# BETWEEN SPOILERS AND GUESSWORK: AVOIDING OVER-SPECIFICATION IN CODING BENCHMARKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Benchmarks drive AI progress, but are hampered by task under-specification and over-specification. Task under-specification causes guesswork, where solutions have to make things up, or worse, resort to contamination or reward hacking. Task over-specification causes spoilers, where solutions can bypass some of the harder steps the benchmark was supposed to measure. Since future-looking benchmarks should be difficult, over-specification reduces a benchmark's shelflife of community adoption. Repository-level coding benchmarks have helped the field progress but experience unnecessary churn due to both over- and under-specification. For instance, SWE-bench Pro, FEA-bench, Doc2Feat-bench, and Multi-SWE-bench all tackle under-specification by augmenting task statements with hints derived from ground truth. We show that leaking ground-truth information into the task statement causes over-specification in some of these benchmarks. We experiment with various specification levels, trying to thread the needle between spoilers and guesswork. Based on these results, we recommend techniques for deriving parsimonious hints. We will release our hints so the community can obtain more reliable signals from popular benchmarks.

## 1 INTRODUCTION

Repository-level coding benchmarks have contributed to great advancements in coding agents by evaluating them via executing held-out tests against their proposed patches (Jimenez et al., 2024). For these evaluations to measure implementation ability, the task description and available repositories have to communicate the requirements enforced by the tests. Achieving this alignment has proven difficult. OpenAI introduced SWE-bench Verified to reduce underspecification and unfair testing (OpenAI, 2024). It later recommended SWE-bench Pro after finding test flaws and contamination in Verified (OpenAI, 2026b), then withdrew that recommendation after finding underspecified prompts and overly restrictive tests in Pro (OpenAI, 2026a). This churn is evidence for both the importance and difficulty of this problem. One source of misalignment is a test that requires a particular new function, class, file, or attribute name that the agent cannot infer from the task description or repository. The problem is especially relevant to feature-addition benchmarks (Li et al., 2025; Deng et al., 2025a), where the new names may be entirely missing in the existing repository. An agent can then implement the requested behavior correctly yet fail solely because it chose a different name. Recovering these unstated interface choices introduces *guesswork* into an evaluation intended to measure software implementation ability.

To reduce this guesswork, several benchmarks augment task descriptions with information derived from reference implementations and test patches (Deng et al., 2025a; Li et al., 2025; Deng et al., 2025b; Badertdinov et al., 2026). We refer to these additions as *hints*. Figure 1 provides an overview of how hints are generated and used in repository-level coding benchmarks. An agent receives a task description $d_{\text{issue}}$ and existing code and tests $(c_{\text{old}}, t_{\text{old}})$, and produces code $c_{\text{gen}}$ evaluated against held-out tests $t_{\text{new}}$. A hint generator additionally accesses the reference code $c_{\text{new}}$ and held-out tests to produce $d_{\text{hints}}$, an extra input to the agent. Hints can expose details that an agent needs to satisfy the evaluation, but the reference implementation also contains design decisions that the benchmark may intend the agent to make independently. Disclosing these decisions introduces *spoilers*, i.e., information that supplies part of the implementation work being evaluated. Prior work documents solution leakage through issue reports and comments as a major problem for SWE-bench (Aleithan et al., 2024). Therefore, identifier hint construction faces a tension between *under-*



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.

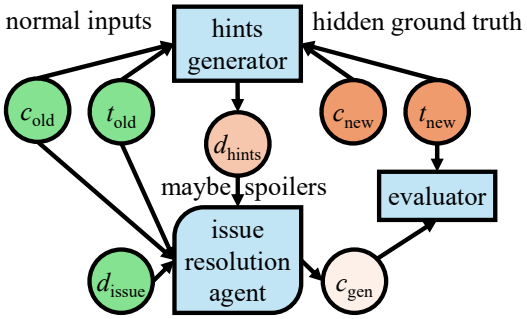


Figure 1: To hint (perchance spoilers) or not to hint (guesswork): that is the question for repository-level coding benchmarks.

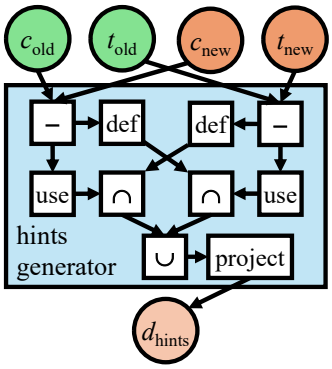


Figure 2: DefUse hints generation pipeline.

*specification*, which leaves agents guessing about hidden constraints, and *over-specification*, which leaks parts of the solution. Higher solve rates alone cannot distinguish between resolving guesswork and supplying spoilers. Our goal is to study identifier hint constructions that clarify the interface required by the tests while leaving implementation choices to the agent.

**A design space for hints.** We consider hint construction through two design decisions: *identifier selection* and *identifier projection*. Selection determines *which* program entities are disclosed, such as file changes, functions edits, etc. Identifier projection asks *how much* is revealed about each selected entity. For instance, if a function identifier is selected, then its name, its signature, its location, or its semantics descriptions are all valid information to be considered being revealed. Existing hint schemes combine different choices along both dimensions (Deng et al., 2025a; Li et al., 2025; Deng et al., 2025b), making it difficult to attribute performance differences to either choice when comparing complete schemes. Separating selection from projection allows us to examine these choices through controlled comparisons.

Guided by this distinction, this paper proposes **DefUse** for identifier selection and **Declaration** for identifier projection. DefUse targets identifiers through which the implementation and its tests interact. Declaration reveals interface attributes while omitting semantic descriptions and implementation guidance. Together, these policies aim to reduce *guesswork* about the required identifiers while limiting *spoilers* about how to implement it.

We compare selection and projection policies across five repository-level coding benchmarks spanning Python and Java. We complement these comparisons with hint attribute ablations and an LLM-based assessment on SWE-bench Pro, examining both task resolution and disclosure of reference-solution information. Our findings indicate that performance gains can accompany greater disclosure of implementation choices, motivating evaluation beyond solve rates alone. Together, these findings motivate combining DefUse selection with Declaration projection to reduce *guesswork* about required interfaces while limit *spoilers* about implementation choices.

**Contributions.** This paper makes three contributions:

- It characterizes identifier hints through selection and projection, providing a common framework for comparing existing benchmark policies.
- It introduces DefUse and Declaration, two policies designed to expose test-facing interfaces while limiting additional reference-derived information.
- It reports controlled comparisons and attribute ablations across five benchmarks and three language models, examining how hint construction affects measured task resolution. Additionally, it adopts an LLM judge to assess information necessity, solution disclosure, and behavioral guidance.

