CARRYBENCH: A MULTI-ROLE, MULTI-MAINTAINER BENCHMARK FOR THE MAINTAINABILITY OF AGENT-GENERATED CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent-generated code is often the starting point for later work. Functional correctness tests show whether the current task is done. They do not show how difficult the next task will be. Existing maintainability benchmarks usually measure one downstream activity, and most let an agent work on its own output, so the score mixes the artifact with the worker. CARRYBENCH separates the two. Nine producer models write repository-scale programs for 13 cases in Python, Go, Rust, and C++, and frozen outputs are evaluated on three maintenance demands. Each demand is a targeted downstream task under three fixed maintainer profiles. An observer predicts behavior from source for Analysability, a consumer builder constructs a new use for Reusability, and a tester writes a black-box test suite for Testability. Static Modularity is measured on the same source as a traditional proxy. Rankings agree most across maintainers for Reusability, and less for Analysability and Testability. Different downstream outcomes produce different producer orderings. Systematic within-case analysis shows that higher state cohesion need not coincide with easier downstream work. Ablations of prespecified source-feature blocks find conditional associations with lexical organization, observable entry-to-result paths, and behavior localization. Those associations differ across maintenance demands and maintainer profiles. CARRYBENCH treats maintainability as the difficulty a fixed downstream worker experiences on a specified next task over a frozen artifact. It reports that difficulty for each maintenance demand and each maintainer profile, from the executable tasks and the source-level associations, without combining those results into a universal score.

1 INTRODUCTION

Large language models now generate repository-scale programs, and the standard way to evaluate them is functional correctness: compile the program, run a test suite, report a pass rate (Chen et al., 2021; Jimenez et al., 2024; Yang et al., 2026). A pass rate records that the current task is solved. It does not measure how difficult that artifact is for the maintainer who next takes it as input.

Recent maintainability benchmarks address this gap (Zhu et al., 2026; Chen et al., 2026; Deng et al., 2026; Orlanski et al., 2026; Wang et al., 2025). A parallel line of work applies classical static measurements to LLM-generated code (Wang et al., 2025; Molison et al., 2025) as a shorthand for how maintainable it will be, including the widely used class-level state cohesion measures (Li & Henry, 1993; Chidamber & Kemerer, 1994; Briand et al., 1998). To measure the difficulty carried into the next piece of work, we hold the artifact fixed while the downstream work varies, and we hold the downstream worker fixed while the artifact varies. Maintainability, as used here, is the difficulty experienced by a fixed downstream worker carrying out a specified next task on a frozen artifact.

Two limits in current designs. The first limit is resolution across maintenance demands. Existing benchmarks evaluate maintainability through targeted repository edits or broader processes of code evolution, but the resulting measurements do not directly compare different maintenance demands on the same frozen artifact. A maintenance task can place several demands on the implementation at once, making it difficult to observe whether the same code is easy under one maintenance demand but difficult under another.

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.

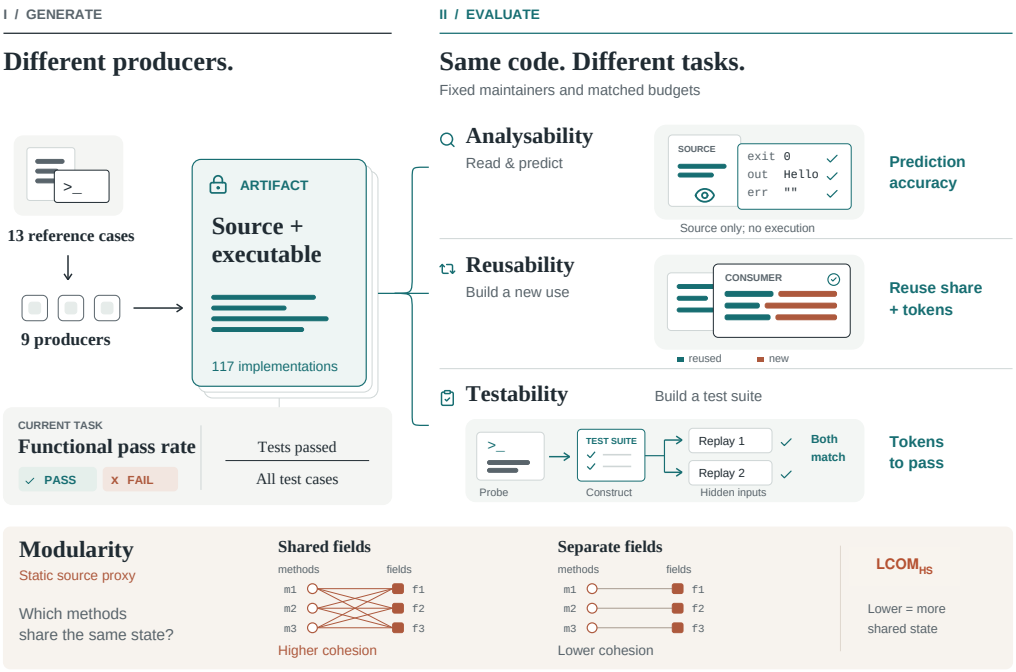

Figure 1: Overview of CARRYBENCH: Stage II evaluates frozen outputs through targeted tasks centered on Analysability, Reusability, and Testability, while a static Modularity proxy measures the same Stage I source.

The second limit is attribution. NITR (Zhu et al., 2026) evaluates agent-produced edits, while SWE-CI (Chen et al., 2026), SWE-Milestone (Deng et al., 2026), and SlopCodeBench (Orlanski et al., 2026) evaluate continued software evolution. MaintainBench (Wang et al., 2025) explicitly separates the initial artifact from the downstream worker: different methods produce the initial code, and a fixed downstream generator adapts it to changed requirements, with the comparison repeated under a second generator. Its downstream measurement is centered on requirement adaptation as a whole. CARRYBENCH builds on this separation by applying multiple targeted tasks to corresponding frozen repositories, making different maintenance demands separately observable under three fixed maintainer profiles.

What CARRYBENCH measures. The name refers to the difficulty that generated code carries forward: work not completed by the producer is transferred to a subsequent maintainer, and CARRYBENCH measures it by running the next work itself. The two-stage protocol runs over 13 repository-level reimplementation cases from ProgramBench (Yang et al., 2026): Stage I obtains executable artifacts from nine producer models, and Stage II freezes each artifact and evaluates it with three fixed downstream maintainer profiles under matched protocols and budgets.

Guided by ISO/IEC (2023), we evaluate three representative maintenance demands: Analysability, Reusability, and Testability. For each demand, we define a targeted downstream task and a corresponding role to perform it (Figure 1). For Analysability, an observer predicts program behavior from source alone. For Reusability, a consumer builder constructs a new use around the retained source. For Testability, a tester constructs an executable black-box test suite. The same protocol can support additional maintenance demands through corresponding downstream tasks and roles. We also report static Modularity on the same source as a traditional maintainability proxy built on state cohesion, a classical structural measurement related to code maintainability (Li & Henry, 1993; Chidamber & Kemerer, 1994; Briand et al., 1998). The same artifacts and maintainer profiles are used for the questions below. Each comparison is reported for that question alone. Analysability, Reusability, and Testability remain separate outcomes.

We ask four questions:

1. With the maintainer fixed, do the outcomes of Analysability, Reusability, and Testability induce consistent producer rankings?
2. For each maintenance demand, do producer rankings transfer across maintainer profiles?
3. Does static state cohesion agree with the downstream measurements on the same artifacts?
4. Which source structures are associated with each downstream measurement, and how do these associations vary across maintainer profiles?

The resulting rankings motivate reporting results separately for each maintenance demand and maintainer profile (Section 4.2).

2 Related Work

Maintainability beyond correctness. Recent benchmarks treat downstream maintainability as a distinct evaluation target (Zhu et al., 2026; Chen et al., 2026; Deng et al., 2026; Orlanski et al., 2026; Wang et al., 2025). NITR (Zhu et al., 2026) adds structural oracles to repository-editing probes across nine maintainability dimensions and identifies failures that still pass functional tests. Two of its dimensions are partially related to Reusability and Testability, and another is partially related to the static Modularity proxy. SWE-CI (Chen et al., 2026) and SWE-Milestone (Deng et al., 2026) evaluate continuous repository changes, SlopCodeBench (Orlanski et al., 2026) tracks structural degradation during repeated extension, and MaintainBench (Wang et al., 2025) evaluates adaptation to changed requirements.

They differ in what the resulting score is about. NITR, SWE-CI, and SlopCodeBench combine properties of the evolving artifact with the agent performing the later work. MaintainBench separates the initial artifact from a fixed downstream generator. The present study builds on that artifact–maintainer separation by evaluating three maintenance demands through separate downstream tasks on corresponding frozen repositories, each repeated under three fixed maintainer profiles.

Task-specific neighbors. Other benchmarks study activities related to individual maintenance demands. FEA-Bench (Li et al., 2025), LibEvolutionEval (Kuhar et al., 2025), and HumanEvo (Zheng et al., 2025) evaluate repository-level feature implementation, version-specific library use, and evolution-aware generation. Their endpoints are generation outcomes, whereas Reusability here measures how a fixed maintainer builds a new use from one frozen artifact. CCISolver (Zhong et al., 2025) and DocChecker (Dau et al., 2024) detect and repair code–comment inconsistency. That covers one documentation channel relevant to Analysability, and behavior recovery as a whole is a separate measurement. SmellBench (Dinu et al., 2026) evaluates detection and repair of cross-module architectural smells. It is the closest structural neighbor to the static Modularity proxy. MiniCode/Librarian (Kovacic et al., 2025) studies library design from multiple independent programs, TEBench (Shang et al., 2026) studies test-suite evolution after a code change, and SWE-Refactor (Xu et al., 2026) evaluates behavior-preserving repository refactoring. These tasks exercise reuse, testing, or structural change, but none freezes producer artifacts and applies the same downstream maintainer across producers. Appendix Table A1 compares the design space along the attribution, separation, and profile axes of comparison.

3 The CarryBench Evaluation Protocol

CarryBench is built in two stages: Stage I produces the artifacts, and Stage II freezes them and measures how difficult they are for a downstream worker who is held fixed.

3.1 Stage I: Producing the Artifacts

Stage I creates the code that the downstream tasks later compare. The 13 repository cases come from ProgramBench (Yang et al., 2026), which supplies pinned open-source repositories, compiled reference executables, documentation, and reference-derived behavioral tests.

