The history-bearing comparisons therefore measure offline prediction, rather than a fully validated online feature pipeline.

3.4 Evaluation protocol

The primary comparisons use fifteen 70/30 session-grouped splits. The user holdout uses fifteen analogous splits grouped by developer. The temporal analysis orders sessions by their creation time and cuts at the 55th through 75th percentiles of that session-level ordering in steps of five, so every block of a session falls on one side of the boundary. The models in each comparison are refitted within the corresponding regime.

We report average precision (AP), computed using `average_precision_score`, alongside ROC-AUC. We construct paired intervals by resampling the per-split score differences 5,000 times and taking the 2.5th and 97.5th percentiles of the resampled means. The intervals describe variation in the split comparisons, not independent-population confidence intervals: test sets overlap, and the experiments reuse one corpus. We do not infer significance from overlap or non-overlap of marginal model intervals.

For review allocation, each model ranks the held-out blocks, and we report the share of all reports falling in the top-scoring 5%, 10% and 20% of blocks, together with the share of those selected blocks that precede a report, averaged over the same fifteen session splits. The budget counts blocks rather than reading time.

3.5 Separate critic comparison

We evaluate a zero-shot GPT-5.5 critic on a separate sample of 426 blocks from 343 sessions (250 positive, 176 negative; prevalence 58.7%). The critic rates next-report risk from the cached 2,000-character prose, without a full tool trace. We compare a critic-score logistic model with one that also uses a trained prose reader's score. This tests added value over the specified prompted critic, not a trained trajectory critic with matched supervision. Appendix C gives the prompt, sampling, and training protocol.

4 RESULTS

4.1 REPORTS AND EXECUTION ERRORS ARE WEAKLY ASSOCIATED

How do reports relate to execution errors? Table 1 crosses the next-message report label with the version-2 execution-error marker. The marker detects structured diagnostics in selected execution-tool outputs, including nonzero exits, tracebacks, and test-failure summaries. The execution marker is an unvalidated proxy rather than an objective correctness label (Appendix A).

Of the 2,736 reports, 2,369 (86.6%) follow blocks without detected execution markers. Conversely, 3,573 of 3,940 marker-positive blocks (90.7%) have no next failure report. The binary association is weak ($\phi = 0.017$), so execution markers and next-user reports are not interchangeable labels. A predictor evaluated only against execution markers would be evaluated against a different outcome.

Table 1: Observed execution markers versus next-user failure reports in the main population. Neither label independently establishes correctness.

Execution marker	No report	Report	Total
Absent	27,737	2,369	30,106
Present	3,573	367	3,940
Total	31,310	2,736	34,046

How often are explicit cues present? Among positive reports, 731 preceding blocks contain a specified error or apology term in assistant prose and 367 have an execution marker; 120 have both cues. In total, 978 blocks (35.7%) contain a prose term or an execution marker, while 1,758 (64.3%) contain neither cue. A matched term may concern a resolved or unrelated error, and an unmatched block may still acknowledge trouble in other words. The breakdown therefore does not measure the fraction of reports that repeat the same already-visible problem. The cue counts motivate a separate question in Section 4.3: does prose improve prediction after blocks with the specified prose terms are excluded?

4.2 PROSE ADDS PREDICTIVE VALUE

Prose versus feature summaries. Adding prose to the full feature baseline raises AP from 0.211 to 0.258, a paired gain of 0.047 with a split-bootstrap interval of [0.041, 0.054] (Table 2). Prose alone reaches 0.234, exceeding the feature baseline by 0.023 [0.014, 0.034]. The combined representation therefore has higher mean AP than either representation alone in this comparison.

Prose also improves the narrower context/history baseline (AP 0.178 to 0.261); the full combined model has slightly lower mean AP (0.258). All context inputs here are structured summaries, so the comparisons do not test added value over a semantic reader of the preceding conversation.

Table 2: **Main population, $N = 34{,}046$, prevalence 8.04%**. Means over 15 session splits, with developer histories rebuilt inside each split from training rows only. The full feature baseline adds work and message statistics to context/history. It is not the same comparator as context/history alone.

Representation	AP	ROC-AUC
Context	.190	.692
History	.115	.585
Context + history	.178	.684
Prose	.234	.749
Prose + context + history	.261	.772
Full feature baseline	.211	.733
Prose + full feature baseline	.258	.776

Adding prose to the tested critic. Adding the trained prose score improves prediction over the tested critic score; prose alone performs similarly to the combined model (Table 3). The result is specific to this critic and the separate sample described in Section 3.5.