To the extent that benchmarks drive AI progress, we hope that finding the right balance between spoilers and guesswork in coding benchmarks will ultimately contribute to more reliable coding assistants.

Table 1: What each hint scheme discloses. DefUse is ours improved from Doc2Feat. Doc2Feat means the identifier hints extraction used in the Doc2Feat benchmark. FEA-Bench presents two hint selection: brief and detailed. In SWE Bench Pro, the authors assign "Requirements" as the essential hints, and make "Interface" optional.

| | **Selection** | | **Projection** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Hint scheme** | Author | Rule | Name | Type | Path | Signature | Semantics | Behaviour | Document Change |
| DefUse | auto | def–use | ✓ | ✓ | new files | ✓ | — | — | — |
| Doc2Feat | auto | new ∩ test | ✓ | ✓ | new files | — | — | — | — |
| FEA-Bench Brief | auto | new defs | ✓ | ✓ | ✓ | ✓ | — | — | — |
| FEA-Bench Detailed | auto | new defs | ✓ | ✓ | ✓ | ✓ | docstring | — | ✓ |
| SWE-bench Pro Interface | human | manual | ✓ | ✓ | ✓ | ✓ | ✓ | — | — |
| SWE-bench Pro Interface + Requirements | human | manual | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | — |

## 2 Related Work

**Benchmark validity.** Audits of SWE-bench and SWE-bench Pro document underspecified tasks and tests enforcing unstated implementation details (OpenAI, 2024; 2026a;b). SWE-Bench+ identifies solutions disclosed in issue reports or comments, alongside inadequate tests (Aleithan et al., 2024). Differential testing and manual analysis further show that passing benchmark tests does not guarantee patch correctness (Wang et al., 2026). Evidence of memorization raises concerns about training-data contamination (Liang et al., 2026), which LiveCodeBench and SWE-rebench seek to mitigate by collecting new tasks (Jain et al., 2025; Badertdinov et al., 2026). Our study complements these audits by examining how reference-derived hints affect the information available to agents at evaluation time.

**Task specification and hints.** Khatib et al. (2026) examine how bug-report content affects repair success, including controlled ablations of benchmark-added requirements and interfaces. Ganhotra (2026) identifies *hidden naming contracts*, where tests require reference-solution identifiers absent from the issue and base repository. Benchmarks address this ambiguity with reference-derived names, declarations, or requirements (Deng et al., 2025a; Li et al., 2025; Deng et al., 2025b). Within agent workflows, SpecRover infers code intent (Ruan et al., 2025), REAgent refines requirements (Kuang et al., 2026), and HULA incorporates developer feedback into plans and code (Takerngsaksiri et al., 2025). Interactive approaches address missing requirements through clarification: Ambig-SWE evaluates agents' ability to detect underspecification and elicit missing information (Vijayvargiya et al., 2026). LHAW systematically removes task information and measures its effects through agent trials (Pu et al., 2026). Our study separates *identifier selection* from *identifier projection* to examine which benchmark disclosures reduce *guesswork* about required interfaces and which supply *spoilers* about implementation choices.

## 3 Methodology

We characterize identifier hints through two decisions: *Identifier Selection*, which determines which program entities are disclosed, and *Identifier Projection*, which determines which attributes of each selected entity are revealed. Table 1 organizes existing hint schemes along these dimensions and situates our proposed selection policy, DefUse, within this design space. This decomposition supports controlled comparisons of selection policies (under a fixed projection) and projection policies (under a fixed selection).

### 3.1 IDENTIFIER SELECTION: DEFUSE

Various identifier selection pipelines exist in the current repo-level coding benchmarks. The SWE-Bench Pro authors employ human engineers to construct requirements and interface, while FEA-Bench includes all new components present in the new files between commits (Deng et al., 2025b; Li et al., 2025). The Doc2Feat pipeline first extracts identifiers that appear after new code patches and new test patches, then selects identifiers from the intersection of those extracted entities between code and test patches (Deng et al., 2025a). Although Doc2Feat has the strictest identifier selection across available pipelines, occurrence in both changes does not distinguish where an identifier is defined from where it is used, leading to including irrelevant information such as field accesses and general imports.

Thus, we propose DefUse, where the selected identifiers are defined and used. Figure 2 demonstrates DefUse pictorially. If first uses the difference (aka. the patch) between old and new to find newly introduced names: $c_{\text{new}} - c_{\text{old}}$ and $t_{\text{new}} - t_{\text{old}}$. Then, DefUse keeps only those names that carry a communication between code and tests, in one of two directions:

1. $\text{def}(c_{\text{new}} - c_{\text{old}}) \cap \text{use}(t_{\text{new}} - t_{\text{old}})$: The code defines the name and the test uses it; $\cup$
2. $\text{def}(t_{\text{new}} - t_{\text{old}}) \cap \text{use}(c_{\text{new}} - c_{\text{old}})$: The test defines the name and the code uses it.

These are the names through which the solution and its tests communicate. The held-out tests have already committed to them, and the agent cannot recover them from the issue alone.

**Why defined and used?** The criterion follows from what a held-out test can actually observe. A test rejects a functionally correct patch only through the identifiers it references in that patch: the functions it imports and calls, the classes it instantiates, the attributes and return structures it inspects. This yields a necessity argument and a sufficiency argument for DefUse.

**Necessity.** If the test patch reads an identifier that the code patch defines (scenario 1), then the identifier's name is an unforced design choice of the reference author, yet the test treats it as a hard constraint. Every such identifier is a potential spurious failure: an agent can implement the required behaviour perfectly and still fail on an exact-match comparison against an arbitrary label. These identifiers must therefore be disclosed, or the benchmark scores guesswork rather than capability. The symmetric direction (scenario 2) is a contract in the same sense: the test constructs an input the code must consume, and the code-side name is equally unrecoverable and equally load-bearing.

**Sufficiency.** Conversely, an identifier that is not both defined in one patch and used in the other never crosses the code–test boundary. The agent may name it, place it, or eliminate it freely without affecting any test outcome. Disclosing it repairs no ambiguity, where it potentially leaks the reference author's internal design decisions, which is precisely the spoilers we aim to avoid. DefUse is thus the minimal selection that removes name-mismatch artifacts. Every selected identifier can cause a false failure if hidden, and no discarded identifier can.

The goal of identifier hints is to address ambiguity in the current coding tasks. We aim to provide agents as few hints as possible to avoid unnecessary information leakage because exposing too much information will make the problems lose the functionality of evaluating coding agents' true capabilities. DefUse follows this principle; it discloses the full interface contract and nothing else.