For each case x and producer model m, the producer receives the compiled reference program and its documentation. It may probe that program through its public interface but cannot read the original source or access the Internet. It writes a new codebase intended to reproduce the reference behavior. The resulting executable is evaluated against the behavioral tests attached to the case, and its Stage I pass rate records the fraction of tests it passes. The resulting repository, executable, and evaluation record form one frozen Stage I artifact $a_{x,m}$, the repository-level unit passed to Stage II.

3.2 Stage II: Fixing the Maintainer

Stage I artifacts used by a downstream task are frozen and handed to Stage II: a downstream worker receives the generated source and only the public input that its own activity requires, while the reference source, the hidden expected outputs, and the scoring rules stay out of reach. Within an evaluation panel the maintainer profile, including its model and deterministic decoding configuration, tool interface, interaction budget, and evaluator are all held fixed, so artifacts from different producer models go through the same procedure.

Stage II evaluates three maintenance demands through corresponding downstream tasks: an observer recovers behavior from source for Analysability, a consumer builder builds a new consumer for Reusability, and a tester constructs an executable black-box test suite for Testability. For downstream task t, a fixed maintainer d and budget b produce the outcome

$$y_{x,m,t,d} = F_t(a_{x,m}, d, b), \tag{1}$$

where F_t is the task-specific evaluator. A static Modularity proxy is computed directly from the frozen Stage I source. It records how responsibilities are divided among the stateful units.

Two of the three downstream task measurements carry an explicit control against shortcut solutions: Reusability adds a no-op substitution test against quiet reimplementation, and Testability requires agreement across two independent source-blind replays. The controls are described with the downstream tasks that use them.

3.3 Analysability

Analysability is evaluated as bounded behavioral recovery: a fixed observer is given a generated repository and command-line questions, and must predict the exit code, standard output, and standard error *without running the program*. Reading the source is the only evidence available. The expected outputs and test assertions are hidden, and repository inspection is bounded by tool-call and step budgets in the protocol.

The questions are derived from generation tests that every compared artifact passes, so the comparison holds functional correctness constant. Questions mix difficulty levels, behavior types, output formats, and argument patterns. Predictions are graded by the original behavioral oracle, and the Analysability score is the fraction of questions whose exit code, standard output, and standard error all match. Question counts, budgets, and grading rules are given in Appendix A2.

3.4 Reusability

Reusability asks a fixed downstream maintainer to build a new consumer on top of the generated implementation. The consumer performs a case-specific workflow, such as aggregating program results or applying a policy. The maintainer receives the frozen repository and a public consumer requirement, and has up to three submissions. To be accepted, a consumer must invoke the generated executable and must pass hidden valid workloads together with malformed-input and schema-invalid controls in each case.

The no-op substitution control. Passing those workloads does not yet prove that the consumer uses the generated program. A consumer that quietly reimplements the behavior would pass them too. The control replaces the generated executable with a successful no-op and runs the valid workloads again. An accepted consumer must now fail. A consumer that still passes never depended on the artifact. It is rejected and receives no reuse credit.

Scoring how much source was reused. Executed task-relevant source lines from both the consumer and the invoked artifact are split into three provenance classes: directly reused lines D, adapted lines A covered by an explicit transformation map to the generated implementation, and new lines N. Direct reuse includes generated-source lines reached through normal artifact invocation as well as unchanged source reused in the consumer. With $T = D + A + N$, the primary reuse metric is

$$\text{SOURCE REUSE SHARE} = \frac{D + A}{T}, \tag{2}$$

which measures the provenance of executed task-relevant code. Construction tokens measure the cumulative interaction cost through the first accepted submission. Construction tokens are summed from the model interaction record across all attempts through the first accepted submission and reported alongside SOURCE REUSE SHARE. Line normalization, startup exclusion, token matching, transformation-map bookkeeping, and outlier review are specified in Appendix A2.

3.5 TESTABILITY

Testability asks a fixed tester to construct an executable black-box test suite for a generated program. The tester receives the frozen source, a set of prescribed behavioral scenarios, and an observation schema covering exit codes, outputs, files, and relations between runs. It probes the executable through a controlled execution interface, writes a test suite, and submits that suite for evaluation. The interface exposes the same process controls and observation fields regardless of the implementation language, and authoring runs under matched budgets with auxiliary capabilities disabled, so testers face a comparable task across Python, Go, Rust, and C++.

The two-replay acceptance control. A suite that hardcodes whatever output it happened to see would pass a single rerun, so acceptance requires more than one run. The submitted suite is executed twice in independent source-blind environments with private inputs. Both runs must be stable and must match trusted observations produced from the same artifact, after a task-defined canonicalization that preserves field types and list order. Acceptance requires exact field-level agreement in both hidden replays. Construction tokens are accumulated across all attempts through the first submission that reaches this threshold. Probe, harness-run, and tool-interaction budgets, disabled capabilities, and canonicalization rules are given in Appendix A2.

3.6 MODULARITY

Modularity requires no downstream worker: it is measured directly from the frozen Stage I source by asking how consistently the instance methods of each stateful unit touch the same fields. Production source is parsed for Python, Go, Rust, and C++, and each class or struct becomes a method–field matrix. Methods that share fields give a lower LCOM_{HS}, and methods that use disjoint fields give a higher one.

For an eligible unit with m methods and a active fields, let $\mu(A_j)$ be the number of methods that access field A_j. Henderson–Sellers normalized LCOM (Henderson-Sellers, 1996) is computed as

$$\text{LCOM}_{\text{HS}} = \frac{m - \frac{1}{a}\sum_{j=1}^{a} \mu(A_j)}{m - 1}. \tag{3}$$

The artifact value is the unweighted mean across eligible units. Per-language extraction rules, unit eligibility, and retained diagnostics are given in Appendix A2.

3.7 TASK INSTANCES

Analysability and Reusability each use one task instance for each of the 13 cases, while Testability comprises 12 prescribed task instances over 7 independent cases, with multiple task outcomes for the same artifact averaged before artifact-level analysis. Each maintainer profile is used for all three maintenance demands and excludes its corresponding producer.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

The comparison includes nine Stage I producer models: DeepSeek V4 Flash (DeepSeek-AI, 2026a), DeepSeek V4 Pro (DeepSeek-AI, 2026b), GLM 5.1 (Zhipu AI, 2026a), GLM 5.2 (Zhipu AI, 2026b), HY3 (Tencent Hy Team, 2026), Kimi K2.7 Code (Kimi AI, 2026), MiniMax M3 (Lai et al., 2026), GPT-5.6 Sol (OpenAI, 2026), and Claude Opus 5 (Anthropic, 2026). Downstream evaluation uses three fixed maintainer profiles with deterministic decoding (DeepSeek V4 Flash, HY3, and Kimi K2.7 Code), under the same task protocols and budgets. Each profile fixes the downstream worker and its interaction configuration, so artifacts from different producer models are processed under the same conditions. Excluding the corresponding producer leaves each profile with $13 \times 8 = 104$ Analysability and Reusability evaluations and $12 \times 8 = 96$ Testability task runs. Modularity covers all 117 Stage I artifacts, of which 66 contain eligible stateful units. Outcomes are first summarized per artifact and then averaged equally across artifacts for each producer and maintainer profile. Construction tokens are normalized by the median of their own maintainer profile, and producer–maintainer identity pairs are excluded, so the comparison is between artifacts.

4.2 RANKING AGREEMENT

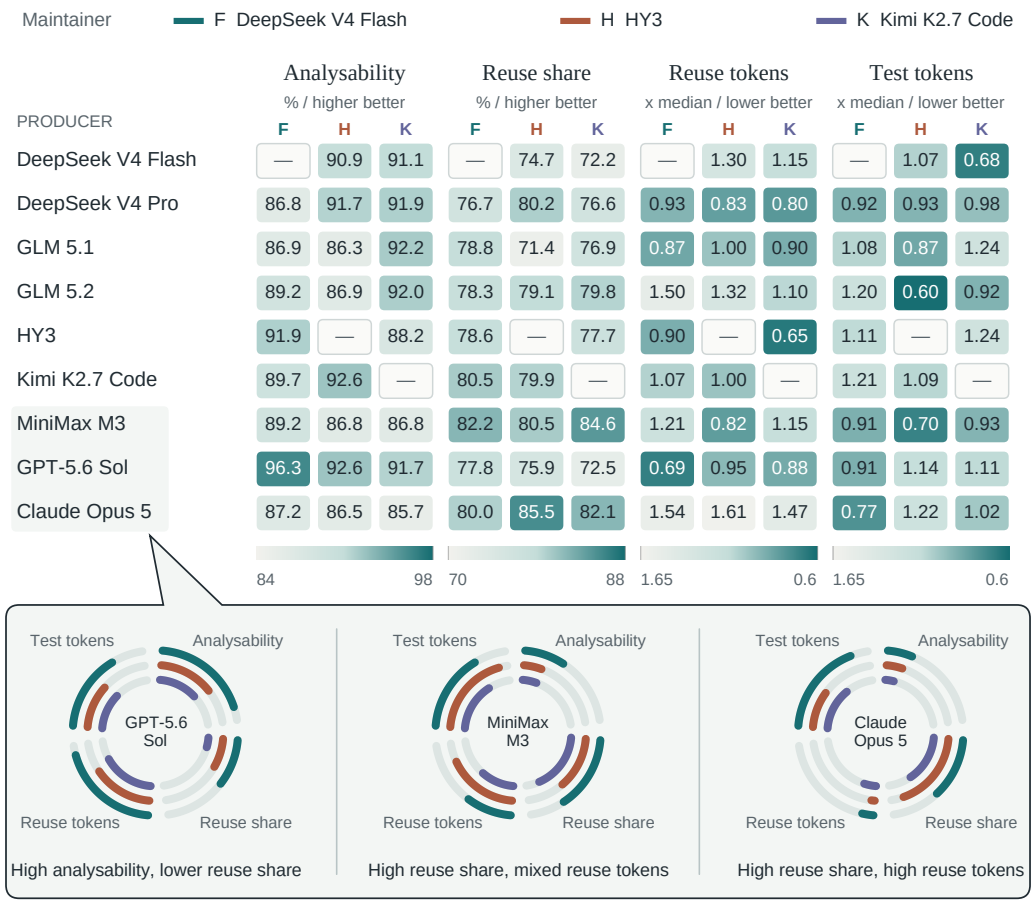

Figure 2: Producer-level downstream outcomes. Each cell gives one evaluated producer–maintainer pair. Dashes exclude self-maintenance. Tokens are multiples of the maintainer profile's median. Darker cells indicate better values on the displayed outcome-specific scales. The three portraits repeat selected rows using those scales. A more filled arc is better on that outcome's scale. Appendix Table A2 also reports static Modularity and Stage I pass rate.