Table 3: **Separate critic sample, $N = 426$, prevalence 58.7%**; 250 positive and 176 negative blocks. All arms share test rows within each split. AP is not directly comparable with Table 2.

Score	AP	ROC-AUC
Critic-score combiner	.599	.511
Prose reader	.761	.726
Critic + prose combiner	.761	.722

4.3 Robustness to lexical cues, user holdout, and temporal shift

Excluding lexical cues. We remove blocks whose cached prose contains the specified error or apology terms and refit the feature-only, prose-only, and combined models on the remaining 26,814 blocks (prevalence 7.54%). With histories constructed from training rows, adding prose to the feature baseline raises AP from 0.189 to 0.223, a paired gain of 0.034 [0.027, 0.039]. Prose used in place of the baseline reaches 0.196, a difference of 0.006 [−0.001, 0.014] whose interval includes zero. Within the lexically filtered subset, the evidence is stronger for adding prose to the baseline than for using prose alone. Unlike the cue counts in Section 4.1, which use full assistant text, the exclusion experiment filters the cached text window for both positive and negative blocks, then refits the models and rebuilds histories on the subset. The lexical filter does not remove all blocks with execution markers or all semantic admissions of trouble.

Holding out users and later periods. The holdout experiments compare prose alone with the feature baseline (Table 4); the combined model is not evaluated under these holdouts. Prose-only AP exceeds baseline AP by 0.029 for held-out developers and by 0.053 under session-ordered temporal splits, compared with 0.023 in the random session-split reference. ROC-AUC differences range from 0.015 to 0.040. The reported split-bootstrap intervals exclude zero. These results describe within-regime predictive comparisons, subject to the timing qualifications in Section 3.3.

Table 4: Within-regime comparisons against the feature baseline. Histories are rebuilt from training rows within each split. The temporal regime keeps sessions intact. Model AP refers to prose alone except in the final row, which uses prose plus the feature baseline. Intervals bootstrap paired split differences, not independent test populations.

Comparison	Splits	Base AP	Model AP	ΔAP	ΔAP interval	ΔROC [interval]
Prose alone versus feature baseline						
Sessions (reference)	15	.211	.234	.023	[.014, .034]	.016 [.007, .026]
Users held out	15	.180	.209	.029	[.021, .037]	.015 [.007, .024]
Temporal, session level	5	.163	.216	.053	[.040, .068]	.040 [.032, .049]
Lexically filtered subset (session splits)						
Lexical exclusion, prose alone	15	.189	.196	.006	[−.001, .014]	–
Lexical exclusion, prose added	15	.189	.223	.034	[.027, .039]	–

4.4 Prose improves selection under a review budget

Adding prose to the full feature baseline improves recall at every evaluated budget (Figure 1), using the same training-only histories as Section 4.2. At a 10% budget, the selected blocks precede 35.5% of all reports, compared with 30.2% for the feature baseline and 32.9% for prose alone. The paired recall gain over the feature baseline is 0.053 [0.042, 0.067]. At 5% and 20%, the combined model captures 22.2% and 53.9%, respectively, versus 18.3% and 47.6% for the baseline.

Precision among selected blocks rises from 24.4% to 28.7% at the same 10% budget, so 71.3% of selected blocks still have no next failure report. This is a no-report fraction, not a measured false-defect rate.

Read in the other direction, the same rankings answer how much work a reviewer inspects for a fixed share of reports (Table 5). Capturing 30% of held-out reports requires the shortest ranked prefix covering 9.92% of blocks under the feature baseline and 7.74% under the combined model, a

reduction of 2.18 percentage points [1.68, 2.81]. The relative reduction stays near 22% at 20%, 30% and 40% capture. Each budget locates a prefix using the held-out labels, so it describes retrospective selection volume rather than a threshold a deployment could choose in advance.

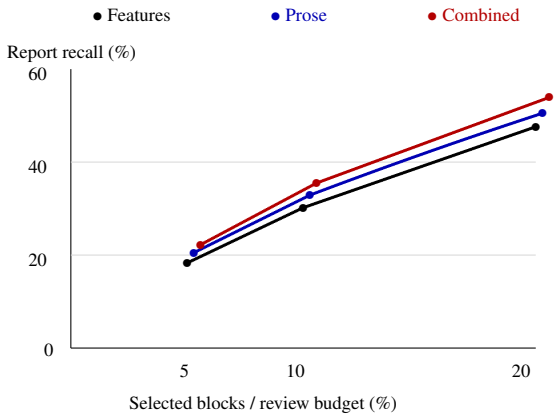


Figure 1: Mean report recall at three evaluated budgets on the main population. Lines connect measured points.