### 3.2 IDENTIFIER PROJECTION: DECLARATION

Declaration reveals, for each selected identifier, what its declaration line states and nothing more: its name; its kind (function, class, variable, or file); the path of the file that declares it; whether that file is newly added; and, for callables, its signature. The principle is that a hint should reproduce what a colleague could read off a header or a stub, namely identity, kind, location, and invocation, and stop where the implementation begins. For example, for the SWE-bench Pro task that adds locale detection to Ansible, the Declaration entry for the new function reads

```
### `get_best_parsable_locale` (function)
    -- `lib/ansible/module_utils/common/locale.py` [new file]
Signature: `get_best_parsable_locale(module, preferences=None)`
```

which fixes the name the hidden tests import, where it must live, and how it is called, and leaves everything else to the agent.

Declaration deliberately omits the semantic attributes that richer schemes attach to each entity: a one-line summary of what the entity does and a paragraph describing how. For the same function, the full hint goes on to name the shell command to run (`locale -a`), the locales to prefer (`C.utf8`, `en_US.utf8`), and the exception to raise, and the complete hint for this task runs to thirteen times the length of the Declaration one. These descriptions are produced by a language model that is shown the reference patch, so they are solution-derived by construction; the open question is how much of the solution survives into the text. We answer it with a language-model judge in Section 4.6: the structural attributes carry no solution content, whereas the semantic ones describe behaviour and mechanism. The ablation in Section 4.5 shows that structural attributes deliver a resolve-rate gain at no disclosure, while the further gain from the semantic ones is bought with disclosed solution content. Declaration is therefore the largest projection we can justify as removing guesswork without supplying spoilers.

# 4 Experiments

Our experiments examine how hint construction affects task resolution and what information it reveals about the reference solution. We first isolate the effects of identifier selection and projection by varying one while holding the other fixed (RQ1 & RQ2). Then, we examine the attributes underlying these policies through ablations and an LLM-based assessment (RQ3 & RQ4). Together, these experiments assess the rationale for combining DefUse selection with Declaration projection. We also repeat the selection and projection comparisons on Java to validate whether the patterns observed in Python extend to a second language.

- **RQ1** (selection, Section 4.3): For which identifiers should a hint expose information?
- **RQ2** (projection, Section 4.4): What information should a hint expose for each identifier?
- **RQ3** (ablation, Section 4.5): Which attributes of a hint carry its effect on task resolution?
- **RQ4** (judge, Section 4.6): Which attributes of a hint reveal the reference solution?

## 4.1 Implementation Details

Our pipeline extracts, for every instance, the identifiers introduced or touched by the reference patch and the test patch together with their attributes, and renders a hint from a selection rule and a projection. To validate the extraction we reimplemented the published hint pipelines of Doc2Feat-bench (Deng et al., 2025a) and FEA-Bench (Li et al., 2025) and reproduced their released hints, with minor bug fixes. We also implemented our version of the SWE-Bench Pro hints pipeline by first extracting desired interfaces, and then using Claude Haiku 4.5 (Anthropic, 2025) to generate requirements and the interface descriptions. Details available in Appendix A.

## 4.2 Settings

**Benchmarks, evaluation subsets, and selection statistics.** We use four Python benchmarks: Doc2Feat-bench (Deng et al., 2025a), FEA-Bench (Li et al., 2025), SWE-bench Pro (denoted as Pro in tables) (Deng et al., 2025b), and SWE-rebench (Badertdinov et al., 2026). We additionally evaluate Java tasks from Multi-SWE-bench (denoted as MSB Java) (Zan et al., 2026). For the selection comparisons (RQ1), we retain instances for which both the Doc2Feat and FEA-Bench Brief selection policies produce non-empty identifier sets. All selection policies are evaluated on this shared subset, comprising 31, 40, 131, and 26 instances from the four Python benchmarks, respectively, and 40 Java instances. For the projection comparisons (RQ2), we retain instances for which DefUse produces a non-empty identifier set. These subsets contain 21, 44, 68, and 14 instances from the four Python benchmarks, respectively, and 29 Java instances. The attribute ablations (RQ3) and LLM-based assessment (RQ4) focus on the same 68-instance SWE-bench Pro subset. Table 2 summarizes the number of identifiers disclosed by each selection policy. DefUse discloses the fewest identifiers per instance across all five benchmarks, while Pro discloses the most.

Table 2: Number of identifiers disclosed per selection method, counted on the same instances as the selection tables under the Declaration projection. *Total* is the number of identifiers over all instances and *Mean* the number per instance. DefUse is empty on the instances whose tests reference no new identifier: it is non-empty on 20/31, 44/142, 66/131, 9/26 and 29/40 instances, with 3.2, 1.8, 2.4, 2.0 and 6.7 identifiers per non-empty instance. The other selections are non-empty on every instance.

| Select | Doc2Feat (31) | | FEA Bench (142) | | Pro (131) | | SWE Rebench (26) | | MSB Java (40) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Total | Mean | Total | Mean | Total | Mean | Total | Mean | Total | Mean |
| Empty | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 |
| DefUse | 64 | 2.1 | 79 | 0.6 | 158 | 1.2 | 18 | 0.7 | 193 | 4.8 |
| Doc2Feat | 173 | 5.6 | 500 | 3.5 | 635 | 4.8 | 106 | 4.1 | 313 | 7.8 |
| FEA | 337 | 10.9 | 449 | 3.2 | 1004 | 7.7 | 110 | 4.2 | 453 | 11.3 |
| Pro | 594 | 19.2 | 1882 | 13.3 | 2251 | 17.2 | 418 | 16.1 | 912 | 22.8 |

**Models and agent.** We run three language models through the same harness: DeepSeek V4 Flash (0431), MiniMax M2.5, and GPT 5.6 Luna (Xu et al., 2026; MiniMax, 2026; OpenAI, 2026). All models use temperature 1.0, with maximum reasoning effort for DeepSeek and MiniMax and high reasoning effort for GPT 5.6 Luna. We use mini-SWE-agent (Yang et al., 2024), which issues one shell command per turn, with a limit of 250 steps per trajectory. Due to budget constraints, we run one trajectory per instance for each configuration.

**Hint attributes.** We parse each patch to extract the structural attributes, namely name, kind, path, new-file status and signature. The two semantic attributes mirror the human-written fields of SWE-bench Pro: *weak semantics* corresponds to the description under its `interface`, and *strong semantics* to its `requirements`. Both are generated by Claude Haiku 4.5 (Anthropic, 2025) given the reference patch, and both appear only in the Pro projection.

**Evaluation and metrics.** We evaluate each benchmark using its official evaluator in Docker containers. An instance counts as resolved only if all of its fail-to-pass and pass-to-pass tests pass, and we report the resolve rate, the percentage of resolved instances.

