Beyond Leaderboards: Protocol Cards for Trustworthy Coding-Agent Evaluation

Keqin Zhang
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
keqinzhang@link.cuhk.edu.cn

Sharon Lee*
University of Cambridge
Christ's College
Cambridge, United Kingdom
shsl2@cam.ac.uk

Abstract

Coding-agent scores depend not only on model capability, but also on the evaluation protocol that determines what information the agent receives, how outputs are produced and checked, how success is measured, and how cost is counted. Building on prior evaluation-documentation frameworks, we empirically study these dependencies and organize them into information, procedure, and measurement protocols for coding-agent evaluation.

Across 7,200 code-review records and 1,350 SWE-bench Lite oracle-file apply-level repair records, we find that evaluation conclusions are protocol-sensitive. In code review, full-raw context yields a statistically reliable but small improvement over diff-only context under deterministic file-level scoring ($\Delta\text{SNR}_{\text{file}} = 0.007$, $p = 0.002$). Increasing post-generation verification depth from V0 to V3 is not reliably beneficial ($\Delta\text{SNR}_{\text{file}} = 0.003$, $p = 0.330$). Under oracle-file access, clean-apply rate varies from 8.0% with diff-only context to 50.4% and 54.7% with structured and full-raw context, respectively; this conditional apply-level result is not an official SWE-bench resolved-rate comparison. Semantic-judge rescoring changes which factors appear significant, while alternative cost-accounting rules reverse the relative ordering of the two multi-call workflows.

These findings motivate reporting coding-agent outcomes as paired objects consisting of a score and the protocol under which it was produced. We operationalize this recommendation through protocol cards that record coding-agent-specific protocol choices and cross-cutting run metadata.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → *Software testing and debugging*.

Keywords

coding agents, agentic AI evaluation, protocol cards, trustworthy AI, LLM-as-a-judge, benchmark auditing

1 Introduction

Coding agents are evaluated as compound systems rather than isolated model calls. A reported score may depend on the task evidence shown to the agent, the workflow used to generate or revise outputs, the checks applied after generation, the metric used to score the output, and the way computation is counted.

These dependencies become especially important in repository-level software-engineering settings, where agents retrieve files, review pull requests, generate patches, call tools, and participate in iterative repair loops. A leaderboard score can therefore appear model-intrinsic even when it is partly produced by hidden choices in context construction, orchestration, verification, measurement, and cost accounting.

This paper studies the evaluation protocol rather than proposing a new coding agent. Our motivating question is:

> When a coding agent is reported to achieve a score, what protocol information must be disclosed for that score to be interpretable, reproducible, and trustworthy?

This question is related to broader work on evaluation cards, benchmark documentation, and run-level reporting. We do not claim that documenting evaluation conditions alongside scores is itself new. Instead, we study which protocol choices are particularly consequential for coding agents and provide controlled empirical evidence for their effects. Coding-agent evaluation introduces domain-specific reporting needs, including repository context construction, file-access assumptions, tool availability, the distinction between verification and refinement, patch-application rules, semantic matching, and accounting for generation and verification calls.

We organize these choices into three dimensions: the *information protocol*, which records what evidence the agent can access; the *procedure protocol*, which records how the agent produces and checks outputs; and the *measurement protocol*, which records how outputs and costs become reported scores. Cross-cutting run metadata, including model identity, prompts, decoding settings, and failure policies, is disclosed alongside these dimensions.

This paper does not introduce a new coding model, learned router, or agent architecture. Its primary contribution is an empirical audit of how evaluation conclusions vary with context, workflow, verification, measurement, and cost accounting. We use these findings to operationalize a coding-agent-specific reporting schema that complements broader evaluation-card and benchmark- documentation frameworks.

We make three contributions:

(1) **A controlled empirical audit of protocol sensitivity in coding-agent evaluation.** Across 7,200 code-review records and 1,350 oracle-file apply-level repair records, we quantify how context construction, workflow, verification depth, metric family, and budget accounting affect measured conclusions.
(2) **A coding-agent-specific operationalization of evaluation reporting.** Building on broader evaluation-documentation frameworks, we organize score-producing conditions into information, procedure, and measurement protocols. The

*Corresponding author.

schema explicitly separates verification, which observes an output, from refinement, which is permitted to modify it.

(3) **Practical reporting guidance.** We instantiate the proposed disclosure structure for the headline experiments and provide a proposed core set of protocol and run-level fields for interpreting coding-agent results.

2 Background and Positioning

Coding-agent benchmarks and evaluation protocols. SWE-bench evaluates repository-level software repair using real GitHub issues and developer tests [6]. SWE-bench Lite provides a smaller subset for lower-cost iteration [14]. Recent benchmark-auditing work has argued that benchmark construction, solution leakage, weak tests, and harness details can materially affect conclusions [1, 15]. These studies motivate treating benchmark construction, execution, and reporting choices as part of the interpretation of an agent score.

Evaluation documentation and reporting frameworks. Protocol cards build on a broader line of work that documents the conditions under which evaluation results are produced. BenchmarkCards document benchmark-level properties, intended uses, methodologies, and limitations, while EvalCards and Evaluation Cards provide broader schemas for disclosing evaluation and run-level metadata [2, 3, 13]. Recent coding-agent work such as Claw-SWE-Bench further shows that prompting, workspace construction, runtime budgets, patch extraction, and evaluator configuration form part of the effective benchmark contract [18].

Our contribution is narrower and empirical. We do not propose protocol cards as a replacement for these frameworks or claim that score-plus-metadata reporting is new. Instead, we identify a coding-agent-specific decomposition—information, procedure, and measurement protocols—and use controlled comparisons to measure how context construction, verification and refinement, matcher choice, and budget accounting are associated with different evaluation conclusions.