Each maintainer profile induces a producer ranking for every downstream outcome. Analysability score and source reuse share are ranked from high to low. Construction tokens are ranked from low to high. A cross-maintainer comparison retains the seven producer models evaluated by both profiles after their respective self-maintenance results are excluded, and asks whether the producer ranking for a downstream outcome remains similar when the fixed maintainer profile changes. Kendall's τ_a compares the order of every producer pair in the two rankings, counting concordant and reversed pairs. Exact permutation p values hold one ranking fixed and enumerate all 7! orderings of the other. Within a maintainer profile, cross-demand outcome comparisons use the eight producer models evaluated for both outcomes.

Figure 2 shows producer-level outcomes under the three maintainer profiles. Appendix Table A2 retains exact values, the static Modularity proxy, and the Stage I pass rate. All Reusability and Testability runs eventually reached acceptance, with construction tokens accumulated through the first accepted submission, including earlier attempts.

Finding 1: Downstream outcomes yield different producer rankings. Within a fixed maintainer profile, mean τ_a across the five cross-demand outcome pairs is $+0.04$ for DeepSeek V4 Flash, -0.02 for HY3, and -0.07 for Kimi K2.7 Code. Each coefficient is computed from producer-level outcomes aggregated over the artifacts shared by both outcomes under the same maintainer profile. The orderings in Figure 2 differ, so the outcomes are reported separately.

Finding 2: Reusability rankings show the strongest cross-maintainer agreement. Across maintainer pairs, producer rankings agree most consistently on Reusability (Figure 2): source reuse share gives $\tau_a = +0.24$ for DeepSeek V4 Flash–HY3, $+0.62$ for DeepSeek V4 Flash–Kimi K2.7 Code, and $+0.52$ for HY3–Kimi K2.7 Code (mean $+0.46$). The latter two comparisons have one-sided exact permutation p values of 0.034 and 0.068, respectively. Construction tokens have mean $\tau_a = +0.49$, with all pairwise coefficients positive. Agreement is weaker for Analysability ($+0.16$) and Testability (-0.02). Complete pairwise coefficients and the restricted six-producer rank view are in Appendix A3.1.

4.3 State Cohesion

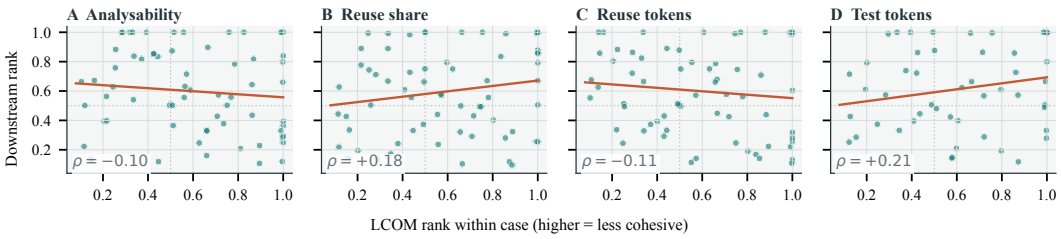

Figure 3: Within-case rank comparison between state cohesion and the four downstream outcomes: the within-case rank of LCOM_{HS} (higher = less cohesive) versus the within-case downstream rank, with rank correlation ρ shown per panel: (A) Analysability score, -0.10, (B) Reusability source reuse share, $+0.18$, (C) Reusability construction tokens, -0.11, and (D) Testability construction tokens, $+0.21$. Associations are weak and mixed in sign: state cohesion is not a directional proxy for downstream outcomes.

Modularity has no downstream worker to ablate against. Pairs of artifacts produced for the same case are examined to determine whether their LCOM_{HS} ordering is reversed by the reported Analysability score, source reuse share, Reusability construction tokens, or Testability construction tokens.

Finding 3: Higher state cohesion does not guarantee easier downstream work. Across cases, the within-case associations between state cohesion and the four downstream outcomes are weak and mixed in direction (Figure 3). In the `rhit` comparison, Claude Opus 5 has LCOM_{HS}=0.500 and GLM 5.2 has 0.700, so the static proxy favors Claude Opus 5. Yet GLM 5.2 attains a higher Analysability score and source reuse share while spending fewer Reusability and Testability construction tokens.

State cohesion alone does not substitute for the downstream evaluation in this case. Additional cohesion analysis is in Appendix A3.3.

4.4 Source-Feature Association

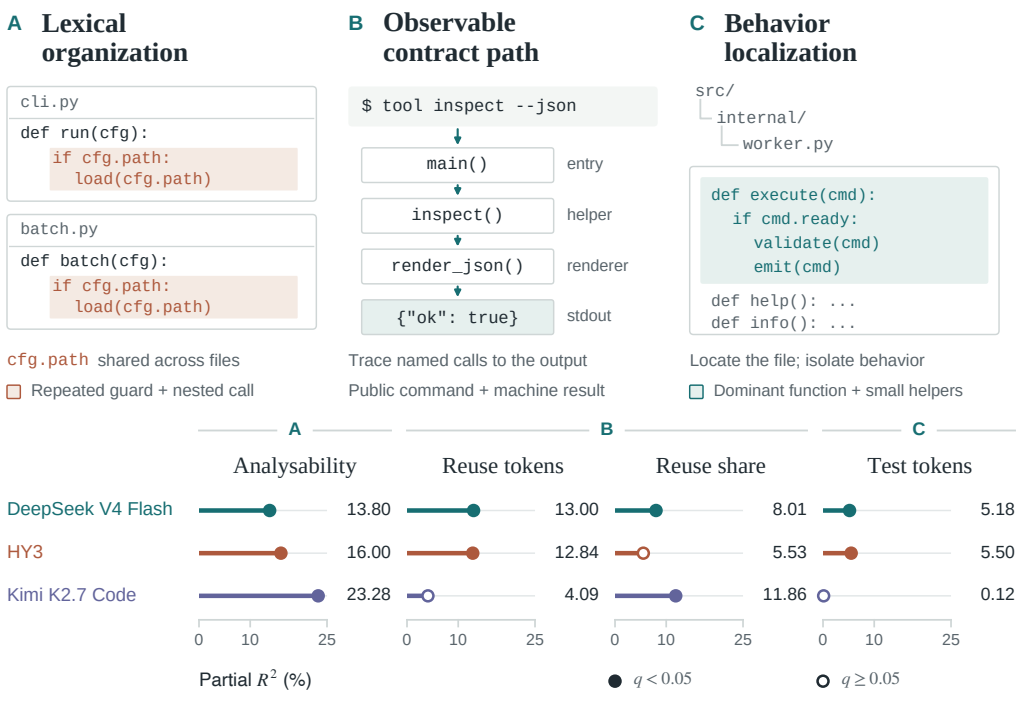

Figure 4: Source-feature associations across maintainer profiles. Illustrative source excerpts (not actual repository code) show cross-file naming and duplicated control structure (A), a public command reaching a machine-readable result (B), and behavior concentrated in a function within a nested source file (C). Matching letters link these examples to the outcome panels. Each maintainer has one shared row. Lengths, endpoint positions, and numbers give partial R^2 on the same 0–25% scale, conditional on the prespecified controls. Filled endpoints indicate $q < 0.05$ and hollow endpoints $q \geq 0.05$ (within-outcome Benjamini–Hochberg). These associations are not causal. Exact values are in Appendix Table A4.

Each maintainer profile is analyzed separately at the artifact level. The baseline regression includes case, producer model, language, and the applicable outcome-specific size or correctness covariate. The full model adds one prespecified feature block, a set of static metrics that represent the same source characteristic. For artifact a, outcome o, maintainer d, controls $B_{a,o}$, and feature block C_a, the full model is

$$Y_{a,o,d} \sim B_{a,o} + C_a. \tag{4}$$

Partial R^2 measures the fraction of outcome variation remaining after the baseline controls that is additionally explained by the feature block:

$$\text{Partial } R^2_C = \frac{R^2(B + C) - R^2(B)}{1 - R^2(B)}. \tag{5}$$

The three profile panels exclude their producer–maintainer identity pairs. Before fitting, Analysability aggregates its 50 questions into one score per artifact, and Testability averages multiple task outcomes for the same artifact. Reusability is already artifact-level. Significance testing permutes baseline residuals within cases. Benjamini–Hochberg correction is applied to the block tests for each outcome (Benjamini & Hochberg, 1995), and leave-one-case-out ranges show sensitivity to individual cases. Outcome-specific control lists are in Appendix A4.1. Block compositions are in Appendix A4.

Finding 4: Source feature associations differ across maintenance demands and maintainer profiles alike. After the prespecified controls, lexical organization is associated with Analysability under all three maintainers (partial $R^2 = 13.80\%$–23.28%, all $q < 0.05$). The observable-contract-path block reaches the threshold for Reusability tokens under DeepSeek V4 Flash and HY3, and for source reuse share under DeepSeek V4 Flash and Kimi K2.7 Code. Behavior localization reaches the threshold for Testability tokens under DeepSeek V4 Flash and HY3, with a near-zero fitted association under Kimi K2.7 Code. Figure 4 shows the magnitudes and threshold status. Appendix Table A4 retains all exact values.

5 DISCUSSION

How to read a CARRYBENCH result. Each CARRYBENCH score is operational. It is the difficulty a fixed downstream worker experiences on a specified next task over a frozen artifact, and it is not an intrinsic property of the code. Choose the panel for the anticipated work: Analysability for reading, Reusability for building around the artifact, and Testability for constructing tests. Then compare producers within a maintainer profile, and check agreement across profiles (Findings 1 and 2).

When the outcomes disagree. A producer that ranks high on one demand can rank lower on another, and that disagreement is reported rather than averaged away (Finding 1). Reusability is the demand on which the producer order agrees most across the three profiles (Finding 2), whereas the producer order for Analysability and Testability depends more strongly on the maintainer profile. State cohesion can be listed beside these outcomes. In the matched comparison it does not stand in for easier downstream work (Finding 3).

Maintainer coverage. With more than one profile, agreement can be measured. Two profiles give one pairwise comparison. The third shows further ranking and source-association patterns (Findings 1 and 4). Three profiles triple the Stage II evaluation count relative to one. How a fixed budget should be split between profiles and cases is untested. Reporting both the coverage and the profile-conditioned outcomes lets a reader match the design to the downstream work.