### 4.3 Identifier Hint Selection

RQ1 examines which identifiers a hint should disclose. We compare four selection policies under two fixed projections: Declaration (Table 3), which exposes structural interface attributes, and Name Only (Appendix B, Table 9), which exposes only identifier names. Because DefUse restricts selection to identifiers through which the implementation and tests communicate (Section 3.1), these comparisons test whether disclosing additional identifiers improves task resolution.

The selection policies differ substantially in how many identifiers they disclose (Table 2). On SWE-bench Pro, DefUse reveals an average of 1.2 identifiers per instance, compared with 4.8 for Doc2Feat, 7.7 for FEA, and 17.2 for Pro. For GPT 5.6 Luna, this substantial difference in disclosure corresponds to a relatively small difference in resolution: DefUse resolves 91 of 131 instances (69.5%), compared with 92–95 (70.2–72.5%) for the broader selections. Thus, for this model and benchmark, the narrower selection substantially reduces disclosure with only a small decrease in observed resolution.

The performance cost of narrower disclosure is larger for the open-source models on SWE-bench Pro. DeepSeek resolves 17 instances with DefUse, compared with 26–29 for the broader selection policies, while MiniMax resolves 20, compared with 21–29. These gains reach 12 and 9 additional instances, respectively, compared with at most 4 for GPT 5.6 Luna. However, the benefit of additional identifiers varies across benchmarks. On FEA-Bench, DeepSeek resolves 16 instances with DefUse, compared with 13–14 for the broader policies, while MiniMax resolves 14, compared with 13–15. On Multi-SWE-bench Java, DeepSeek resolves 13 of 40u instances with DefUse under Declaration, compared with 12–15 for the other policies (Table 4). Doc2Feat-bench remains challenging for both open-source models, with DeepSeek resolving 0–2 instances and MiniMax resolving one under every policy. Together, these results indicate that the benefit of broader disclosure depends on both the model and the benchmark.

Table 3: Model performance across benchmarks and selection methods. We choose Declaration projection as the fixed projection given different identifiers.

| Model | Select | Doc2Feat (31) | | FEA Bench (40) | | Pro (131) | | SWE Rebench (26) | |
|---|---|---|---|---|---|---|---|---|---|
| | | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. |
| DeepSeek V4 Flash | Empty | 0 | 0.0% | 13 | 32.5% | 11 | 8.4% | 11 | 42.3% |
| | DefUse | 1 | 3.2% | 16 | 40.0% | 17 | 13.0% | 9 | 34.6% |
| | Doc2Feat | 0 | 0.0% | 14 | 35.0% | 26 | 19.8% | 2 | 7.7% |
| | FEA | 2 | 6.5% | 13 | 32.5% | 27 | 20.6% | 6 | 23.1% |
| | Pro | 2 | 6.5% | 14 | 35.0% | 29 | 22.1% | 5 | 19.2% |
| MiniMax M2.5 | Empty | 1 | 3.2% | 14 | 35.0% | 8 | 6.1% | 8 | 30.8% |
| | DefUse | 1 | 3.2% | 14 | 35.0% | 20 | 15.3% | 8 | 30.8% |
| | Doc2Feat | 1 | 3.2% | 13 | 32.5% | 21 | 16.0% | 9 | 34.6% |
| | FEA | 1 | 3.2% | 15 | 37.5% | 27 | 20.6% | 13 | 50.0% |
| | Pro | 1 | 3.2% | 15 | 37.5% | 29 | 22.1% | 13 | 50.0% |
| GPT 5.6 Luna | Empty | 16 | 51.6% | 31 | 77.5% | 77 | 58.8% | 18 | 69.2% |
| | DefUse | 17 | 54.8% | 30 | 75.0% | 91 | 69.5% | 18 | 69.2% |
| | Doc2Feat | 18 | 58.1% | 30 | 75.0% | 92 | 70.2% | 20 | 76.9% |
| | FEA | 16 | 51.6% | 27 | 67.5% | 94 | 71.8% | 19 | 73.1% |
| | Pro | 22 | 71.0% | 32 | 80.0% | 95 | 72.5% | 19 | 73.1% |

Table 4: DeepSeek V4 Flash performance on Multi-SWE-bench Java across selection policies under Declaration projection.

| Project | Select | Multi-SWE-bench Java (40) | |
|---|---|---|---|
| | | Resolved | Acc. |
| Declaration | Empty | 3 | 7.5% |
| | DefUse | 13 | 32.5% |
| | Doc2Feat | 15 | 37.5% |
| | FEA | 15 | 37.5% |
| | Pro | 12 | 30.0% |

Table 5: Hint attribute ablation with DeepSeek V4 Flash on SWE-bench Pro (68 instances) under DefUse selection.

| Hint attributes | Pro (68) | |
|---|---|---|
| | Resolved | Acc. |
| Empty | 7 | 10.3% |
| Name Only | 15 | 22.1% |
| Doc2Feat | 14 | 20.6% |
| Declaration | 20 | 29.4% |
| Name Only and Semantics | 29 | 42.6% |
| All Hints | 49 | 72.1% |
| Oracle Hint | 60 | 88.2% |

Broader selection can therefore improve measured resolution, but its usefulness to an agent does not establish that the additional identifiers are necessary to specify the interface required by the tests. Our motivation for DefUse is to expose those test-facing identifiers while limiting disclosure of reference-specific implementation choices.

### 4.4 Identifier Hint Projection

RQ2 examines what information a hint should disclose about each selected identifier. We hold selection fixed at DefUse and compare five projections against an empty baseline (Table 6). This comparison examines how structural interface attributes and additional semantic descriptions affect task resolution.

The results show that structural interface information can help agents resolve tasks. On SWE-bench Pro, Declaration increases resolve rates over the empty baseline from 10.3% to 29.4% for DeepSeek, from 2.9% to 14.7% for MiniMax, and from 55.9% to 73.5% for GPT 5.6 Luna. It also improves over Name Only for all three models on this benchmark, suggesting that attributes beyond identifier names contribute useful interface information. These improvements occur without the semantic descriptions included in the Pro projection. The Java results provide supporting evidence: Declaration increases DeepSeek's resolved instances from 5 to 10 out of 29 (Table 7).

The effect of adding semantic descriptions varies across models. On SWE-bench Pro, MiniMax resolves 16 instances with Pro, compared with 10 under Declaration. DeepSeek resolves 19 with Pro and 20 with Declaration, while GPT 5.6 Luna resolves 39 and 50, respectively. Thus, additional reference-derived descriptions can improve resolution, but their benefit is not consistent across models. These comparisons alone do not establish whether the descriptions help by clarifying requirements or by supplying implementation guidance. We also interpret small differences cautiously, given the single trajectory per instance and the limited sizes of some evaluation subsets.