Harnesses, context, and workflows. Recent harness- and workflow-engineering work argues that LLM-system performance depends not only on model weights, but also on external code and workflow structure that control context construction, retrieval, memory, and orchestration [7, 11, 16]. Meta-Harness, for example, optimizes executable harness code by giving a coding-agent proposer access to prior source code, scores, and execution traces [7]. Whereas these studies optimize harnesses or workflows for performance, we treat harness- and workflow-level choices as variables that must be disclosed and audited when reporting scores.

Metric validity and LLM-as-a-judge. LLM-based judging can recover semantic equivalence missed by deterministic matching, but it can also introduce reliability and bias concerns, including prompt-, position-, style-, and scoring-related biases [4, 8, 17]. We therefore treat semantic judging as a validation layer rather than the only headline metric. Our headline metric remains deterministic and reproducible, and we explicitly test whether semantic judging changes factor-level conclusions.

Workflow, refinement, and escalation. Prior systems such as MetaGPT and ChatDev decompose software-development tasks across role-specialized agents [5, 10], while Reflexion and Self-Refine improve agents through feedback, memory, or revision [9, 12]. Recent workflow benchmarks further treat workflow structure itself as an evaluation target [11]. Our workflows are narrower controlled evaluation conditions, not new collaborative coding architectures. We also include a bounded human audit inspired by selective prediction and escalation, but because it uses one annotator, it is diagnostic rather than population-level validation.

Table 1: Organizing dimensions of a coding-agent protocol card. Cross-cutting run metadata is required in addition to the three dimensions.

Dimension	What it records	Audit question
Information	Dataset/version, task selection, context and oracle access, retrieval, packaging, token budget	What evidence could the agent access?
Procedure	Workflow, tools, environment, revision policy, verification depth, retry and failure handling	How was the output produced and checked?
Measurement	Metric, matcher/judge, human audit, aggregation, uncertainty, cost accounting	How did the output become a reported score?

Cross-cutting run metadata: model/provider/version, prompts, decoding settings, seed support, dates, and software environment.

3 Protocol Cards for Coding-Agent Evaluation

3.1 Protocol-card dimensions

A protocol card is a concise description of the evaluation protocol that produced a coding-agent score. It is intended to accompany leaderboard-style results, not replace them. The goal is to make explicit the assumptions that determine how an agent score should be interpreted.

The three dimensions provide an organizing decomposition of the evaluation conditions that produce a coding-agent score. They answer three distinct audit questions: what the agent was allowed to see, how it was allowed to act, and how its output was scored. We do not claim that these dimensions exhaust all variables required for reproducibility. Cross-cutting run metadata—including exact model and provider versions, prompt templates, decoding settings, seed support, retry and failure policies, execution dates, and environment versions—remain first-class disclosure fields within the card.

3.2 Verification and refinement as separate operations

A coding-agent evaluation may either *verify* an output or allow the agent to *refine* it. Verification scores, filters, flags, or escalates an already generated output without modifying the final answer. Refinement returns a draft or feedback to the generator and permits a revised answer.

For example, a verifier may determine whether a review finding correctly localizes an unlocked read–write path in