Limitations. CARRYBENCH gives each maintenance demand one downstream task protocol and one corresponding role. Comparisons across demands use corresponding frozen outputs and hold that protocol fixed. Further task protocols within a demand are not included, so within-demand variation is not measured. The downstream maintainers are LLM-based coding agents, and the reported patterns describe those automated maintainers. The same source-level associations have not been measured for human maintainers. Modularity covers all 117 Stage I artifacts, of which 66 yield measurements from eligible stateful units. The source-feature analyses are observational (Finding 4). The prespecified blocks connect outcomes to source properties that can be inspected. The conditional associations are candidates for reranking studies, for reward signals based on task success, construction tokens, or retained executable source, and for paired behavior-preserving refactorings. They do not show gains from reranking or training, and they are not a causal account of how to change the source. These limits qualify the reported associations. They leave the evaluation protocol as specified.

6 CONCLUSION

CARRYBENCH measures maintainability as the difficulty a fixed downstream worker experiences on a specified next task over a frozen artifact. Nine producer models generate repository-scale programs for 13 cases in four languages. Analysability, Reusability, and Testability each have a targeted downstream task, run under three fixed maintainer profiles. Static Modularity is reported separately. The executable evaluations use a no-op substitution control and a two-replay control. Producer rankings agree most across maintainers for Reusability, and they differ across downstream outcomes. A matched case shows that state cohesion does not stand in for easier downstream work. Prespecified source-feature ablations find conditional associations that vary by maintenance demand and by profile. Each maintenance demand is reported under each maintainer profile.

AI USE STATEMENT

Generative AI tools were used to assist with benchmark workflow development and inspection, the design of individual downstream task instances, manuscript drafting and editing, literature organization, and interpretation of recorded experimental results. The authors reviewed all AI-assisted work and checked the reported methods, values, and claims against the experiment artifacts. The authors take responsibility for the final content of this work, including text and analyses produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 specify the evaluation protocol, outcome definitions, and analysis procedures. Appendix A2 provides the detailed question sampling, task budgets, grading rules, and extraction mechanics, Appendix A4 gives the full statistical specification and the composition of all prespecified ablation blocks, Appendix A5 introduces the selected repository cases, and Appendix A6 provides model-level block results. Appendix A3 retains the complete producer-level outcome and source-feature summary tables, rank comparisons, and additional cohesion analysis. The benchmark protocol implementation is available at `https://anonymous.4open.science/r/CarryBench`.

REFERENCES

Anthropic. Claude Opus 5 System Card. System card, 2026. URL https://www.anthropic.com/claude-opus-5-system-card.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

Lionel C. Briand, John W. Daly, and Jürgen Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, 3(1):65–117, 1998. doi: 10.1023/A:1009783721306.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. SWE-CI: Evaluating agent capabilities in maintaining codebases via continuous integration. *arXiv preprint arXiv:2603.03823*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

Anh Dau, Jin LC Guo, and Nghi Bui. DocChecker: Bootstrapping code large language model for detecting and resolving code-comment inconsistencies. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 187–194, 2024.

DeepSeek-AI. DeepSeek-V4-Flash. Hugging Face model card, 2026a. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash. Accessed: 2026-09-06.

DeepSeek-AI. DeepSeek-V4-Pro. Hugging Face model card, 2026b. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro. Accessed: 2026-09-06.

Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, Qian Zhang, Viktor Prasanna, Xiangru Tang, and Xingyao Wang. SWE-Milestone: Evaluating AI agents on continuous software evolution. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2603.13428.

Ion George Dinu, Marian Cristian Mihăescu, and Traian Rebedea. SmellBench: Evaluating LLM agents on architectural code smell repair. *arXiv preprint arXiv:2605.07001*, 2026.

Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1996.

ISO/IEC. ISO/IEC 25010:2023: Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – product quality model. International Standard, 2023. URL https://www.iso.org/standard/78176.html.

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*, 2024.

Kimi AI. Kimi K2.7 Code. Hugging Face model card, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.7-Code. Accessed September 7, 2026.

Ziga Kovacic, Justin T Chiu, Celine Lee, Wenting Zhao, and Kevin Ellis. Refactoring codebases through library design. *arXiv preprint arXiv:2506.11058*, 2025.

Sachit Kuhar, Wasi Ahmad, Zijian Wang, Nihal Jain, Haifeng Qian, Baishakhi Ray, Murali Krishna Ramanathan, Xiaofei Ma, and Anoop Deoras. LibEvolutionEval: A benchmark and study for version-specific code generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6826–6840, 2025.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, Jinkai Hu, Jiayao Li, Rui Gao, Zekun Li, Songquan Zhu, Jingkai Zhou, and Pengyu Zhao. MiniMax Sparse Attention, 2026. URL https://arxiv.org/abs/2606.13392.

Wei Li and Sallie M. Henry. Maintenance metrics for the object-oriented paradigm. In *Proceedings of the First International Software Metrics Symposium*, pp. 52–60. IEEE, 1993. doi: 10.1109/METRIC.1993.263801.

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.

Alfred Santa Molison, Marcia Moraes, Glaucia Melo, Fabio Santos, and Wesley K. G. Assuncao. Is LLM-generated code more maintainable & reliable than human-written code? In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 151–162. IEEE, 2025. doi: 10.1109/ESEM64174.2025.00036.

OpenAI. GPT-5.6 System Card. System card, 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

Gabriel Orlanski, Devjeet Roy, Alexander Yun, Changho Shin, Alex Gu, Albert Ge, Dyah Adila, Nicholas Roberts, Frederic Sala, and Aws Albarghouthi. SlopCodeBench: Benchmarking how coding agents degrade over long-horizon iterative tasks. *arXiv preprint arXiv:2603.24755*, 2026.

Ye Shang, Quanjun Zhang, Haichuan Hu, Chunrong Fang, Liang Xiao, and Zhenyu Chen. Breaking, stale, or missing? benchmarking coding agents on project-level test evolution. *arXiv preprint arXiv:2605.06125*, 2026.

Tencent Hy Team. Hy3 model card. Hugging Face, 2026. URL https://huggingface.co/tencent/Hy3. Accessed September 7, 2026.

Zhengren Wang, Rui Ling, Chufan Wang, Yongan Yu, Sizhe Wang, Zhiyu Li, Feiyu Xiong, and Wentao Zhang. MaintainCoder: Maintainable code generation under dynamic requirements. *Advances in Neural Information Processing Systems*, 38:17378–17415, 2025. doi: 10.52202/085713-0521.

Yisen Xu, Jinqiu Yang, and Tse-Hsun Chen. SWE-Refactor: A repository-level benchmark for real-world LLM-based code refactoring. *arXiv preprint arXiv:2602.03712*, 2026.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, et al. ProgramBench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026.

Dewu Zheng, Yanlin Wang, Ensheng Shi, Ruikai Zhang, Yuchi Ma, Hongyu Zhang, and Zibin Zheng. HumanEvo: An evolution-aware benchmark for more realistic evaluation of repository-level code generation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1372–1384. IEEE, 2025.

Zhipu AI. GLM-5.1. Zhipu AI model documentation, 2026a. URL https://docs.bigmodel.cn/cn/guide/models/text/glm-5.1. Accessed: 2026-09-06.

Zhipu AI. GLM-5.2. Zhipu AI model documentation, 2026b. URL https://docs.bigmodel.cn/cn/guide/models/text/glm-5.2. Accessed: 2026-09-06.

Renyi Zhong, Yintong Huo, Wenwei Gu, Jinxi Kuang, Zhihan Jiang, Guangba Yu, Yichen Li, David Lo, and Michael R Lyu. CCISolver: End-to-end detection and repair of method-level code-comment inconsistency. *arXiv preprint arXiv:2506.20558*, 2025.

Haichao Zhu, Qian Zhang, Jiyuan Wang, Zhaorui Yang, and Yuxin Qiu. Needle in the repo: A benchmark for maintainability in AI-generated repository edits. *arXiv preprint arXiv:2603.27745*, 2026.

APPENDIX INDEX

Appendix and section	Contents	Page
A1 Benchmark Comparison		13
Comparison of representative benchmarks	Design-space axes and coverage.	14
A2 Protocol Implementation Details		14
Analysability budget and grading	Question sampling, inspection and step budgets, grading rules.	14
Reusability instrumentation	Line normalization, startup exclusion, and provenance classes.	14
Testability budgets and replays	Probe, harness-run, tool budgets, and replay rules.	15
Modularity extraction	Per-language state-ownership rules and unit eligibility.	15
A3 Supplementary Summary Results		16
Producer-level outcomes	Complete values underlying the outcome panels.	16
Rank comparisons	Six-producer rank view and pairwise comparison details.	17
Source-feature summary	Exact partial R^2 and adjusted q values.	17
Additional cohesion analysis	Methods per unit and residual state-cohesion variation.	17
A4 Ablation Block Catalogue		18
Statistical model and testing	Regression specification, Partial R^2 definition, permutation testing.	18
Analysability blocks	Semantic definitions and components of the 10 observer-oriented feature blocks.	18
Reusability blocks	Semantic definitions and components of the 4 consumer-oriented feature blocks.	19
Testability blocks	Semantic definitions and components of the 15 tester-oriented feature blocks.	20
A5 Repository Cases		20
A6 Model-Level Block Results		21
Analysability block	Lexical-organization component means and association results.	22
Reusability block	Observable-contract-path means and association results.	22
Testability block	Behavior-localization component means and association results.	23
Remaining candidate blocks	Component means and association results for the remaining prespecified blocks.	23
A7 Concrete Downstream Task Examples		28
A7.1 Analysability example	Melody pattern compilation and validation.	28
A7.2 Reusability example	Rhit incident-triage consumer.	29
A7.3 Testability example	Tokei streaming and filtering tester.	29

A1 BENCHMARK COMPARISON

Table A1 compares representative benchmarks on three axes: what the score is attributed to, whether the maintainer is held fixed across the artifacts, and how many such fixed maintainers the comparison is reported under. MaintainBench attributes its score to the artifact, separates the artifact from the maintainer, and reports two fixed maintainers on one downstream activity. The other benchmarks in the table rank the coding agent. CARRYBENCH attributes the score to the artifact, separates the maintainer, reports under three fixed maintainer profiles, and evaluates the three maintenance demands through separate downstream tasks on corresponding frozen outputs. The comparison shows how difficult each artifact is under each maintenance demand and maintainer profile, so both forms of variation remain observable.