Table 6: Model performance across benchmarks and projection methods. We choose DefUse selection as the fixed selection given different projections. *Declaration+Doc* adds a one-sentence description of each identifier to Declaration.

| Model | Project | Doc2Feat (21) | | FEA Bench (44) | | Pro (68) | | SWE Rebench (14) | |
|---|---|---|---|---|---|---|---|---|---|
| | | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. |
| DeepSeek V4 Flash | Empty | 0 | 0.0% | 13 | 29.5% | 7 | 10.3% | 4 | 28.6% |
| | NameOnly | 0 | 0.0% | 15 | 34.1% | 15 | 22.1% | 3 | 21.4% |
| | Doc2Feat | 0 | 0.0% | 17 | 38.6% | 14 | 20.6% | 5 | 35.7% |
| | Declaration | 2 | 9.5% | 15 | 34.1% | 20 | 29.4% | 4 | 28.6% |
| | Declaration+Doc | 0 | 0.0% | 15 | 34.1% | 11 | 16.2% | 4 | 28.6% |
| | Pro | 2 | 9.5% | 16 | 36.4% | 19 | 27.9% | 5 | 35.7% |
| MiniMax M2.5 | Empty | 1 | 4.8% | 15 | 34.1% | 2 | 2.9% | 4 | 28.6% |
| | NameOnly | 0 | 0.0% | 18 | 40.9% | 7 | 10.3% | 5 | 35.7% |
| | Doc2Feat | 0 | 0.0% | 14 | 31.8% | 8 | 11.8% | 4 | 28.6% |
| | Declaration | 2 | 9.5% | 15 | 34.1% | 10 | 14.7% | 4 | 28.6% |
| | Declaration+Doc | 0 | 0.0% | 14 | 31.8% | 11 | 16.2% | 3 | 21.4% |
| | Pro | 0 | 0.0% | 17 | 38.6% | 16 | 23.5% | 4 | 28.6% |
| GPT 5.6 Luna | Empty | 12 | 57.1% | 33 | 75.0% | 38 | 55.9% | 7 | 50.0% |
| | NameOnly | 12 | 57.1% | 34 | 77.3% | 37 | 54.4% | 7 | 50.0% |
| | Doc2Feat | 13 | 61.9% | 33 | 75.0% | 37 | 54.4% | 8 | 57.1% |
| | Declaration | 11 | 52.4% | 35 | 79.5% | 50 | 73.5% | 9 | 64.3% |
| | Declaration+Doc | 11 | 52.4% | 34 | 77.3% | 45 | 66.2% | 7 | 50.0% |
| | Pro | 11 | 52.4% | 34 | 77.3% | 39 | 57.4% | 7 | 50.0% |

Table 7: Model performance on Multi-SWE-bench Java across projection methods. We choose DefUse selection as the fixed selection given different projections. *Declaration+Doc* adds a one-sentence description of each identifier to Declaration. Model: DeepSeek V4 Flash.

| Model | Project | Multi-SWE-bench Java (29) | |
|---|---|---|---|
| | | Resolved | Acc. |
| DeepSeek V4 Flash | Empty | 5 | 17.2% |
| | NameOnly | 10 | 34.5% |
| | Doc2Feat | 7 | 24.1% |
| | Declaration | 10 | 34.5% |
| | Declaration+Doc | 9 | 31.0% |
| | Pro | 14 | 48.3% |

Together, these observations support Declaration as a reasonable projection for identifier hints: it provides interface information that can improve task resolution while omitting semantic descriptions derived from the reference solution. The attribute ablations and LLM-based assessment (Sections 4.5 and 4.6) further examine the contributions of individual attributes and the solution information they disclose.

### 4.5 Hint Attribute Ablation

RQ3 examines how different hint attributes contribute to task resolution. We fix selection to DefUse and evaluate DeepSeek V4 Flash on the SWE-bench Pro subset (Table 5). The rows up to *Declaration* successively add structural attributes. *Name Only and Semantics* combines names with the two semantic descriptions, while *All Hints* combines Declaration with both descriptions.

Names improve resolution over the empty baseline, and Declaration provides a further gain. Adding entity type alone to names yields no observed improvement, suggesting that structural attributes differ in their contribution. Semantic descriptions produce larger gains in these runs: combining them with Declaration increases resolution from 29.4% to 72.1%. The combination also outperforms names with semantic descriptions, suggesting that structural and semantic information can provide complementary benefits.

These results distinguish the usefulness of hint attributes from their appropriateness for evaluation. Interface attributes can help agents satisfy the test-facing contract, while semantic descriptions may also supply implementation guidance. Section 4.6 examines this distinction. Each configuration uses a single trajectory per instance, and the recorded variation across runs limits attribution of

Table 8: Hint attribute scores on SWE-bench Pro under DefUse selection. $N$: necessity; $D$: disclosure; $G$: guidance; $\mathrm{HVM} = N - \max(D, G)$. $^*$ marks margins indistinguishable from zero. HVM: Hint Value Margin. Judge details appear in Appendix C.

| **Attribute** | **n** | $N$ | $D$ | $G$ | **HVM** |
|---|---|---|---|---|---|
| *Structural (interface-level)* | | | | | |
| Signature | 31 | 2.45 | 0.00 | 0.00 | $+2.45$ |
| Name | 160 | 1.88 | 0.00 | 0.11 | $+1.76$ |
| Entity type | 160 | 1.54 | 0.00 | 0.00 | $+1.54$ |
| Is new file | 58 | 0.62 | 0.00 | 0.00 | $+0.62$ |
| Path | 58 | 1.19 | 0.00 | 1.00 | $+0.19^*$ |
| *group mean* | 467 | 1.56 | 0.00 | 0.16 | $\mathbf{+1.39}$ |
| *Semantic (behaviour-level)* | | | | | |
| Weak semantics | 160 | 1.19 | 1.09 | 0.29 | $+0.03^*$ |
| Strong semantics | 160 | 1.82 | 1.81 | 1.68 | $-0.13^*$ |
| *group mean* | 320 | 1.50 | 1.45 | 0.99 | $\mathbf{-0.05}$ |

the observed differences solely to hint content. The semantic configurations also use a different rendering from Pro in RQ2, so their absolute rates should not be directly compared with Table 6.