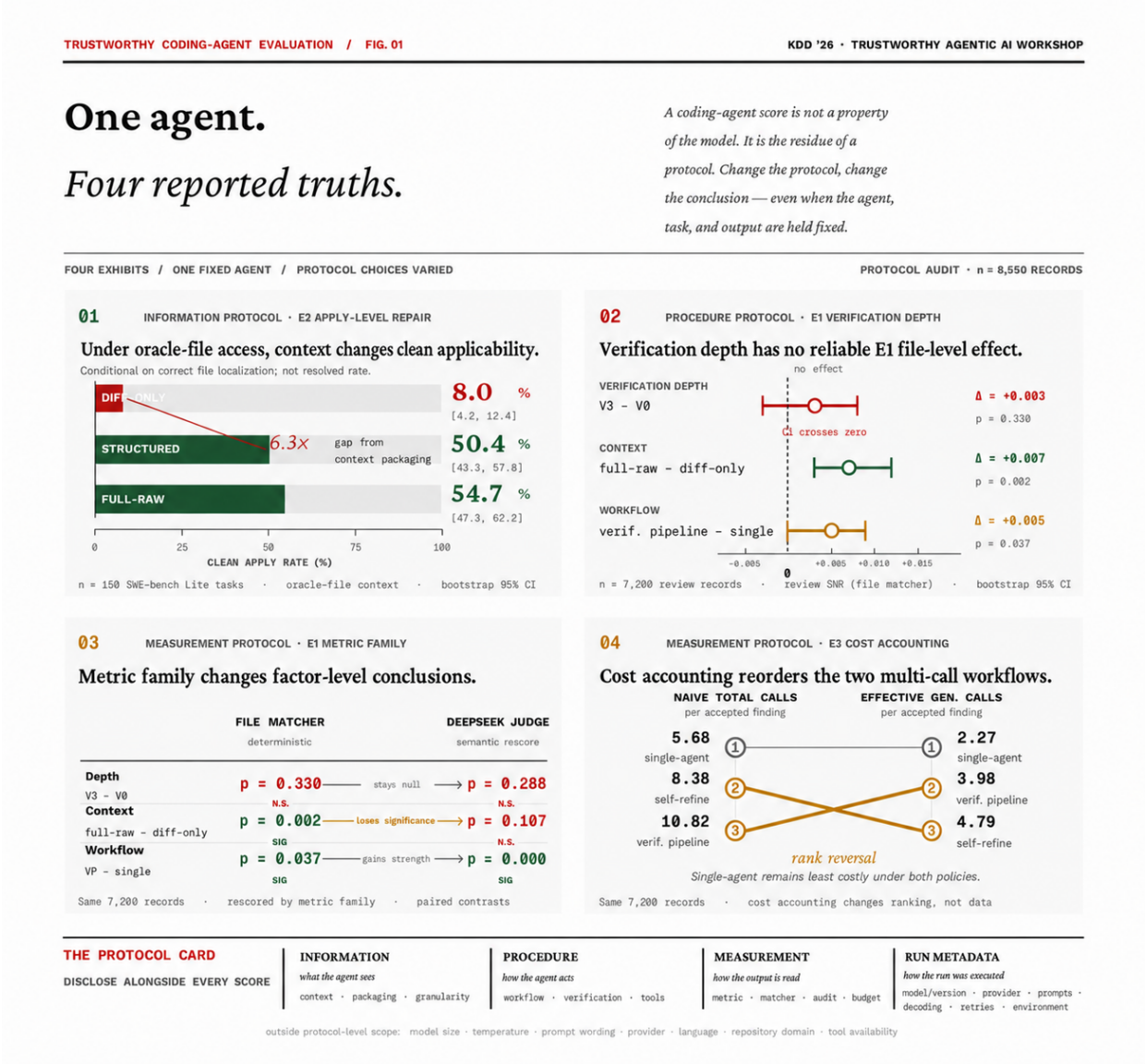

Figure 1: Overview of the measured forms of protocol sensitivity. Under oracle-file access, context construction is associated with large differences in clean patch applicability; in code review, verification depth has no reliable effect under deterministic file-level scoring; semantic judging changes factor-level conclusions; and cost accounting changes the relative ordering of the two multi-call workflows. Protocol cards record these information, procedure, and measurement choices together with cross-cutting run metadata.

`CacheManager.get()`, but it cannot rewrite the finding. A self-refine workflow may instead replace a vague statement such as "there may be a concurrency issue" with a localized explanation of the read–write race. Protocol cards therefore report verification and refinement separately.

3.3 Procedure protocol: workflow and verification depth

The procedure protocol records both the workflow that produces the output and the verification depth applied to it. A single-agent workflow produces one final answer; self-refine permits revision; and a verification pipeline separates generation from checking and may trigger one additional generation after checking an initial draft.

Verification depth is an orthogonal factor. V0–V3 apply increasingly restrictive post-generation checks to the finalized candidate,

but do not themselves permit revision. Workflow can therefore change the generated answer, whereas final-scoring verification changes only how that answer is checked. Table 2 gives the exact E1 operations.

3.4 Measurement protocol: what counts as signal or noise

The measurement protocol specifies how an agent output becomes a reported score. For code review, an output usually contains one or more *findings*. A finding is treated as *signal* if it matches a ground-truth concern under a specified matcher. A finding is treated as *noise* if it is unmatched, too vague to localize, or points to an irrelevant file.

Our headline deterministic metric is file-level signal-to-noise ratio, denoted $\mathrm{SNR}_{\mathrm{file}}$. Under matcher family m, let M_m be the set of generated findings that match a ground-truth concern, and let U_m be the set of unmatched findings. We define:

$$\mathrm{SNR}_m = \frac{|M_m|}{|M_m| + |U_m|}. \tag{1}$$

For example, suppose an agent produces five review findings. If two findings point to files that match ground-truth concerns and three findings are unmatched, vague, or irrelevant, then $\mathrm{SNR}_{file} = 2/(2+3) = 0.40$. This metric rewards useful localized findings and penalizes extra unmatched findings. It is conservative because it measures file-level localization rather than full semantic correctness.

The same generated output can look different under different matchers. File-level matching asks whether the finding localizes to the relevant file. A semantic judge can recover findings that use different wording but describe the same underlying issue. This distinction motivates a measurement protocol that records whether the score is produced by deterministic file-level matching, semantic judge-based scoring, or both. We therefore use $\mathrm{SNR}_{\mathrm{file}}$ as the reproducible headline metric and use judge-based scoring as a validation layer.

Bug-hit rate (BHR) is complementary to SNR. BHR asks whether the agent finds at least one ground-truth concern for a task. SNR asks how much of the generated output is useful signal rather than unmatched noise. A model can have high BHR but low SNR if it finds one real issue while also producing many irrelevant findings. Conversely, a model can have moderate BHR but higher SNR if it produces fewer but more precise findings.

For repair-style tasks, the measurement protocol shifts from finding-level evaluation to patch-level evaluation. We use clean apply rate to measure whether a generated patch can be applied to the base repository state, and gold-file alignment to measure whether the patch modifies the relevant files. These patch-level metrics are useful for apply-level repair analysis, but they should not be interpreted as official benchmark resolved rates unless executable tests are run.

4 Experimental Design

This section instantiates the protocol-card dimensions defined in Section 3 into three primary empirical studies and two diagnostic analyses. Section 3 defines context regime, workflow, verification depth, and measurement protocol as abstract evaluation dimensions. Here, we operationalize these dimensions as controlled experimental factors. Table 3 summarizes the design.

4.1 Experimental factors

All studies are organized around the protocol-card dimensions introduced above. The *context regime* controls how file information is presented, including diff-only, structured, and full-raw variants. The *workflow* controls how outputs are generated or revised, including single-agent generation, self-refine, and verification-pipeline settings. The *verification depth* controls the amount of post-generation checking, using the V0–V3 levels defined in Table 2. The *measurement protocol* controls which metric observes the output, including deterministic file-level scoring, semantic judge scoring, apply-level repair metrics, and budget accounting.

4.2 Empirical studies

E1: code-review evaluation. E1 is the main factorial review sweep over 200 code-review tasks, four verification depths, three context regimes, and three workflows. Outputs are review findings evaluated with deterministic file-level scoring and bug-hit rate. E1-Judge rescoring applies a semantic judge to the same E1 outputs to test metric-family sensitivity.

E2: oracle-file conditional patch construction. E2 isolates patch construction after file localization has succeeded. We select the first 150 SWE-bench Lite test instances with at least one `FAIL_TO_PASS` test using a deterministic procedure. Oracle patches are used only to identify the relevant repository-relative file paths; patch contents, edit locations, and target edits are not exposed to the model. The corresponding file contents are retrieved from the base commit and serialized under the three context regimes.