Table A1: Comparison of representative benchmarks. **Ranks** records what the benchmark's score is attributed to: the coding agent that performed the task, or the artifact under test. **Sep.** is artifact–maintainer separation, that is, whether the downstream work is performed by a worker held fixed across the artifacts being compared. **Profiles** counts the distinct fixed downstream maintainers under which the comparison is reported, and is not applicable where there is no separated maintainer. For the three maintenance demands and the static Modularity proxy, ✓ denotes direct coverage, ∘ a related but different task, and × no corresponding endpoint. Analysability, Reusability, Testability, and Modularity are abbreviated as Analys., Reus., Test., and Modul.

Benchmark	Ranks	Sep.	Profiles	Analys.	Reus.	Test.	Modul.
Single-activity benchmarks							
Zhu et al. (2026)	Agent	×	—	×	∘	∘	∘
Chen et al. (2026)	Agent	×	—	×	×	×	×
Deng et al. (2026)	Agent	×	—	×	×	×	×
Orlanski et al. (2026)	Agent	×	—	×	×	×	∘
Wang et al. (2025)	Artifact	✓	2	×	×	×	×
Dinu et al. (2026)	Agent	×	—	×	×	×	✓
Kovacic et al. (2025)	Agent	×	—	×	∘	×	×
Shang et al. (2026)	Agent	×	—	×	×	∘	×
Multi-activity benchmark							
CARRYBENCH (Ours)	Artifact	✓	**3**	✓	✓	✓	✓

A2 PROTOCOL IMPLEMENTATION DETAILS

This appendix collects the operational details removed from the main text: question sampling, budgets, grading rules, and extraction mechanics for the three downstream tasks and the static Modularity proxy. All settings are held fixed across the artifacts being compared.

A2.1 ANALYSABILITY BUDGET AND GRADING

Each case contributes 50 questions sampled toward a mix of 15% easy, 70% medium, and 15% hard. Difficulty reflects the behavior being exercised, the output structure, the use of fixtures and files, the output length, and the number of invocations. Sampling is balanced across behavior types, output formats, generation-test branches, and argument patterns. The observer may list directories, read files, and search production source and fixtures. It has five such inspection calls shared across the whole exam and an overall limit of 200 agent steps. Predictions are graded by the original behavioral oracle: exit codes must match exactly, and each stream is compared under the oracle-defined exact, JSON/NDJSON, or trailing-whitespace rule.

A2.2 REUSABILITY INSTRUMENTATION AND PROVENANCE

The accepted consumer is run once more under process-tree instrumentation. Executed source lines are normalized across languages, excluding comments, structural lines, evaluator code, and third-party code. Lines reached by a malformed-input startup control form E_0, and lines reached by the valid workloads form E_{task}. Only task-relevant lines actually reached during valid workloads enter the reuse calculation, including generated-source lines reached through normal artifact calls. The classification set is $E_2 = E_{\mathrm{task}} \setminus E_0$.

D counts executed generated lines that remain unchanged at their original path, are exact whole-file copies, or match at least 24 consecutive normalized tokens. A counts executed generated lines copied into the consumer and then modified, as recorded by the transformation map. N counts every other executed task-relevant line, including newly written consumer code and similar-looking code without a mapped source range.

Because unchanged generated-source lines reached through ordinary artifact calls contribute to D, a long execution path could raise the measured share without additional source being incorporated

into the consumer. Source reuse shares were screened for outliers using the interquartile range (IQR), and high- and low-share outliers were reviewed by human experts. The review found no high-share outlier driven by a long execution chain through unchanged artifact code.

A2.3 TESTABILITY BUDGETS AND REPLAY RULES

Before the first submission check, authoring is bounded by 64 tool interactions, 64 controlled process starts shared by direct probes and harness execution, and 12 local harness runs. The first rejected check unlocks an additional reserve of 64 tool interactions, 64 process starts, and 12 harness runs in the same session, giving cumulative ceilings of 128 tool interactions, 128 process starts, and 24 harness runs. At most 12 submission checks are permitted. Operating-system shell access, networking, container control, compilers, package managers, and language-specific test frameworks are all disabled. Each submission check runs two hidden replays in independent source-blind environments with private inputs, and observations are compared after a task-defined canonicalization that preserves field types and list order.

A2.4 MODULARITY EXTRACTION

Extraction follows each language's own notion of state ownership. Python classes contribute instance methods and accesses to receiver-owned fields. Go structs contribute receiver methods and field selectors. Rust structs contribute methods defined in implementation blocks. C++ classes contribute inline methods along with methods defined outside the class declaration. Static members are excluded throughout, and a unit is eligible once it has at least two instance methods and one accessed instance field. Matrix columns contain fields accessed by at least 1 method. Unit-level values, parser status, source size, and measurement coverage are retained with the artifact-level aggregate.

A3 SUPPLEMENTARY SUMMARY RESULTS

A3.1 PRODUCER-LEVEL OUTCOMES AND RANK COMPARISONS

Table A2 retains the complete producer-level outcomes underlying Figure 2, including the upstream correctness reference and static Modularity proxy. Figure A1 shows the restricted set of six producers observed under all three maintainer profiles. Its coefficients use a different producer intersection from the pairwise comparisons in Finding 2.

Table A2: Producer-level outcomes under each maintainer. A dash marks the producer that is that row's maintainer, and that cell is not evaluated. Construction tokens are multiples of the row median (Reusability 859k, 659k, 1,343k, and Testability 292k, 243k, 160k). Bold and underline mark the best and second best. The mean skips missing cells. † marks a mean over two profiles, where the third profile has no cell. Rank statistics use unrounded producer-level values and retain exact ties.

	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Analysability score (%), higher better									
under DeepSeek V4 Flash	–	86.8	86.9	89.2	<u>91.9</u>	89.7	89.2	**96.3**	87.2
under HY3	90.9	<u>91.7</u>	86.3	86.9	–	**92.6**	86.8	**92.6**	86.5
under Kimi K2.7 Code	91.1	91.9	**92.2**	<u>92.0</u>	88.2	–	86.8	91.7	85.7
mean of the above	91.0†	90.1	88.5	89.4	90.0†	<u>91.2</u>†	87.6	**93.5**	86.5
Reusability source reuse share (%), higher better									
under DeepSeek V4 Flash	–	76.7	78.8	78.3	78.6	<u>80.5</u>	**82.2**	77.8	80.0
under HY3	74.7	80.2	71.4	79.1	–	79.9	<u>80.5</u>	75.9	**85.5**
under Kimi K2.7 Code	72.2	76.6	76.9	79.8	77.7	–	**84.6**	72.5	<u>82.1</u>
mean of the above	73.5†	77.8	75.7	79.1	78.2†	80.2†	<u>82.4</u>	75.4	**82.5**
Reusability construction tokens, × profile median, lower better									
under DeepSeek V4 Flash	–	0.93	<u>0.87</u>	1.50	0.90	1.07	1.21	**0.69**	1.54
under HY3	1.30	<u>0.83</u>	1.00	1.32	–	1.00	**0.82**	0.95	1.61
under Kimi K2.7 Code	1.15	<u>0.80</u>	0.90	1.10	**0.65**	–	1.15	0.88	1.47
mean of the above	1.23†	0.86	0.93	1.30	**0.78**†	1.03†	1.06	<u>0.84</u>	1.54
Testability construction tokens, × profile median, lower better									
under DeepSeek V4 Flash	–	0.92	1.08	1.20	1.11	1.21	<u>0.91</u>	<u>0.91</u>	**0.77**
under HY3	1.07	0.93	0.87	**0.60**	–	1.09	<u>0.70</u>	1.14	1.22
under Kimi K2.7 Code	**0.68**	0.98	1.24	<u>0.92</u>	1.24	–	0.93	1.11	1.02
mean of the above	<u>0.87</u>†	0.94	1.06	0.91	1.17†	1.15†	**0.85**	1.05	1.00
Measured once per artifact, independent of maintainer									
Modularity $\mathrm{LCOM}_{\mathrm{HS}}$, lower better	0.494	0.468	0.586	0.628	0.470	**0.409**	0.674	<u>0.428</u>	0.596
Stage I pass rate (%)	51.8	55.5	56.4	70.4	45.3	58.8	55.7	<u>70.6</u>	**86.2**

Table A3: Pairwise Kendall τ_a over the seven producers shared by each maintainer pair.

Outcome	**Flash–HY3**	**Flash–Kimi**	**HY3–Kimi**	**Mean**
Analysability	$+0.52$	-0.10	$+0.05$	$+0.16$
Reuse share	$+0.24$	$+0.62$	$+0.52$	$+0.46$
Reuse tokens	$+0.43$	$+0.52$	$+0.52$	$+0.49$
Test tokens	-0.33	$+0.05$	$+0.24$	-0.02

A3.2 SOURCE-FEATURE ASSOCIATION SUMMARY

Table A4 gives the exact values visualized in Figure 4. These are conditional associations, not estimates of the effect of editing the source. The behavioral-localization association under Kimi K2.7 Code has signed $r = -0.035$. A small fitted association or a non-significant test is not evidence that the underlying association is exactly zero. The block catalogue and full candidate family results remain in Appendices A4 and A6.

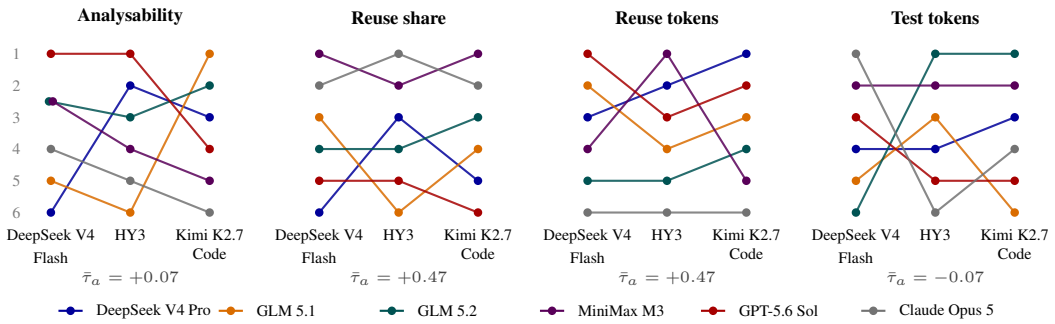

Figure A1: Producer rank under each downstream outcome, as the fixed maintainer profile changes. Ranks run from 1 (best) to 6, with average ranks for exact ties, and cover the six producers evaluated under all three profiles. The other three, which each serve as a maintainer, are excluded from this restricted view. A path that stays level is an ordering that transfers between maintainers, and $\bar{\tau}_a$ is the mean Kendall τ_a over the three profile pairs for these six producers. The two Reusability panels agree the most ($\bar{\tau}_a = +0.47$, with 12 of the 45 producer pairs changing order) and the Testability panel the least (-0.07, 24 of 45). Crossings remain common even where the coefficient is positive. Finding 2 reports the same quantity over the seven producers common to each profile pair. The four panels order the producers differently from one another, as summarized in Finding 1.