Table 5: Share of held-out blocks that must be inspected to capture a fixed share of reports, averaged over the same fifteen session splits. Intervals bootstrap paired per-split differences.

Report capture	Baseline budget	Combined budget	Reduction (pp) [interval]
20%	5.62%	4.37%	1.26 [0.93, 1.62]
30%	9.92%	7.74%	2.18 [1.68, 2.81]
40%	15.36%	12.10%	3.25 [2.57, 4.10]

5 Discussion

Within-block prose adds predictive value beyond the evaluated structured features and improves offline selection of blocks followed by reports. The open questions concern the source of that information and the benefit of using the ranking in a review workflow.

Report risk and defect risk. A report depends on a perceived problem, an opportunity to notice it, and a decision to communicate it. A repaired execution error may draw no report; a requirement mismatch may draw one without a shell diagnostic. The user may inspect the work independently or encounter a failure elsewhere, so a report need not be a reaction to the prose.

Sources of predictive information. The study assumes neither that prose tone tracks work quality nor that prose causes the user's response. Faulty work can be described confidently, and correct work uncertainly; prose can flag, omit, or misdiagnose a problem. Task difficulty, verification coverage, misunderstanding, and unresolved issues may influence both the prose and the later report. The experiments do not identify which mechanism explains the signal.

Allocation and action. The offline result supports a specific allocation decision: for a fixed number of blocks to inspect, the prose-augmented ranking selects more blocks followed by user reports than the feature baseline. A developer could use such a ranking to prioritize inspection of the recorded agent activity and any associated changes. Acting on such a ranking is a separate question from ranking well: a critic with 0.94 AUROC offline degraded one agent by 26 percentage points when its predictions triggered intervention (Vasudev et al., 2026). That proposed use still requires choosing an action after inspection, such as further verification or another agent pass; each action carries a cost and changes the interaction that follows. A prospective study should fix one action, measure

review time and final work quality against the unflagged workflow, and establish whether the interval between the last agent activity and the user's reply is long enough to act in.

LIMITATIONS

Measurement and selection. Automated report labels were checked on 150 model-relabeled and 50 human-relabeled turns; the single human coder provides no inter-human agreement estimate. Both checks predate the retained-population filter. Execution markers are unvalidated, and the heuristic filter can exclude genuine pasted errors. Requiring a following user message also excludes blocks without observed feedback.

History and uncertainty. Training-only histories exclude held-out labels but can include events later than test blocks in random session splits. Session-ordered temporal splits do not resolve overlapping sessions. Split-bootstrap intervals describe variation within a reused corpus, not independent confirmation. A prospective online evaluation would require strictly past-only features and evaluation on a fresh population.

Comparators and scope. Context features summarize rather than semantically encode preceding text. The critic comparison uses one zero-shot prompt on a separate enriched sample and is not matched in supervision to the trained prose model; no trained full-trajectory critic is evaluated. Lexical exclusion does not remove all semantic admissions of trouble. Cross-agent and cross-interface transfer are not established. Block-count budgets do not measure review time, prevented defects, or intervention benefit.

6 CONCLUSION

Next-user failure reports and detected execution errors are weakly associated in the evaluated corpus. Adding agent prose to context summaries, developer history, and work statistics raises average precision by 0.047. The additive gain remains after excluding blocks with specified error or apology terms. Separately, prose alone outperforms the feature baseline under developer holdouts and temporal splits. At a fixed 10% review budget, the combined model raises report capture from 30.2% to 35.5%; read the other way, capturing 30% of reports costs 7.74% of blocks rather than 9.92%, about a fifth fewer blocks inspected for the same retrospective capture. Whether using the ranking saves review time or prevents defects requires an intervention study with a specified action and measured outcomes.

AI USE STATEMENT

Generative AI assisted with literature search, manuscript preparation, research-question refinement, methodological feedback, analysis-code implementation, and result interpretation. Experimental language-model use is described in Section 3. The author takes responsibility for the final content.

REFERENCES

Laksh Advani. From confident closing to silent failure: Characterizing false success in LLM agents. In *Workshop on Failure Modes of Agents (FAGEN) at ICML 2026*, 2026. URL https://arxiv.org/abs/2606.09863. arXiv:2606.09863.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. SWE-chat: Coding agent interactions from real users in the wild. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=KMAxjswrFe.