Conceptually, E2 estimates

$$P(\text{clean apply} \mid \text{relevant files correctly identified}),$$

rather than the end-to-end probability that an agent independently localizes the correct files, constructs a semantically correct patch, and passes executable tests. E2 therefore varies context regime and workflow while holding oracle-based file identification fixed. It does not measure official SWE-bench resolved performance.

E3: cost accounting sensitivity. E3 reinterprets outputs from E1 under alternative cost-accounting schemes. Task inputs and generated outputs are held fixed; only the cost aggregation rule changes. The goal is to test whether workflow rankings remain stable when cost is counted as total calls versus generation calls.

E4–E5: diagnostic analyses. Additional diagnostic analyses examine human-audit boundaries and granularity sensitivity. They support interpretation of E1–E3 but are not treated as headline empirical evaluations.

4.3 Unit of analysis and statistical protocol

The unit of analysis is a generated record under a fixed task, context regime, workflow, and verification depth. For E1, paired contrasts compare protocol conditions that share the same task identity, reducing variance from task difficulty. Tasks are loaded from the preprocessed c-CRAB JSONL in file order, and the first 200

Table 2: Operational definition of verification depth in E1. All checks are local and deterministic, introduce no additional LLM calls, and do not revise the finalized candidate. Detailed acceptance rates and verifier-event counts are reported in the supplement.

Depth	Added check	Operational rule	Role in evaluation
V0	None	The finalized generator output is measured directly.	Baseline with no depth-specific verification.
V1	`TestVerifier`	Requires at least one normalized source-file path or basename shared by a predicted and reference finding.	Failed checks set `accepted=false`, but the candidate is retained for metric computation.
V2	V1 + `StaticCheckVerifier`	Additionally rejects predicted findings that exactly match `known_false_positives`; this set was empty in the reported c-CRAB run.	Operationally close to V1 in the reported experiment.
V3	V2 + `CriticVerifier`	Uses a deterministic oracle-aware check requiring nonempty predictions to be an exact subset of the reference findings; no critic LLM is called.	May flag or escalate a candidate but does not revise the final output.

In the `verification_pipeline` workflow, these checks may also be applied to an initial draft and trigger one additional generation before final scoring.

loaded task IDs are selected deterministically. This improves reproducibility but may introduce order bias; we therefore report the cross-provider/model slice as a robustness check rather than as a randomized validation set.

For E1, headline contrasts are paired by task identity: conditions are compared only when they share the same underlying task. We use task-level bootstrap resampling with replacement to construct 95% confidence intervals and paired p-values, thereby preserving the within-task dependence among protocol conditions. Specifically, we use 10,000 task-level bootstrap resamples (percentile method, fixed random seed) to compute all confidence intervals and two-sided paired-bootstrap p-values. The complete resampling implementation, random seed, and analysis configuration are released with the artifact. For E2, bootstrap samples are likewise drawn at the task level using 10,000 resamples, and we report clean-apply rate and gold-file alignment over the deterministically selected SWE-bench Lite instances. These metrics are not interpreted as executable-test resolved rates. For E3, we hold generated records fixed and vary only the accounting rule used to aggregate generation and verification calls.

5 Results

We report results for the three primary empirical studies defined in Section 4. E1 tests code-review protocol sensitivity, E2 tests apply-level repair under oracle-file access, and E3 tests budget-accounting sensitivity. We organize the results by protocol question rather than by experiment: information protocol, procedure protocol, measurement protocol, and cost accounting. R1–R4 are the headline measured findings. E4–E5 are used only for diagnostic interpretation and are not reported as headline results.

5.1 R1: Context sensitivity differs across evaluation settings

Figure 2 reports the three prespecified E1 contrasts under deterministic file-level scoring. Full-raw context improves over diff-only context by $\Delta\text{SNR}_{\text{file}} = +0.007$ ($p = 0.002$), with a 95% confidence interval of $[+0.003, +0.012]$. The difference is statistically reliable but small in absolute magnitude. It is larger than the other two reported E1 contrasts, but it should not be interpreted as a large improvement in review quality. A cross-provider/model robustness slice shows the same direction of association ($\Delta = +0.019$, $p < 0.001$). Increasing verification depth from V0 to V3 remains small and non-significant.

The E1 result shows that context construction can measurably affect file-level review scores even when the absolute difference is modest. It motivates disclosing context regime and task packaging, rather than establishing that context universally dominates workflow or verification.

E2 exhibits a substantially larger context-associated difference, but it measures a different task, metric, and unit of analysis from E1. Conditional on oracle identification of the relevant files, diff-only context yields an 8.0% clean-apply rate, compared with 50.4% for structured context and 54.7% for full-raw context. This result indicates that context serialization can strongly affect patch formation after file localization has succeeded.

The E2 values should not be compared directly with the E1 effect size. E2 excludes file-retrieval and localization failures, and it does not evaluate executable correctness. It therefore does not estimate the improvement that an end-to-end coding agent would obtain on official SWE-bench resolved rate.

A common diff-only failure is path ambiguity. In SWE-bench-style patches, the output must name exact repository-relative file paths. Under diff-only context, the model may identify the intended logic change but generate a patch for a partial path, omit the expected prefix convention, or place the edit in a file not present in the patch context. Such patches fail clean application even when the high-level repair idea is plausible. Structured context mitigates this failure by exposing file-level anchors before generation.

5.2 R2: Verification-depth and workflow effects are limited in the evaluated E1 setting

Within E1, increasing verification depth from V0 to V3 yields a small and non-significant change under deterministic file-level scoring ($\Delta\text{SNR}_{\text{file}} = +0.003$, $p = 0.330$). The verification pipeline has a modest positive contrast relative to single-agent generation under the file matcher ($\Delta\text{SNR}_{\text{file}} = +0.005$, $p = 0.037$). These effect sizes are specific to the evaluated tasks, models, prompts, context regimes, and measurement rule.