Table A4: Ablation study: principal prespecified feature blocks for the three maintenance demands, evaluated under DeepSeek V4 Flash, HY3, and Kimi K2.7 Code. Partial R^2 is the residual variation the block accounts for conditional on the outcome-specific controls. q is the within-outcome Benjamini–Hochberg value.

Demand	Outcome	Feature block and meaning	DeepSeek V4 Flash		HY3		Kimi K2.7 Code	
			Partial R^2	q	Partial R^2	q	Partial R^2	q
Analysability	Analysability score	Lexical organization: density, cross-file spread, duplication, and local control	13.80%	0.0031	16.00%	0.0157	23.28%	0.0198
Reusability	Construction tokens	Observable contract path: distance from public entry to an observable result	13.00%	0.0108	12.84%	0.0084	4.09%	0.0753
Reusability	Source reuse share	Observable contract path: distance from public entry to an observable result	8.01%	0.0095	5.53%	0.0588	11.86%	0.0098
Testability	Construction tokens	Behavior localization: layout opacity and behavior concentrated in dominant functions	5.18%	0.0406	5.50%	0.0210	0.12%	0.8082

A3.3 Additional Cohesion Analysis

Within eligible artifacts, the mean number of methods per stateful unit accounts for 22.17% of residual LCOM_{HS} variation ($p = 0.0013$, $q = 0.0052$, leave-one-case-out $r = 0.321$–0.542), conditional on case, producer-model, and language controls. Broader method sets are associated with less field sharing. This association concerns a structural quantity that also enters the definition of LCOM_{HS}. It does not establish the downstream effect of cohesion.

Three pairs of artifacts produced for the same case have reversed static and downstream orderings. Finding 3 uses the `rhit` pair as a counterexample to treating state cohesion as sufficient for easier downstream work. Taken together, these analyses motivate distinguishing internal state sharing from the behavioral surface as an interpretive framework. They neither establish two statistically independent layers nor rule out measurement noise.

A4 Ablation Block Catalogue

A4.1 Statistical model and testing

For artifact a, outcome o, downstream maintainer d, outcome-specific controls $B_{a,o}$, and artifact feature block C_a, the full model is

$$Y_{a,o,d} \sim B_{a,o} + C_a. \tag{A1}$$

The DeepSeek V4 Flash, HY3, and Kimi K2.7 Code panels are fitted separately after excluding the producer–maintainer identity pair from each panel. The controls include case, producer model, implementation language, and the outcome-specific size or correctness covariate where applicable. For a feature block C and baseline controls B, Partial R^2 is the block's unique explained variation after the controls, expressed as the fraction of variation remaining after the controls alone:

$$\text{Partial } R^2_C = \frac{R^2(B + C) - R^2(B)}{1 - R^2(B)}. \tag{A2}$$

Analysability aggregates its 50 behavioral questions into a single score per artifact and uses case, producer-model, and language controls. Reusability is already artifact-level. For Testability, outcomes from multiple tasks on the same artifact are averaged before the artifact-level analysis. Significance testing uses within-case residual permutations with Benjamini–Hochberg correction within each outcome (Benjamini & Hochberg, 1995), and leave-one-case-out ranges assess sensitivity to individual cases. Modularity has no downstream worker to ablate against. Unit method and field counts are analyzed with case, producer-model, and language controls.

Multiple ablation blocks are prespecified for each maintenance demand, chosen to represent source characteristics that may affect the corresponding downstream outcome. The semantic name is the name used in the main text. The formal ID and component list give the exact mapping to the analysis files. The catalogue contains 10 Analysability blocks, 4 Reusability blocks, and 15 Testability blocks. Metric names denote static quantities extracted from the frozen Stage I artifact. The three principal blocks are illustrated in Figure 4.

A4.2 Analysability blocks

For each Analysability block, every listed continuous metric is transformed as specified in the artifact feature extractor (count metrics use $\log(1 + x)$) and standardized over the artifact panel. The standardized metrics remain separate columns in a joint multivariate block. They are not averaged into a single block score. Block-level R^2 therefore measures the joint contribution of all listed components taken jointly.

Table A5: Analysability block composition.

Semantic block (formal ID)	Components and interpretation
Lexical organization (lexical_structure)	comment_line_ratio, functions_per_kloc, types_per_kloc, branches_per_kloc, imports_per_kloc, duplicate_line_ratio, shared_identifier_ratio, mean_identifier_file_span, long_line_ratio, and max_indent. Captures lexical density, cross-file naming, duplication, and local control structure in source.
Source concentration (source_concentration)	max_file_share, top3_file_share, mean_file_loc, max_file_loc, and source_file_count. Captures the concentration of source volume in a few files.
Repository topology (project_topology)	root_source_share, source_dir_count, max_source_depth, max_file_share, and top3_file_share. Captures source placement and hierarchy.
Structural decomposition (structural_decomposition)	source_file_count, functions_per_kloc, types_per_kloc, mean_file_loc, and max_file_share. Captures decomposition across files, functions, and types.
Lexical readability (lexical_readability)	comment_line_ratio, blank_line_ratio, long_line_ratio, mean_line_length, max_indent, and duplicate_line_ratio. Captures presentation, nesting, and duplication.
Project scale (project_size)	source_nonblank_loc, source_file_count, source_dir_count, function_count, and type_definition_count. Captures physical artifact size.
Entrypoint visibility (entrypoint_visibility)	entry_file_count, root_source_share, max_source_depth, and source_dir_count. Captures how readily an observer can locate program entry points.
Support scaffolding (support_scaffolding)	test_source_share, doc_file_count, manifest_count, and entry_file_count. Captures explicit navigational and usage scaffolding.
Control-flow density (control_flow_density)	branches_per_kloc, max_indent, and max_brace_depth. Captures local branching and nesting.
Cross-file coupling (cross_file_coupling)	imports_per_kloc, shared_identifier_ratio, mean_identifier_file_span, and source_file_count. Captures relationships that require reasoning across files.

A4.3 Reusability blocks

For each Reusability block, every listed metric is standardized over the artifact panel and retained as a separate column. The columns are residualized against the case, producer model, programming language, and source-size controls, then tested jointly as one multivariate block. No arithmetic mean is used to form the block score. The reported block-level R^2 is the joint contribution of its components.

Table A6: Reusability block composition.

Semantic block (formal ID)	Components and interpretation
Observable contract path (observable_contract_path)	command_direct_output_ratio, machine_renderer_coverage, branch_to_output_distance_mean, and observable_path_function_ratio. Captures how directly a public command reaches an observable result, including direct output and named-helper paths.
Public-entry responsibility (public_entry_observation_cost)	entry_loc_share, entry_branch_share, observable_path_function_ratio, and pure_core_function_ratio. Captures parsing, dispatch, state, behavior, and output work concentrated at the entry.
Machine-result contract (machine_result_contract)	command_direct_output_ratio, output_function_ratio, machine_renderer_coverage, and format_route_replication_fraction. Captures explicit result schemas, renderer coverage, and route duplication.
Repository navigation layout (ecosystem_navigation_layout)	root_source_share, source_dir_count, visible_file_count, file_count, max_source_depth, and entry_file_count. Captures source placement, hierarchy, visible surface, and entry exposure.

A4.4 TESTABILITY BLOCKS

Table A7: Testability block composition.

Semantic block (formal ID)	Component and interpretation
Layout recoverability (layout_recoverability)	Standardized contrast of root_source_share, max_source_depth, and dominant-file concentration (the gap between max_file_share and top3_file_share). Captures how quickly a tester can map the repository.
Behavior localization (behavior_localization_burden)	Equal directional combination of behavior monolith and layout recoverability. Captures the joint work of locating externally visible behavior and separating it into callable, assertable units of work.
Manual output assembly (manual_output_assembly_burden)	Standardized mean of machine_human_command_ratio, manual_serializer_ratio, and $\log(1 + \texttt{manual_json_output_count})$. Measures manual serialization and machine/human output assembled inside command handlers.
Output-boundary leakage (output_boundary_leakage)	Mean of standardized manual_serializer_ratio, machine_human_command_ratio, and direct_output_outside_helper_ratio. Measures output that enters command logic instead of a narrow renderer boundary.
Input-boundary leakage (input_boundary_leakage)	Mean of standardized manual_cli_outside_entry_ratio, manual_cli_branch_share, dispatch_option_interleaving_ratio, and post_dispatch_option_ratio. Measures parsing and option dispatch distributed beyond the entry.
Deep termination (deep_termination_burden)	Mean of standardized hard_termination_outside_entry_ratio, hard_termination_mixed_logic_ratio, and hard_termination_branch_share. Measures exits, panics, or fatal termination in deep branching.
Hidden context (hidden_context_burden)	Mean of standardized hidden_context_function_ratio, hidden_context_file_ratio, and hidden_context_seam_deficit. Measures dispersed environment, filesystem, process, time, and other implicit context.
Effect-boundary entanglement (effect_boundary_entanglement)	Mean of standardized effect_mixed_logic_ratio, effect_branch_share, and one minus effect_seam_parameter_ratio. Measures decision-heavy functions that mix external effects with core behavior.
Entry-core entanglement (entry_core_entanglement)	Mean of standardized entry_mixed_responsibility_ratio, entry_branch_share, and entry_effect_share. Measures entry functions retaining branching and effects instead of delegating.
Behavior monolith (behavior_monolith)	Mean of standardized largest_function_dominance, largest_branch_function_dominance, and largest_behavior_function_dominance. Measures one function dominating source, branches, and visible behavior.
Control-state complexity (control_state_burden)	Mean of standardized $\log(1 + \texttt{ccn_p90})$, $\log(1 + \texttt{argument_count_p90})$, and mutable_global_state_density_per_kloc. Measures control-flow tail, interface width, and mutable global state.
Dependency topology (dependency_topology_burden)	Mean of standardized dependency_density, max_module_fanout_ratio, cycle_node_ratio, and shared_identifier_ratio. Measures dense dependencies and cross-file coupling.
Lexical complexity (lexical_complexity_burden)	Mean of standardized mean_line_length, max_line_length, max_brace_depth, and duplicate_line_ratio. Measures long, nested, and duplicated source.
Machine-command surface (machine_command_count)	Transformed standardized machine_command_count. Counts distinct command-level source locations that expose machine-readable output.
Effect-behavior locality (effect_behavior_locality)	Standardized effect_mixed_logic_ratio. Captures the local recoverability of externally visible behavior decisions and effects.