### 4.6 Hint Leakage Judge

RQ4 examines what information hint attributes disclose, complementing the task-resolution experiments. Following prior work on LLM-based evaluation (Zheng et al., 2023; Liu et al., 2023), we use GPT 5.6 Luna with reasoning effort medium to assess attributes on SWE-bench Pro. The rubric scores necessity $N$ (unrecoverability without the reference solution), disclosure $D$ (solution content), and guidance $G$ (direction of the agent's actions). We summarize these ordinal scores using the heuristic hint value margin $\mathrm{HVM} = N - \max(D, G)$. The rubric and assessment procedure appear in Appendix C.

Table 8 shows similar average necessity scores for structural and semantic attributes, but greater disclosure and guidance for semantic descriptions. Names and signatures receive the highest margins, whereas the semantic attributes have margins near zero. Strong semantics receives higher disclosure and guidance scores than weak semantics, consistent with descriptions that convey implementation mechanisms.

Together with the projection comparison and attribute ablation (Sections 4.4 and 4.5), this assessment supports Declaration as a reasonable projection for identifier hints. The experiments show that structural interface attributes can improve task resolution without semantic descriptions, while the judge associates those descriptions with greater solution disclosure and implementation guidance. Declaration therefore provides a principled boundary for hint construction. It clarifies test-facing interfaces while limiting additional reference-derived information that may supply part of the implementation being evaluated.

## 5 Conclusions and Limitations

We study identifier hints through selection and projection, showing that hint policies affect both measured task resolution and the reference-derived information available to agents. Our findings motivate combining DefUse selection with Declaration projection to clarify test-facing interfaces while limiting implementation guidance. More broadly, benchmark designers should assess hints by what they disclose as well as the performance gains they produce.

**Limitations.** Budget constraints limited each reported configuration to one trajectory per instance, making small differences sensitive to stochastic variation. The recorded experiments represent approximately $2,300 in inference value, including unbilled MiniMax usage valued at public API rates. Small benchmark subsets and a single agent harness limit generalizability, while serving variation and rubric-dependent LLM judgments introduce additional uncertainty.

### AI USE STATEMENT

Generative AI was used in this work in two distinct roles, which we disclose separately.

**AI as a component of the studied method.** Two language models are part of the experimental pipeline rather than of the writing process. The *weak* and *strong semantics* attributes of every hint were generated by a language model (Claude Haiku 4.5) that was shown the reference patch, so these attributes are synthetic, solution-derived text by construction; this is stated where the attributes are introduced and is the reason they are studied as a potential source of over-specification. The attribute-level necessity, disclosure and guidance scores in the leakage analysis were produced by a language model judge (GPT-5.6 Luna, reasoning effort fixed at *medium*, median of three samples), following a rubric described in the paper; every judgment records the serving provider, finish reason and token usage so the run can be audited. The coding agents under evaluation are themselves based on language models, as in all prior work on these benchmarks.

**AI as a research assistant.** We used generative AI tools to implement methods, including the hint extraction and projection code. We have not used generative AI tools to develop theoretical models, to formulate or prove mathematical claims, or for translation, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of this paper for readability, to summarise and identify relevant literature, and to suggest experimental parameters.

We have reviewed all AI-assisted work. All AI-generated code was read and tested by the authors; every quantitative result reported here was recomputed from the stored evaluation outputs; and the judge was validated against mechanically measured ground truth on the one attribute where such truth exists before being applied to the rest. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

We will open-source the code repository of this paper, including the hint extraction and projection pipeline, the DefUse selection and Declaration projection implementations, and the baseline implementations as well. We will also release the hints extracted and generated from our pipelines.

## REFERENCES

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024.

Anthropic. Introducing Claude Haiku 4.5, October 2025. URL https://www.anthropic.com/news/claude-haiku-4-5.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Le Deng, Zhonghao Jiang, Jialun Cao, Michael Pradel, and Zhongxin Liu. Nocode-bench: A benchmark for evaluating natural language-driven feature addition. *arXiv preprint arXiv:2507.18130*, 2025a.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. SWE-bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025b.

Jatin Ganhotra. Hidden naming contracts in SWE-agent benchmarks, April 2026. URL https://jatinganhotra.dev/blog/swe-agents/2026/04/05/hidden-naming-contracts-in-swe-agent-benchmarks.html.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Lara Khatib, Noble Saji Mathews, Meiyappan Nagappan, Pengyu Nie, and Thomas Zimmermann. What makes a good bug report for an ai agent? *arXiv preprint arXiv:2607.07593*, 2026.

Shiqi Kuang, Zhao Tian, Kaiwei Lin, Chaofan Tao, Shaowei Wang, Haoli Bai, Lifeng Shang, and Junjie Chen. REAgent: Requirement-driven llm agents for software issue resolution. *arXiv preprint arXiv:2604.06861*, 2026.

Wei Li, Xin Zhang, Zhongxin Guo, Shaoguang Mao, Wen Luo, Guangyue Peng, Yangyu Huang, Houfeng Wang, and Scarlett Li. FEA-Bench: A benchmark for evaluating repository-level code generation for feature implementation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17160–17176, 2025.

Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-bench illusion: When state-of-the-art llms remember instead of reason. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, pp. 395–405, 2026.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023.

MiniMax. MiniMax M2.5: Built for real-world productivity, February 2026. URL https://www.minimax.io/news/minimax-m25.

OpenAI. Introducing SWE-bench Verified, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

OpenAI. GPT-5.6 Luna: Frontier Intelligence that scales with your ambition, July 2026. URL https://openai.com/index/gpt-5-6/.

OpenAI. Separating signal from noise in coding evaluations, July 2026a. URL https://openai.com/index/separating-signal-from-noise-coding-evaluations/.

OpenAI. Why SWE-bench verified no longer measures frontier coding capabilities, February 2026b. URL https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/.

George Pu, Michael S Lee, Udari Madhushani Sehwag, David J Lee, Bryan Zhu, Yash Maurya, Mohit Raghavendra, Yuan Xue, and Samuel Marc Denton. LHAW: Controllable underspecification for long-horizon tasks. *arXiv preprint arXiv:2602.10525*, 2026.

Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. SpecRover: Code intent extraction via llms. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 963–974. IEEE, 2025.

Wannita Takerngsaksiri, Jirat Pasuksmit, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Ruixiong Zhang, Fan Jiang, Jing Li, Evan Cook, Kun Chen, and Ming Wu. Human-in-the-loop software development agents. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 342–352. IEEE, 2025.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. AmbigSWE: Interactive agents to overcome underspecificity in software engineering. In *International Conference on Learning Representations*, volume 2026, pp. 146716–146737, 2026.