The results do not establish that verification is generally ineffective, nor do they show that verification and context have a universal ordering. They show that the tested post-generation checks do not reliably improve the deterministic file-level metric when verification depth is increased from V0 to V3 in E1.

Table 3: Main empirical audits used in this paper. E4/E5 diagnostics are reported only as limitations or supplementary material.

Audit	Design	Protocol-card dimension tested	Primary output
E1 Review audit	200 tasks × 4 depths × 3 contexts × 3 workflows = 7,200 records	Information and procedure protocols	SNR_{file}, BHR
E1-Judge	Semantic rescore of E1	Measurement protocol	SNR_{judge}, factor significance
E2 Repair bridge	150 SWE-bench Lite tasks × 3 contexts × 3 workflows = 1,350 records	Information protocol under oracle-file apply-level repair	Clean apply rate, gold-file alignment
E3 Budget audit	Post-hoc cost analysis over E1	Measurement protocol and budget policy	Workflow ranking under cost accounting

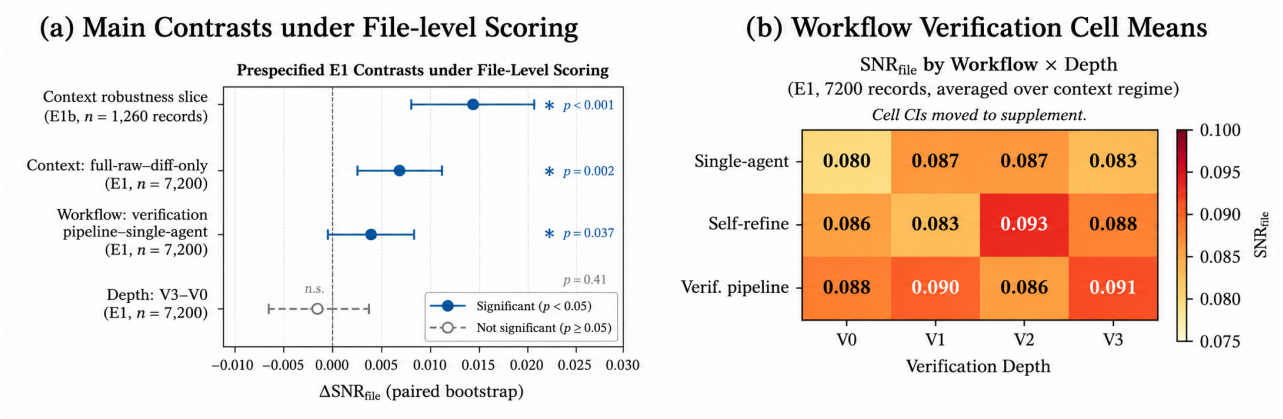

Figure 2: E1 results under deterministic file-level scoring. (a) Full-raw context yields a statistically reliable but small improvement over diff-only context. V3 versus V0 is not significant, and the verification-pipeline contrast is modest. (b) Workflow-by-verification cell means averaged over context regimes. The figure characterizes the evaluated E1 setting and does not establish a universal ordering of context, workflow, and verification effects.

Table 4: E1 headline contrasts under deterministic file-level scoring.

Contrast	ΔSNR_{file}	95% CI	p-value
V3 − V0	+0.003	[-0.003, +0.008]	0.330
full-raw − diff-only	+0.007	[+0.003, +0.012]	0.002
verification pipeline − single-agent	+0.005	[0.000, +0.009]	0.037

Interpretation. One possible explanation is that some failures arise before verification, when relevant evidence is absent or poorly serialized. However, the present design does not isolate evidence availability from context length, formatting, ordering, prompt design, or verifier strength. We therefore treat this explanation as a hypothesis rather than a causal conclusion.

5.3 R3: Measurement protocol changes which factors appear significant

Figure 4 compares deterministic file matching with DeepSeek V3 judge labels on the same E1 outputs. The judge assigns a slightly higher aggregate SNR (0.093 versus 0.087) and a higher BHR (0.440 versus 0.365). These differences show that the semantic judge labels more findings as matches; they do not by themselves establish that the additional matches are correct.

We therefore do not treat the judge output as ground truth. The judge rescore is a measurement-sensitivity analysis: it asks whether a more semantic observation rule changes factor-level conclusions. Table 6 shows that the evidence associated with context and workflow differs across the two measurement rules, while the V3–V0 depth contrast remains non-significant.

The file–judge discrepancy is not treated as a contradiction. File-level scoring operationalizes success through deterministic localization, whereas the semantic judge applies a model-based matching rule that can label differently worded findings as matches. The present analysis does not determine which observation rule is universally more valid, because the judge has not been calibrated on a sufficiently large independent human-labeled set.

The result instead demonstrates that metric choice is part of the evaluation protocol. A protocol card should therefore disclose whether a reported score is produced by deterministic matching, semantic judging, human audit, or a combination of these measurement layers, together with the judge model, prompt, decoding settings, and failure-handling policy where applicable.

Table 5: E2 oracle-file apply-level repair bridge. Metrics are clean-apply and gold-file alignment, not official SWE-bench resolved rates. Brackets show bootstrap 95% CIs.