A5 REPOSITORY CASES

The evaluation uses the following 13 repository cases from ProgramBench (Yang et al., 2026). Each paragraph names the repository, states its main operation, and gives the difficulty label from the ProgramBench catalogue. Stage I and the downstream tasks use these cases as fixed artifacts.

alexpovel/srgn. srgn performs syntax-aware search and replacement over source code in multiple programming languages. Difficulty: 3.7 (Medium).

bootandy/dust. dust analyzes disk usage and presents directory sizes as a visual hierarchy. Difficulty: 2.9 (Medium).

canop/rhit. rhit processes Nginx access logs and produces traffic summaries and statistics. Difficulty: 2.8 (Medium).

cweill/gotests. gotests reads Go source declarations and generates test-function boilerplate from their signatures. Difficulty: 2.5 (Medium).

dundee/gdu. `gdu` provides an interactive terminal interface for inspecting disk usage and navigating directory-size summaries. Difficulty: 3.5 (Medium).

guumaster/hostctl. `hostctl` manages entries in `/etc/hosts` through named profiles and command-line operations. Difficulty: 2.7 (Medium).

hairyhenderson/gomplate. `gomplate` renders templates from structured inputs and supports multiple external data-source formats. Difficulty: 4.0 (Hard).

isona/dirble. `dirble` scans web targets to discover exposed directories and files. Difficulty: 3.0 (Medium).

nachoparker/dutree. `dutree` summarizes disk usage as a colored directory tree. Difficulty: 1.7 (Easy).

naggie/dstask. `dstask` is a Git-backed command-line manager for tasks and Markdown notes. Difficulty: 3.0 (Medium).

oppiliappan/statix. `statix` analyzes Nix expressions and reports lint and diagnostic findings. Difficulty: 3.0 (Medium).

xampprocky/tokei. `tokei` counts code, comment, and blank lines across programming languages. Difficulty: 3.2 (Medium).

yoav-lavi/melody. `melody` compiles a readable pattern language into regular expressions. Difficulty: 3.0 (Medium).

A6 Model-Level Block Results

This appendix reports the model-level view of every prespecified ablation block. Component rows show producer-level means of the underlying static measurements, while association rows show the corresponding artifact-level Partial R^2 for each downstream maintainer profile, so that a block result can be traced back to the measurements that produce it. The three principal blocks discussed in the paper appear first. The remaining blocks follow in the same format.

Component columns contain artifact-level metrics averaged within each producer model. Analysability and Reusability means cover 13 artifacts per producer. Testability component means are computed once per unique Stage I artifact represented by the 12 task instances. For Testability, task outcomes for the same artifact are averaged before association rows report artifact-level Partial R^2. Headers use the same names as Table A2.

Table A8: Analysability: Lexical organization. The block represents lexical density, cross-file naming, duplication, and local control structure. Each producer mean covers 13 artifacts.

Component values (A)

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Comment lines (%)	1.56	3.30	2.97	3.79	6.02	1.46	5.21	2.35	2.25
Functions/KLOC	38.93	41.54	43.53	49.22	42.78	47.27	36.76	56.59	40.33
Types/KLOC	4.12	4.67	6.66	5.56	6.67	3.59	6.91	4.62	6.58
Branches/KLOC	276.07	230.07	288.74	243.59	250.75	242.35	225.59	414.54	201.65
Imports/KLOC	3.82	8.23	4.73	9.09	11.62	7.89	10.31	3.77	5.84

Component values (B)

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Duplicate lines (%)	21.13	20.11	19.17	17.12	16.67	15.57	18.33	5.74	17.91
Shared identifiers (%)	8.73	3.87	5.85	24.76	12.60	0.80	14.35	1.79	31.14
Mean identifier span	0.92	0.52	0.59	2.39	1.09	0.18	1.43	0.20	2.76
Long lines (%)	0.84	1.40	1.82	3.93	1.07	2.24	1.21	5.96	0.76
Maximum indent	30.08	29.85	30.38	28.62	27.08	30.38	31.31	24.38	33.92

Block result

Outcome	Profile	Partial R^2
Analysability score	DeepSeek V4 Flash	13.80%
Analysability score	HY3	16.00%
Analysability score	Kimi K2.7 Code	23.28%

The regression applies $\log(1 + x)$ to maximum indent and standardizes all 10 components over the complete 117-artifact Stage I panel. Each profile-specific model uses the resulting 104 non-diagonal observations. Partial R^2 is conditional on case, producer model, and language.

Table A9: Reusability: Observable contract path. The block represents how directly a public command reaches an observable result, including direct output and named-helper paths. Each producer mean covers 13 artifacts.

Component values

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Direct output (%)	28.38	44.33	58.32	46.25	48.72	48.38	36.93	51.24	47.66
Renderer coverage (%)	57.29	37.19	40.54	47.44	63.78	33.33	70.81	61.54	24.58
Branch-to-output distance	0.887	0.985	1.064	1.521	1.049	0.967	1.280	0.912	2.177
Observable-path functions (%)	32.32	33.84	26.28	18.61	32.00	22.93	25.25	27.81	23.74

Block result

Outcome	Profile	Partial R^2
Construction tokens	DeepSeek V4 Flash	13.00%
Construction tokens	HY3	12.84%
Construction tokens	Kimi K2.7 Code	4.09%
Source reuse share	DeepSeek V4 Flash	8.01%
Source reuse share	HY3	5.53%
Source reuse share	Kimi K2.7 Code	11.86%

Direct output is the share of public commands that emit an observable result without an intervening named helper. Branch-to-output distance is the mean number of branch levels between a command entry and the statement producing its result. All four components are standardized over the complete 117-artifact Stage I panel. Each profile-specific model uses the resulting 104 non-diagonal observations. Results control for case, producer model, implementation language, and log source size.

Table A10: Testability: Behavior localization. The block combines source-layout recoverability with the concentration of source, branches, and externally visible behavior in dominant functions. Component values are producer-level means, with source features extracted once per unique Stage I artifact.

Layout components

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Root source (%)	71.4	71.4	57.1	42.9	42.9	85.7	24.5	100.0	28.6
Maximum depth	0.29	0.29	0.57	0.57	0.71	0.14	1.00	0.00	0.71
Largest file (%)	82.5	95.5	100.0	49.4	78.7	100.0	79.0	100.0	42.5
Top 3 files (%)	95.6	99.2	100.0	72.5	89.7	100.0	88.8	100.0	68.9
Layout score	0.51	−0.02	0.06	−0.39	0.19	−0.19	0.42	−0.12	−0.45

Behavior components

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Largest function (%)	16.6	5.5	7.2	3.7	9.7	7.8	7.3	15.2	9.5
Largest branch function (%)	17.2	4.4	12.2	7.0	13.7	4.0	10.4	16.7	10.5
Largest behavior function (%)	17.3	9.7	12.0	6.1	11.9	5.7	8.9	15.7	10.6
Monolith score	0.67	−0.36	0.01	−0.46	0.14	−0.41	−0.14	0.55	0.00
Localization score	0.08	−0.17	−0.03	−0.04	−0.02	−0.11	−0.28	0.34	0.22

Block result

Outcome	Profile	Signed r	Partial R^2
Construction tokens	DeepSeek V4 Flash	0.228	5.18%
Construction tokens	HY3	0.235	5.50%
Construction tokens	Kimi K2.7 Code	−0.035	0.12%

The layout score contrasts visible internal layering with dominant-file concentration. The monolith score averages the three standardized dominance measures. The behavior-localization score is $(\text{monolith} - \text{layout})/2$, so higher values represent behavior that is harder to locate and more concentrated in dominant functions. The 96 non-diagonal task runs are first averaged within artifact before fitting the artifact-level model, which controls for case, producer model, language, and within-case Stage I pass rate.

Table A11: Analysability: Source concentration. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Largest-file share (%)	81.93	93.15	91.18	47.38	74.70	99.18	70.63	96.47	33.41
Top-3-file share (%)	96.91	99.52	96.34	72.95	87.45	99.68	82.15	99.48	62.72
Mean file LOC	874.7	1,014	924.4	367.3	616.0	936.1	811.0	459.7	392.5
Largest file LOC	1,104	1,232	1,117	743.3	731.9	1,033	977.3	558.2	1,053
Source files	2.769	1.615	2.231	8.308	4.154	1.385	6.308	1.538	10.00
Analysability score Partial R^2	DeepSeek V4 Flash: 4.88%				HY3: 4.35%		Kimi K2.7 Code: 12.04%		

Table A12: Analysability: Repository topology. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Root source (%)	59.29	71.15	53.85	31.02	30.77	78.21	36.26	95.19	30.00
Source directories	1.692	1.231	1.231	1.385	1.231	1.231	3.769	1.231	1.462
Maximum depth	0.692	0.462	0.692	0.846	0.923	0.385	1.077	0.231	0.923
Largest-file share (%)	81.93	93.15	91.18	47.38	74.70	99.18	70.63	96.47	33.41
Top-3-file share (%)	96.91	99.52	96.34	72.95	87.45	99.68	82.15	99.48	62.72
Analysability score Partial R^2	DeepSeek V4 Flash: 4.97%				HY3: 2.71%		Kimi K2.7 Code: 8.83%		

Table A13: Analysability: Structural decomposition. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Source files	2.769	1.615	2.231	8.308	4.154	1.385	6.308	1.538	10.00
Functions/KLOC	38.93	41.54	43.53	49.22	42.78	47.27	36.76	56.59	40.33
Types/KLOC	4.118	4.668	6.657	5.556	6.675	3.590	6.910	4.617	6.579
Mean file LOC	874.7	1,014	924.4	367.3	616.0	936.1	811.0	459.7	392.5
Largest-file share (%)	81.93	93.15	91.18	47.38	74.70	99.18	70.63	96.47	33.41
Analysability score Partial R^2	DeepSeek V4 Flash: 9.93%				HY3: 11.36%			Kimi K2.7 Code: 5.66%	