Praveen Kumar Bodigutla, Aditya Tiwari, Spyros Matsoukas, Josep Valls-Vargas, and Lazaros Polymenakos. Joint turn and dialogue level user satisfaction estimation on multi-domain conversations. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 3897–3909, 2020.

Valerie Chen, Rohit Malhotra, Xingyao Wang, Juan Michelini, Xuhui Zhou, Aditya Bharat Soni, Hoang H. Tran, Calvin Smith, Ameet Talwalkar, and Graham Neubig. How can we assess human-agent interactions? Case studies in software agent design. In *International Conference on Machine Learning (ICML)*, 2026. Main conference poster; icml.cc/virtual/2026/poster/64319.

Yang Deng, Wenxuan Zhang, Wai Lam, Hong Cheng, and Helen Meng. User satisfaction estimation with sequential dialogue act modeling in goal-oriented conversational systems. In *Proceedings of the ACM Web Conference (WWW)*, 2022.

Yuchen Guo, Martin Shepperd, and Ning Li. Improving classifier-based effort-aware software defect prediction by reducing ranking errors. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2024.

Jing Jiang, Liehao Li, Jinyun Hou, Xin Tan, and Li Zhang. Understanding and Predicting Accepted Code Suggestions in AI-Assisted Programming. *Proceedings of the ACM on Software Engineering (FSE)*, 2026. doi: 10.1145/3808125.

Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Scale Red Team, Christina Q. Knight, and Zifan Wang. Reliable weak-to-strong monitoring of LLM agents. In *International Conference on Learning Representations (ICLR), Oral*, 2026. URL https://arxiv.org/abs/2508.19461.

Obada Kraishan and Kulsawasd Jitkajornwanich. Plans They Abandon, Reports They Author: The Narrative Layer of Autonomous Agents, 2026. URL https://arxiv.org/abs/2609.12205. arXiv preprint arXiv:2609.12205.

Abigail See and Christopher D. Manning. Understanding and predicting user dissatisfaction in a neural generative chatbot. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 1–12, 2021.

Omar Shaikh, Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Navigating rifts in human–LLM grounding: Study and benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20832–20847, 2025. URL https://arxiv.org/abs/2503.13975.

Ningzhi Tang, Chaoran Chen, Gelei Xu, Yiyu Shi, Yu Huang, Collin McMillan, Tao Dong, and Toby Jia-Jun Li. How Coding Agents Fail Their Users: A Large-Scale Analysis of Developer-Agent Misalignment in 20,574 Real-World Sessions, 2026. URL https://arxiv.org/abs/2605.29442. arXiv preprint arXiv:2605.29442.

Arne Vanhoyweghen, Brecht Verbeken, Andres Algaba, and Vincent Ginis. Lexical hints of accuracy in llm reasoning chains. *Scientific Reports*, 2026.

Rakshith Vasudev, Melisa Russak, Dan Bikel, and Waseem Alshikh. Accurate failure prediction in agents does not imply effective failure prevention, 2026. URL https://arxiv.org/abs/2602.03338. arXiv preprint arXiv:2602.03338.

Xingyao Wang, Valerie Chen, Heng Ji, and Graham Neubig. A Rubric-Supervised Critic from Sparse Real-World Outcomes, 2026. URL https://arxiv.org/abs/2603.03800. arXiv preprint arXiv:2603.03800.

A Labels, Eligibility, and Cue Definitions

Blocks are aligned with the immediately following user prompt; those without a matching prompt are excluded. The target `is_fr` indicates `failure_report`, whereas the prior-pushback feature includes both failure reports and takeovers. The `is_human` flag selects eligible messages, not human-annotated labels.

The filter removes strings matching system-notification markup, including `task-notification`, `system-reminder`, and tool-result delimiters. It also removes messages longer than 800 characters that match specified Markdown-formatting or compiler/tool-output patterns. The latter rule may remove legitimate user reports. The 150-turn model relabeling used texts truncated to 600 characters, with shipped labels hidden from the independent language model. The one-versus-rest failure-report kappa is 0.645 and the overall multiclass kappa is 0.374. A separate human relabeling of 50 stratified turns gives a failure-report kappa of 0.650 and a four-way kappa of 0.209, with 11 shipped-label positives and 11 human-label positives among the 50 turns. Neither pass audits the filtered population block by block.