Context	Apply rate	Gold-file recall	Gold-file precision
diff-only	8.0 [4.2, 12.4]	79.8 [73.3, 85.8]	78.0 [71.3, 84.4]
structured	50.4 [43.3, 57.8]	96.7 [94.2, 98.7]	96.9 [94.5, 98.8]
full-raw	54.7 [47.3, 62.2]	96.0 [93.3, 98.2]	96.5 [94.0, 98.5]

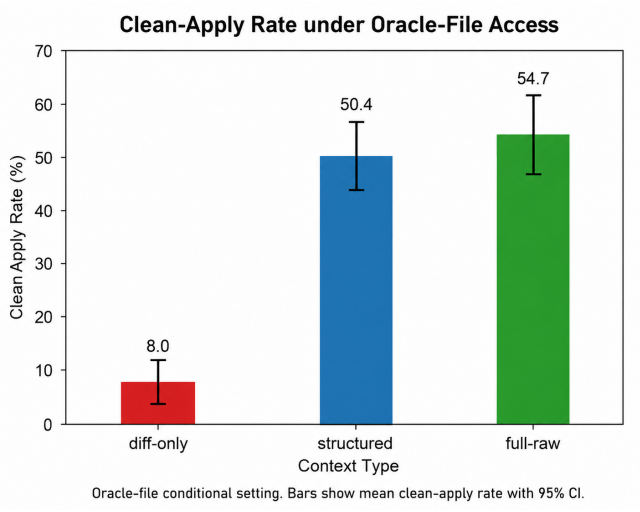

Figure 3: E2 clean-apply rate conditional on oracle identification of relevant files. Structured and full-raw context are associated with higher clean-apply rates than diff-only context. The experiment excludes file-localization failures and does not run executable tests; the values are therefore not official SWE-bench resolved rates.

Table 6: Metric sensitivity in E1. The evidence associated with context and workflow depends on the measurement rule, while the depth contrast remains non-significant. A difference between significant and non-significant results does not itself establish that the two effect sizes differ significantly.

Contrast	File matcher	DeepSeek judge
Depth: V3–V0	n.s. ($p = 0.330$)	n.s. ($p = 0.288$)
Context: full-raw–diff-only	sig. ($p = 0.002$)	n.s. ($p = 0.107$)
Workflow: VP–single	marginal ($p = 0.037$)	sig. ($p < 0.001$)

5.4 R4: Cost accounting reverses the relative order of the two multi-call workflows

E3 asks whether workflow rankings are stable when the cost policy changes. Let A_w denote accepted outputs, G_w generation calls, and V_w verification calls for workflow w. We compare two accounting schemes:

$$C_w^{\text{total}} = \frac{G_w + V_w}{A_w}, \qquad C_w^{\text{gen}} = \frac{G_w}{A_w}.$$

Table 7: Cost accounting crossover effects on workflow ranking (E3). Values indicate mean cost per accepted output; lower is better. SA = single-agent, SR = self-refine, VP = verification pipeline.

Accounting scheme	Rank 1	Rank 2	Rank 3
Total calls / accepted	SA (5.68)	SR (8.38)	VP (10.82)
Generation calls / accepted	SA (2.27)	VP (3.98)	SR (4.79)

The first treats verification calls as primary cost, while the second reports generation cost separately from verification overhead. We use these cost ratios as an accounting illustration rather than as the headline quality metric; the headline quality comparison remains SNR_{file}.

Thus, workflow comparisons should report at least total calls, generation calls, verification calls, and the cost policy used to aggregate them. Table 7 shows the accounting crossover: among non-single-agent workflows, self-refine is cheaper under total-call accounting, while the verification pipeline becomes cheaper under generation-call accounting. This ranking change supports our claim that cost accounting is part of the measurement protocol rather than a reporting detail.

Single-agent generation remains the least costly workflow under both accounting rules. The observed reversal is therefore limited to self-refine and the verification pipeline. Moreover, call counts are an accounting proxy rather than a complete economic-cost model: they do not capture input and output tokens, provider-specific prices, caching, parallel latency, or heterogeneous model costs.

6 From Scores to Protocol Cards

Consistent with prior evaluation-documentation work, we treat a reported result as a paired object:

$$(\text{score}, \text{protocol}).$$

The score summarizes measured performance, while the protocol records the conditions under which that performance was produced. Our contribution is not the general observation that scores require metadata; it is the controlled coding-agent sensitivity analysis and the domain-specific operationalization of information, procedure, measurement, and run-level disclosures.

Table 8 summarizes the proposed core disclosure fields.

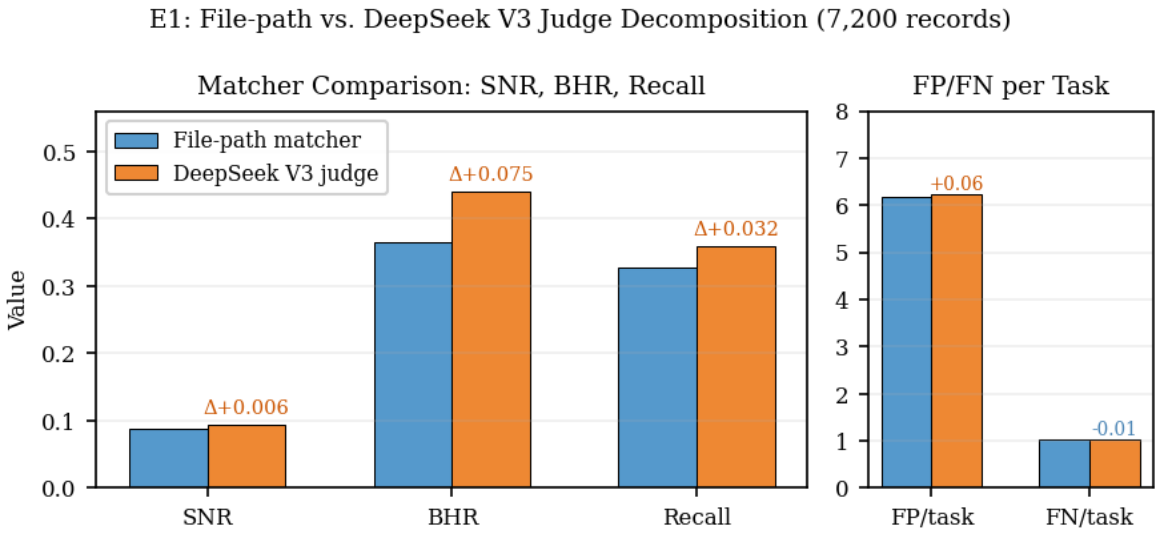

Figure 4: File-path matcher versus DeepSeek V3 judge labels on E1. The semantic judge labels more findings as matches, increasing the observed BHR and recall while changing aggregate SNR only slightly. These differences characterize measurement sensitivity and should not be interpreted as evidence that judge-only matches are necessarily correct. Factor-level results are reported in Table 6.