You Wang, Michael Pradel, and Zhongxin Liu. Are "solved issues" in SWE-bench really solved correctly? an empirical study. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 169–181, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, et al. Multi-SWE-bench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

# A Implementation Details: Baselines

## A.1 Selection

For both Doc2Feat and FEA Bench Brief hint generation, we adapted the original code so that the pipeline can extract hints on various benchmarks. For SWE Bench Pro hint generation, since they employ human engineers to build the hints, it's impossible to completely replicate Pro's pipeline. Thus, we use a LLM to approximate human engineers in terms of providing detailed hint text. We first find the different identifiers between old code and new cold patches, extract the information about identifiers, and then have Claude Haiku 4.5 to generate interface descriptions and requirements with few shot prompts. Comparing the generated interface descriptions and requirements with original ones in python subset of SWE Bench Pro (266 instances), the generated ones achieved consine similarity of $0.8978$ and $0.8405$ respectively, using Qwen3-0.6-Embedding (Zhang et al., 2025) as the embedding model.

## A.2 Projection

To implement controllable projection, we first have a complete pipeline that extracts all possible identifiers from both the code patches and the test patches. For each identifier, we extract relevant information by checking its abstract syntax tree to get the information not just about the identifier name. Then, we select DefUse as our selection by performing the communication between code and test patches, and the we adopt Claude Haiku 4.5 to enrich the semantics, where weak semantics corresponds to SWE Bench Pro interface descriptions, and strong semantics corresponds to requirements. Then:

- Doc2Feat: the name and entity type of each identifier, rendered as a flat list; no path, signature or description.
- Declaration: name, entity type, file path, a new-file flag and the signature; class members are nested under their class. No natural-language text.
- Declaration+Doc: Declaration plus the weak semantics, a one-sentence (at most 20 words) description of what the entity is and does. It corresponds to the docstring that FEA-Bench's detailed hints add, without their code; FEA-Bench's brief hints carry no description and coincide with Declaration.

- Pro: Declaration+Doc plus the strong semantics, a two-to-four sentence paragraph on behaviour, edge cases and test-implied constraints. Both descriptions are written by Claude Haiku 4.5 from the reference patch, so the LLM acts as the extractor that SWE-bench Pro's human annotators were.

## B IDENTIFIER SELECTION UNDER NAME ONLY PROJECTION

We repeat the identifier selection comparison with the projection fixed at Name Only. Each hint reveals only the names of the selected identifiers, allowing us to examine selection policies without disclosing additional attributes. Tables 9 and 10 report the Python and Java results, respectively.

The effect of selection varies across models and benchmarks even when hints contain only names. On SWE-bench Pro, Pro and FEA outperform DefUse for all three models, whereas DefUse leads the hint selections for DeepSeek on FEA-Bench. The Java results provide a further example: DefUse and Pro each resolve 9 of 40 instances, compared with 6 for Doc2Feat and FEA. Thus, broader selection can help, but disclosing additional names does not consistently improve resolution. As in the Declaration comparison, small differences should be interpreted cautiously given the limited evaluation subsets and single trajectory per instance.

Table 9: Model performance across benchmarks and selection methods. We choose NameOnly Bench projection as the fixed projection given different identifiers.

| Model | Select | Doc2Feat (31) | | FEA Bench (40) | | Pro (131) | | SWE Rebench (26) | |
|---|---|---|---|---|---|---|---|---|---|
| | | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. | Resolved | Acc. |
| DeepSeek V4 Flash | Empty | 0 | 0.0% | 13 | 32.5% | 14 | 10.7% | 12 | 46.2% |
| | DefUse | 2 | 6.7% | 16 | 40.0% | 17 | 13.0% | 11 | 42.3% |
| | Doc2Feat | 1 | 3.3% | 14 | 35.0% | 17 | 13.0% | 11 | 42.3% |
| | FEA | 0 | 0.0% | 13 | 32.5% | 20 | 15.3% | 11 | 42.3% |
| | Pro | 1 | 3.3% | 14 | 35.0% | 20 | 15.3% | 12 | 46.2% |
| MiniMax M2.5 | Empty | 1 | 3.3% | 14 | 35.0% | 12 | 9.2% | 8 | 30.8% |
| | DefUse | 2 | 6.7% | 14 | 35.0% | 15 | 11.5% | 8 | 30.8% |
| | Doc2Feat | 1 | 3.3% | 13 | 32.5% | 16 | 12.2% | 10 | 38.5% |
| | FEA | 0 | 0.0% | 15 | 37.5% | 20 | 15.3% | 13 | 50.0% |
| | Pro | 0 | 0.0% | 15 | 37.5% | 20 | 15.3% | 10 | 38.5% |
| GPT 5.6 Luna | Empty | 21 | 66.7% | 31 | 77.5% | 85 | 64.9% | 19 | 73.1% |
| | DefUse | 22 | 70.0% | 30 | 75.0% | 83 | 63.4% | 19 | 73.1% |
| | Doc2Feat | 21 | 66.7% | 30 | 75.0% | 88 | 67.2% | 19 | 73.1% |
| | FEA | 17 | 53.3% | 27 | 67.5% | 88 | 67.2% | 20 | 76.9% |
| | Pro | 16 | 50.0% | 32 | 80.0% | 93 | 71.0% | 18 | 69.2% |

## C HINT LEAKAGE JUDGE DETAILS

**Rubric.** Each attribute is assessed on three ordinal scales. Necessity $N$ measures how recoverable its value is without the reference solution: 0 if stated in the issue, 1 if findable in the repository, 2 if constrained by convention, and 3 if freely chosen by the reference author. Disclosure $D$ measures solution content: 0 for identity and addressing, 1 for the observable contract, 2 for an implementation mechanism, and 3 for a transcription. Guidance $G$ measures direction of the agent's actions: 0 for declarative information, 1 for orientation, 2 for procedural guidance, and 3 for prescriptive instructions. Disclosure and guidance are distinct: a file path can orient the agent without describing an implementation. The heuristic margin $\mathrm{HVM} = N - \max(D, G)$ combines these judgments; it is not a calibrated measure of information quantity.

**Assessment procedure.** We assess 787 identifier–attribute pairs across 160 identifiers selected by DefUse on 68 SWE-bench Pro instances. Necessity is assessed using the issue without showing the reference or test patches; disclosure and guidance are assessed with both patches available. Each call presents all attributes of an identifier and specifies one target attribute, so the necessity judgment is conditional on the other visible attributes. The judge is GPT 5.6 Luna with medium reasoning effort. It returns a category label, an evidence span, and a rationale. We map labels to integers and take the median of three samples at temperature 1. We estimate uncertainty using instance-clustered bootstrap and measure consistency using three-sample unanimity.