Execution markers are matched in results from Bash, BashOutput, Agent, and Task tools, plus unnamed results that do not resemble line-numbered file listings. Markers include structured nonzero exit-code/status lines, Python traceback headers, npm error lines, fatal diagnostics, selected compiler diagnostics, failed-test counts, build failure, command-not-found, permission-denied, and segmentation-fault strings. Read-like results are not simply searched for the word "error." Missing execution labels are treated as no detected marker in the cross-tabulation. Tool-name and output-format variation can still cause measurement error.

The prose cue vocabulary consists of the words *error, errors, failed, failing, fail, fails, exception, traceback, stacktrace, sorry, apologize, apologies, oops, broke, broken, bug, crash, crashed, incorrect, wrong*, and the phrases *my mistake, didn't work, doesn't work, did not work, does not work, not working, went wrong, i missed, was wrong, that failed*. Matching is case-insensitive, with word boundaries for single terms. The cue counts use full concatenated assistant text; the exclusion experiment uses cached truncated prose and excludes on prose cues only, so it does not reproduce the two-channel filter.

B Model and Split Details

The TF-IDF configuration uses at most 20,000 features, minimum document frequency 5, unigram/bigram features, and sublinear term frequency. Truncated SVD has 200 components and random seed 0. HistGradientBoostingClassifier uses 300 iterations, learning rate 0.05, at most 31 leaves, balanced class weights, and the split seed as classifier seed. Combined arms concatenate the SVD features and the relevant structured features. Logistic critic combiners use balanced class weights and a maximum of 1,000 iterations.

The context columns are `prior_pos`, `prior_corr`, `prior_rej`, `turn_pos`, and `log_turn_pos`. History columns are `hist_rate` and `hist_n`. The remaining baseline counts cover all tool uses; Bash, Edit, Read, Write, and Grep; combined edits; distinct files; Action and Research categories; test commands; assistant messages; total assistant-text length; and hedge matches.

Fifteen GroupShuffleSplit seeds use a 30% test fraction for session and user regimes. Unknown users are grouped together in the user holdout. The temporal regime orders sessions by their creation time and cuts at the 0.55, 0.60, 0.65, 0.70, and 0.75 quantiles of that session-level ordering, so every block of a session falls on one side of the boundary. Paired intervals use 5,000 resamples of the split differences with a seeded random generator.

C Critic Prompt and Sample

The historical prompt uses "completed work-block"; that wording does not establish task completion. We reproduce the prompt prefix verbatim:

> You are reviewing ONE completed work-block from an AI coding agent, exactly as the user sees it before replying. Judge the work itself: whether the change is likely wrong, incomplete, off-target, or likely to break something. Then rate from 0 to 10 how likely the user's next message reports a problem with this work (10 = certain the user reports a problem, 0 = certain the user is satisfied). Reply with ONLY the integer 0-10.

The prompt prefix is followed by an "AGENT WORK-BLOCK" delimiter and the cached text. Although the caller allows up to 12,000 characters, the cache is already truncated to 2,000. The prompt's wording about satisfaction does not change the scored outcome: evaluation uses the next-message failure-report label. Absence of such a label does not imply satisfaction.

The initial draw uses random seed 42 and samples 250 blocks from each class. API quota exhaustion left 250 positives and 176 negatives rated; the 74 missing negative ratings limit representativeness. The TF-IDF/SVD gradient-boosting prose reader (Section 3.2) is trained on sessions disjoint from the rated sample. Within fifteen session-grouped splits of that sample, logistic models are fitted using either the critic rating alone or the rating plus the prose reader's predicted probability. Both combiners and the prose-only score are evaluated on the same test rows. Adding prose yields paired gains of 0.162 [0.142, 0.183] AP and 0.211 [0.186, 0.237] ROC-AUC.

D Target and Text-Window Sensitivity

Broadening the cross-tabulated outcome to failure-report-or-takeover gives $\phi = 0.023$ and 86.1% of positives without execution markers. For any pushback, the corresponding values are -0.019 and 89.1%. These sensitivity checks preserve the weak observed association; they do not validate either label.

A separate text-window comparison on 34,046 filtered blocks reports AP 0.232 for the first 2,000 characters, 0.235 for the last 2,000, and 0.232 for the first and last 1,000 combined. Only 7.9% of blocks exceed the window. We retain the existing first-window representation; these exploratory comparisons do not constitute fresh confirmation after model selection.