How Protocol Cards Should Be Used

A protocol card is useful in three practical scenarios where scalar scores alone are ambiguous. First, it helps compare leaderboard results produced under different context regimes or verification policies: a higher score under full-raw context should not be interpreted in the same way as a score obtained under diff-only context. Second, it helps diagnose why an apparent improvement may disappear under a different matcher, judge, or budget policy. Third, it supports reproducibility by making task packaging, measurement, and cost assumptions explicit. A protocol card is therefore not a replacement for metrics such as pass@1, resolved rate, clean apply rate, or SNR_{file}; it is an audit layer that explains how those metrics were produced.

Proposed Core Disclosure Fields

We propose a core rather than universally sufficient disclosure set. It combines the variables audited here with reproducibility metadata from prior reporting frameworks; its human usefulness and authoring cost remain to be validated.

Score-only reporting identifies which number is higher. Protocol disclosure records the conditions under which that number was produced. Protocol cards do not replace metrics such as pass@1, resolved rate, clean-apply rate, or SNR_{file}; they provide an audit layer for interpreting and reproducing those metrics.

7 Discussion and Limitations

Protocol Sensitivity

Our results should not be read as establishing that one context regime, workflow, verification depth, or metric is universally best. They show that measured coding-agent conclusions are conditional on the evaluated protocol. Context construction changes the available and serialized evidence; workflow and verification determine how outputs are produced and checked; metric family changes the observation rule; and cost accounting reverses the relative ordering of self-refine and the verification pipeline while leaving single-agent generation least costly under both policies.

Verification and Evidence

The weak V0–V3 contrast should not be interpreted as evidence that verification is generally ineffective. It shows only that the tested depth-specific checks do not reliably improve the deterministic file-level metric in E1. Verification may still affect filtering, escalation, semantic plausibility, or executable correctness under other tasks and measurement rules. The present results motivate reporting information and procedure protocols together, but they do not establish a universal causal ordering between context and verification.

Apply-Level Repair Boundary

E2 should be interpreted as an oracle-file apply-level repair bridge. Its purpose is to isolate whether context packaging improves patch applicability once relevant files are known. Clean apply rate and gold-file alignment are engineering validity proxies, not guarantees of semantic correctness or official SWE-bench resolved rate.

Threats to Validity

Our results should be interpreted within several validity boundaries. First, SNR_{file} measures file-level localization signal rather than full semantic correctness. Second, E1 uses a deterministic task slice, which improves reproducibility but may introduce order and

Table 8: Proposed core disclosure fields for coding-agent evaluation. The field set is not claimed to be universally sufficient.

Field group	Required disclosure
Evaluation identity	Dataset and version; split; task type; complete task-ID list or sampling rule; evaluation date.
Model identity	Exact model and API version; provider; identities of the generator, refiner, verifier, and semantic judge where applicable.
Inference settings	System and user prompts or prompt identifiers; temperature; top-p; maximum output length; seed support; retry, timeout, and parse-failure policies.
Information protocol	Accessible task evidence; oracle information; retrieval policy; context regime; serialization and ordering; token budget; truncation policy.
Procedure protocol	Workflow topology; tool access; execution environment; revision policy; verification depth; number and type of generation and verification calls.
Measurement protocol	Primary and secondary metrics; exact matcher or judge; judge prompt; human-audit procedure; aggregation rule; uncertainty estimation; missing-output and failure handling.
Budget policy	Generation calls; verification calls; token use, latency, or monetary cost where available; aggregation rule and cost cap.
Known limitations	Proxy metrics; oracle assumptions; task-selection bias; unvalidated judges; unsupported generalizations; unavailable artifacts.

repository-composition bias; the cross-provider/model slice is a robustness check rather than a randomized validation set. Third, E2 is oracle-file and apply-level, so it excludes retrieval failures and does not establish official SWE-bench resolved rate.

Fourth, the human audit is single-author and diagnostic and should not be interpreted as population-level human validation. Fifth, structured and full-raw contexts may differ in ordering, redundancy, formatting, and length as well as information content, preventing a pure causal attribution to information quantity.

Sixth, the DeepSeek V3 judge is a measurement model rather than ground truth. Its labels may depend on the exact model version, provider, prompt, decoding configuration, and failure policy, and we do not have a sufficiently large independent human-labeled set to establish its absolute calibration. Seventh, the proposed disclosure fields have not been validated through a user study, and we do not measure the time required to create or maintain a protocol card. Eighth, call-count accounting does not represent complete token, latency, energy, or monetary cost.

These limitations define the scope of our empirical claims and illustrate the conditions that should accompany reported scores.

8 Reproducibility and Ethical Considerations

Scope and Non-Headline Diagnostics

Human audit and granularity are included as protocol-card fields, but they are not headline causal evidence in this paper. They motivate future protocol-aware evaluation studies rather than establishing population-level calibration or causal review-cadence effects.

Model size, temperature, prompt wording, provider, programming language, repository domain, and tool availability can also affect performance. We treat these as implementation-level or future-extension variables rather than the main protocol-level axes studied in this paper.

Reproducibility. Code, experiment configurations, selected task IDs, metric definitions, and table- and figure-generation scripts are available in our public artifact repository. The repository documents the E2 instance-selection rule and maps each headline table and figure to its input records. The main paper remains self-contained.

Ethical and Societal Considerations

This work studies evaluation protocols for coding agents. Better protocol reporting can reduce overconfidence in automated code tools by making context, metric, cost, and escalation assumptions explicit. However, benchmark scores can still be misused if interpreted as deployment-safety guarantees. We therefore recommend reporting protocol settings, failure modes, and conditions under which human review remains necessary.