Table 10: DeepSeek V4 Flash performance on Multi-SWE-bench Java across selection policies under Name Only projection.

| Project | Select | Multi-SWE-bench Java (40) | |
|---|---|---|---|
| | | Resolved | Acc. |
| Name Only | Empty | 3 | 7.5% |
| | DefUse | 9 | 22.5% |
| | Doc2Feat | 6 | 15.0% |
| | FEA | 6 | 15.0% |
| | Pro | 9 | 22.5% |

**Validation and limitations.** We check repository membership for a subset of identifier names using `git grep` at the base commit. When supplied with this evidence, the judge reproduces the repository-membership distinction for all 31 identifiers checked, compared with 18 without it. This check validates a limited aspect of the necessity assessment; it does not validate the full rubric. Agreement also varies across attributes: three-sample unanimity is 94% for signatures but 41–42% for semantic descriptions. These differences indicate greater judgment uncertainty for semantic content and should be considered alongside the aggregate scores.

**Rubrics and prompts.** The three blocks below reproduce, verbatim from the judge implementation, the rubric and the prompt skeleton behind each column of Table 8. Necessity is one call per (instance, identifier, attribute) triple that sees the issue but neither patch; disclosure and guidance are rated together in a second call that also sees the reference and test patches. Angle-bracketed fields are filled per call. The judge returns category labels only; the integers in the table are assigned in code (`REDUNDANT` $\to 0, \ldots,$ `LOAD-BEARING` $\to 3$, and likewise for the other two scales). All three calls share the system prompt shown at the top of the first block.

**Necessity ($N$): rubric and solution-blind prompt**

```
[system]
You are an exacting research assistant evaluating hint quality for a software
engineering benchmark. You apply a rubric literally and consistently. You do not
reward values for being detailed or useful; you classify them by the stated
criteria only. You always answer with a single JSON object and nothing else.

[user]
Rate NECESSITY: could a competent engineer obtain this attribute's value WITHOUT
seeing the reference solution?

  0 REDUNDANT      - the value is stated in the problem statement itself.
  1 SEARCH-SAVING  - not stated, but an engineer with the repository at its current
                     commit would reach it through normal investigation (the entity
                     already exists; grep/reading finds it). The hint saves effort,
                     not information.
  2 DISAMBIGUATING - not stated and not already present, but strongly constrained by
                     repository convention (e.g. a new test file's location follows
                     an obvious existing layout). An engineer would likely guess it.
  3 LOAD-BEARING   - encodes a free choice the solution's author made that is not
                     recoverable from the problem statement or the existing codebase
                     (e.g. the name of a function that does not yet exist). Without
                     the hint an engineer would pick something else.

Judge ONLY the target attribute's value. Ignore whether the value seems important;
necessity is about RECOVERABILITY, not importance.

## Measured repository fact            (present only when the target is `name`)
The identifier `<name>` appears in <k> file(s) in this repository at the commit
the agent starts from.
<k > 0: "An engineer could therefore find it by searching the existing code."
 k = 0: "It does not exist anywhere in the tree yet, so it cannot be found by
         searching; any engineer would have to choose a name.">

## Repository
<owner/repo>

## Problem statement (this is ALL the agent is told)
<issue text, first 6000 characters>

## The identifier, for context
```

```
  name: <...>
  entity_type: <...>
  path: <...>                 <<< TARGET      (one attribute is marked)
  is_new_file: <...>
  signature: <...>
  weak_semantics: <...>
  strong_semantics: <...>

## The value you are rating
attribute : `<target attribute>`
value     : ```
<the target attribute's value, alone>
```

You are rating THE TEXT ABOVE - not the entity it refers to, and not the other
attributes listed for context. Ask what this text, by itself, tells a reader.

## Your task
Rate NECESSITY.

Respond with exactly this JSON object:
{"label": "<REDUNDANT|SEARCH-SAVING|DISAMBIGUATING|LOAD-BEARING>",
  "evidence": "<a short verbatim quote from the problem statement that justifies
your rating, or the empty string if the statement says nothing relevant>",
  "rationale": "<one sentence>",
  "confidence": <0.0-1.0>}
```

**Disclosure ($D$): rubric**

```
Rate DISCLOSURE: how much of the reference solution's CONTENT does this value carry?

  0 INTERFACE      - identity and addressing only: what it is called, where it lives,
                     how it is invoked. This is the line the benchmark itself
                     publishes to agents, so it is not leakage.
  1 CONTRACT       - observable behaviour: what it does, what its inputs and outputs
                     mean. A specification someone could implement many ways.
  2 MECHANISM      - how it does it: control flow, algorithm, ordering of operations,
                     or the names of specific helpers/APIs it calls internally.
  3 TRANSCRIPTION  - restates the solution closely enough that the code could be
                     transliterated from the text with little independent thought.

Example of 2: "overrides add_job to check the hostname against allowed hosts using
fnmatch patterns" - names the method and the matching library.
Example of 1: "A set of exact key names that should not be sanitized even if they
contain sensitive substrings" - states the contract without the mechanism.
```

**Guidance ($G$): rubric, and the solution-aware prompt shared with $D$**

```
Rate BEHAVIOURAL GUIDANCE: how much does this value DIRECT the agent's actions, as
opposed to merely stating a fact it must act upon?

  0 DECLARATIVE    - states that something exists or what it is called. No action
                     is implied.
  1 ORIENTING      - narrows where to look or what to touch, without saying what to do.
  2 PROCEDURAL     - implies a sequence of steps or a specific approach to take.
  3 PRESCRIPTIVE   - reads as an instruction the agent can follow directly; the text
                     could substitute for a plan.

This is INDEPENDENT of disclosure. A line range discloses a lot (exactly where the
change goes) but guides nothing (it does not say what to do). An algorithm
description does both. Score them separately.

--------------------------------------------------------------------------------
[user message of the D/G call: the DISCLOSURE rubric, then the GUIDANCE rubric,
 then the following]

## Problem statement
<issue text, first 4000 characters>

## Reference solution (gold patch)
```diff
<gold patch, first 14000 characters>
```

## Test patch (defines what must pass)
```
```

```
```diff
<test patch, first 6000 characters>
```

## The identifier, for context
<all attributes of the identifier, target marked "<<< TARGET", as above>

## The value you are rating
<the target attribute's value, alone, as above>

## Your task
Rate DISCLOSURE and BEHAVIOURAL GUIDANCE for the value above, by comparing it
against the reference solution.

Respond with exactly this JSON object:
{"disclosure_label": "<INTERFACE|CONTRACT|MECHANISM|TRANSCRIPTION>",
  "guidance_label": "<DECLARATIVE|ORIENTING|PROCEDURAL|PRESCRIPTIVE>",
  "evidence": "<the span of the target value that carries the most solution content,
verbatim>",
  "rationale": "<one sentence>",
  "confidence": <0.0-1.0>}
```