9 Conclusion

We presented a controlled empirical audit of protocol sensitivity in coding-agent evaluation and used the findings to operationalize a coding-agent-specific protocol-card schema. In code review, full-raw context yields a statistically reliable but small file-level improvement, while increasing verification depth from V0 to V3 is not reliably beneficial in the evaluated E1 setting. Under oracle-file access, context construction is associated with large differences in clean patch applicability, but this conditional apply-level result is not an official SWE-bench resolved-rate comparison.

Semantic-judge rescoring changes factor-level conclusions, although judge-only matches are not treated as ground truth. Alternative cost policies reverse the relative ordering of self-refine and the verification pipeline, while single-agent generation remains least costly under both policies.

These findings do not establish that protocol cards are a complete or universally sufficient reporting standard. They show that coding-agent scores are conditional on information, procedure, measurement, and run-level choices that should be disclosed. Coding-agent results are therefore more interpretable when reported as a score together with the protocol and run configuration that produced it.

References

[1] Reem Aleithan et al. 2025. SWE-Bench+: Enhanced Coding Benchmark for LLMs. OpenReview. https://openreview.net/forum?id=pwIGnH2LHJ

[2] Ruchira Dhar, Danae Sanchez Villegas, Antonia Karamolegkou, Alice Schiavone, Yifei Yuan, Xinyi Chen, Jiaang Li, Stella Frank, Laura De Grazia, Monorama Swain, Stephanie Brandl, Daniel Hershcovich, Anders Søgaard, and Desmond Elliott. 2025. EvalCards: A Framework for Standardized Evaluation Reporting. arXiv:2511.21695 [cs.CL] doi:10.48550/arXiv.2511.21695

[3] Avijit Ghosh, Anka Reuel, Jenny Chim, Wm. Matthew Kennedy, Srishti Yadav, Jennifer Mickel, Yanan Long, Andrew Tran, Anastassia Kornilova, Damian Stachura, Kevin Klyman, Felix Friedrich, Jeba Sania, Max Lamparth, Jan Batzner, Anoop Mishra, Eliya Habba, Yixiong Hao, Nathan Heath, Shalaleh Rismani, Usman Gohar, Andrea Loehr, David Manheim, Ruchira Dhar, Sree Harsha Nelaturu, Aarush Sinha, Leshem Choshen, Drishti Sharma, Ishan Khire, Amit Saha, Subramanyam Sahoo, Michael Hardy, Michael Alexander Riegler, Kabir Manghnani, Michelle Lin, Yanan Jiang, Yilin Huang, Asaf Yehudai, Jessica Ji, Aris Hofmann, Mubashara Akhtar, Nuno Moniz, Yacine Jernite, Stella Biderman, Zeerak Talat, Sanmi Koyejo, Mykel Kochenderfer, and Irene Solaiman. 2026. Evaluation Cards: An Interpretive Layer for AI Evaluation Reporting. arXiv:2606.09809 [cs.AI] doi:10.48550/arXiv.2606.09809

[4] Jiawei Gu et al. 2024. A Survey on LLM-as-a-Judge. arXiv:2411.15594 [cs.CL] https://arxiv.org/abs/2411.15594

[5] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven K. S. Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[6] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *International Conference on Learning Representations*. https://www.swebench.com/

[7] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-Harness: End-to-End Optimization of Model Harnesses. arXiv:2603.28052 [cs.AI] https://arxiv.org/abs/2603.28052

[8] Qingquan Li, Shaoyu Dou, Kailai Shao, Chao Chen, and Haixiang Hu. 2025. Evaluating Scoring Bias in LLM-as-a-Judge. arXiv:2506.22316 [cs.CL] https://arxiv.org/abs/2506.22316

[9] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651 [cs.CL] https://arxiv.org/abs/2303.17651

[10] Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Jian Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Communicative Agents for Software Development. arXiv:2307.07924 [cs.SE] https://arxiv.org/abs/2307.07924

[11] Shuofei Qiao et al. 2025. Benchmarking Agentic Workflow Generation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=vunPXOFmoi

[12] Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs.AI] https://arxiv.org/abs/2303.11366

[13] Anna Sokol, Elizabeth Daly, Michael Hind, David Piorkowski, Xiangliang Zhang, Nuno Moniz, and Nitesh Chawla. 2025. BenchmarkCards: Standardized Documentation for Large Language Model Benchmarks. In *Advances in Neural Information Processing Systems*, Vol. 38. Curran Associates, Inc. arXiv:2410.12974 [cs.CL] https://arxiv.org/abs/2410.12974

[14] SWE-bench Team. 2026. SWE-bench Lite. https://www.swebench.com/lite.html. Accessed 2026.

[15] Xinming Tu, Tianze Wang, Kexin Huang, Yuanhao Qu, and Sara Mostafavi. 2026. BenchGuard: Who Guards the Benchmarks? Automated Auditing of LLM Agent Benchmarks. arXiv:2604.24955 [cs.AI] https://arxiv.org/abs/2604.24955

[16] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2024. AFlow: Automating Agentic Workflow Generation. arXiv:2410.10762 [cs.AI] https://arxiv.org/abs/2410.10762

[17] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL] https://arxiv.org/abs/2306.05685

[18] Mengyu Zheng, Kai Han, Boxun Li, Haiyang Xu, Yuchuan Tian, Wei He, Hang Zhou, Jianyuan Guo, Hailin Hu, Lin Ma, Chao Xu, Guohao Dai, Lixue Xia, Yunchao Wei, Yunhe Wang, and Yu Wang. 2026. Claw-SWE-Bench: A Benchmark for Evaluating OpenClaw-style Agent Harnesses on Coding Tasks. arXiv:2606.12344 [cs.LG] doi:10.48550/arXiv.2606.12344