Table A14: Analysability: Lexical readability. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Comment lines (%)	1.563	3.295	2.972	3.790	6.022	1.460	5.208	2.353	2.255
Blank lines (%)	10.45	12.16	10.77	6.564	8.177	10.06	8.953	5.287	6.405
Long lines (%)	0.835	1.396	1.818	3.935	1.066	2.237	1.205	5.960	0.758
Mean line length	27.80	31.49	33.44	43.42	32.89	35.11	30.44	43.71	27.13
Maximum indent	30.08	29.85	30.38	28.62	27.08	30.38	31.31	24.38	33.92
Duplicate lines (%)	21.13	20.11	19.17	17.12	16.67	15.57	18.33	5.738	17.91
Analysability score Partial R^2	DeepSeek V4 Flash: 1.95%				HY3: 7.92%			Kimi K2.7 Code: 6.87%	

Table A15: Analysability: Project scale. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Nonblank LOC	1,272	1,215	1,165	1,865	1,074	940.5	1,719	637.0	3,149
Source files	2.769	1.615	2.231	8.308	4.154	1.385	6.308	1.538	10.00
Source directories	1.692	1.231	1.231	1.385	1.231	1.231	3.769	1.231	1.462
Functions	64.08	48.54	45.92	97.54	45.85	40.54	73.92	36.69	126.1
Type definitions	5.462	5.538	7.077	10.69	7.000	3.692	10.31	3.538	18.15
Analysability score Partial R^2	DeepSeek V4 Flash: 10.18%				HY3: 8.32%			Kimi K2.7 Code: 3.83%	

Table A16: Analysability: Entrypoint visibility. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Entry files	0.846	0.538	0.538	1.154	0.462	0.231	0.538	0.154	1.154
Root source (%)	59.29	71.15	53.85	31.02	30.77	78.21	36.26	95.19	30.00
Maximum depth	0.692	0.462	0.692	0.846	0.923	0.385	1.077	0.231	0.923
Source directories	1.692	1.231	1.231	1.385	1.231	1.231	3.769	1.231	1.462
Analysability score Partial R^2	DeepSeek V4 Flash: 2.11%				HY3: 0.56%			Kimi K2.7 Code: 5.45%	

Table A17: Analysability: Support scaffolding. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Test source (%)	1.923	0.000	2.564	0.496	0.855	2.564	0.721	1.923	0.615
Documentation files	4.462	4.462	4.462	4.462	4.462	4.462	4.462	4.462	4.692
Manifests	0.923	0.615	0.692	0.846	0.462	0.385	0.538	0.308	0.923
Entry files	0.846	0.538	0.538	1.154	0.462	0.231	0.538	0.154	1.154
Analysability score Partial R^2	DeepSeek V4 Flash: 2.53%				HY3: 6.44%			Kimi K2.7 Code: 5.40%	

Table A18: Analysability: Control-flow density. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Branches/KLOC	276.1	230.1	288.7	243.6	250.8	242.3	225.6	414.5	201.7
Maximum indent	30.08	29.85	30.38	28.62	27.08	30.38	31.31	24.38	33.92
Maximum brace depth	7.154	4.000	5.385	7.154	4.769	3.846	5.615	3.231	8.462
Analysability score Partial R^2	DeepSeek V4 Flash: 1.26%				HY3: 7.50%		Kimi K2.7 Code: 17.34%		

Table A19: Analysability: Cross-file coupling. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Imports/KLOC	3.822	8.230	4.733	9.087	11.62	7.891	10.31	3.769	5.845
Shared identifiers (%)	8.729	3.873	5.853	24.76	12.60	0.803	14.35	1.787	31.14
Mean identifier span	0.918	0.520	0.593	2.393	1.093	0.183	1.431	0.200	2.756
Source files	2.769	1.615	2.231	8.308	4.154	1.385	6.308	1.538	10.00
Analysability score Partial R^2	DeepSeek V4 Flash: 4.17%				HY3: 3.99%		Kimi K2.7 Code: 10.09%		

Table A20: Reusability: Repository navigation layout. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Root source (%)	59.29	71.15	53.85	31.02	30.77	78.21	36.26	95.19	30.00
Source directories	1.69	1.23	1.23	1.38	1.23	1.23	3.77	1.23	1.46
Visible files	17.08	15.85	16.31	23.46	18.08	15.15	21.62	15.15	26.08
Files	2.77	1.62	2.23	8.31	4.15	1.38	6.31	1.54	10.00
Maximum depth	0.69	0.46	0.69	0.85	0.92	0.38	1.08	0.23	0.92
Entry files	0.85	0.54	0.54	1.15	0.46	0.23	0.54	0.15	1.15
Source reuse share Partial R^2	DeepSeek V4 Flash: 11.45%				HY3: 10.25%		Kimi K2.7 Code: 7.63%		
Construction tokens Partial R^2	DeepSeek V4 Flash: 6.03%				HY3: 5.97%		Kimi K2.7 Code: 3.97%		

Table A21: Reusability: Public-entry Responsibility. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Entry LOC share (%)	23.26	24.33	25.47	16.11	28.58	22.19	20.93	23.06	17.20
Entry branch share (%)	25.52	25.48	24.44	14.65	31.63	23.38	22.38	23.11	16.74
Observable-path functions (%)	32.32	33.84	26.28	18.61	32.00	22.93	25.25	27.81	23.74
Pure-core functions (%)	51.92	57.19	58.32	56.55	53.96	61.31	66.20	62.54	52.32
Source reuse share Partial R^2	DeepSeek V4 Flash: 0.77%				HY3: 3.67%		Kimi K2.7 Code: 8.49%		
Construction tokens Partial R^2	DeepSeek V4 Flash: 11.72%				HY3: 3.13%		Kimi K2.7 Code: 7.30%		

Table A22: Reusability: Machine-result contract. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Direct output (%)	28.38	44.33	58.32	46.25	48.72	48.38	36.93	51.24	47.66
Output functions (%)	24.75	25.32	19.08	12.47	14.87	16.70	16.80	22.19	10.81
Renderer coverage (%)	57.29	37.19	40.54	47.44	63.78	33.33	70.81	61.54	24.58
Route replication (%)	3.846	0.233	3.846	7.890	3.846	0.350	4.188	0.000	7.692
Source reuse share Partial R^2	DeepSeek V4 Flash: 10.77%				HY3: 5.08%		Kimi K2.7 Code: 6.78%		
Construction tokens Partial R^2	DeepSeek V4 Flash: 6.25%				HY3: 9.44%		Kimi K2.7 Code: 0.89%		

Table A23: Testability: Layout recoverability. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Root source (%)	71.43	71.43	57.14	42.86	42.86	85.71	24.49	100.0	28.57
Maximum depth	0.286	0.286	0.571	0.571	0.714	0.143	1.000	0.000	0.714
Largest-file share (%)	82.48	95.49	100.0	49.36	78.68	100.0	79.04	100.0	42.47
Top-3-file share (%)	95.60	99.16	100.0	72.53	89.71	100.0	88.82	100.0	68.87
Construction tokens Partial R^2	DeepSeek V4 Flash: 3.62%				HY3: 1.03%			Kimi K2.7 Code: 0.35%	

Table A24: Testability: Manual output assembly. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Manual serializer (%)	63.81	41.74	39.52	26.67	39.29	20.95	60.71	28.57	53.57
Machine/human commands (%)	85.71	57.14	71.43	57.14	71.43	42.86	50.00	85.71	61.90
Manual JSON output sites	1.286	3.857	2.571	3.857	1.143	0.857	1.429	1.429	1.429
Construction tokens Partial R^2	DeepSeek V4 Flash: 7.17%				HY3: 0.14%			Kimi K2.7 Code: 0.01%	

Table A25: Testability: Output-boundary leakage. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Manual serializer (%)	63.81	41.74	39.52	26.67	39.29	20.95	60.71	28.57	53.57
Machine/human commands (%)	85.71	57.14	71.43	57.14	71.43	42.86	50.00	85.71	61.90
Output outside helper (%)	46.79	65.95	72.56	54.95	49.30	48.19	43.51	86.33	88.31
Construction tokens Partial R^2	DeepSeek V4 Flash: 10.29%				HY3: 0.01%			Kimi K2.7 Code: 0.04%	

Table A26: Testability: Input-boundary leakage. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
CLI outside entry (%)	14.29	7.143	14.29	21.43	22.14	14.29	14.29	0.000	21.43
CLI branch share (%)	22.63	5.690	13.98	4.625	20.86	5.022	12.51	20.80	8.974
Option interleaving (%)	0.000	0.000	0.000	0.000	0.000	14.29	0.000	14.29	0.000
Post-dispatch options (%)	28.57	2.857	0.000	0.000	14.29	9.091	0.000	10.20	0.000
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.37%				HY3: 0.06%			Kimi K2.7 Code: 0.84%	

Table A27: Testability: Deep termination. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Termination outside entry (%)	14.29	15.24	16.67	37.53	28.57	27.82	22.08	41.07	47.14
Termination mixed logic (%)	50.00	64.29	85.71	39.68	35.71	61.90	28.57	38.10	39.73
Termination branch share (%)	19.39	17.67	16.12	4.706	2.807	13.34	2.127	14.64	7.351
Construction tokens Partial R^2	DeepSeek V4 Flash: 1.73%				HY3: 0.01%			Kimi K2.7 Code: 0.04%	

Table A28: Testability: Hidden context. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Hidden-context functions (%)	6.651	9.585	3.898	2.083	2.525	3.934	1.860	4.505	2.405
Hidden-context files (%)	42.86	78.57	42.86	21.04	44.44	57.14	32.65	57.14	28.21
Hidden-context seam deficit	0.157	0.429	0.155	0.250	0.500	0.405	0.143	0.071	0.360
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.35%				HY3: 1.54%			Kimi K2.7 Code: 0.00%	

Table A29: Testability: Effect-boundary entanglement. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Effect mixed logic (%)	81.55	96.43	95.92	94.05	79.37	94.76	94.13	98.57	76.79
Effect branch share (%)	39.41	36.07	22.43	19.02	26.13	23.46	24.64	37.31	18.56
Effect seam parameters (%)	66.64	69.13	75.41	77.38	53.33	77.01	66.67	94.29	64.25
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.05%				HY3: 0.67%			Kimi K2.7 Code: 0.26%	

Table A30: Testability: Entry-core entanglement. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Entry mixed responsibility (%)	80.25	76.53	80.20	44.88	59.29	67.62	71.10	100.0	47.02
Entry branch share (%)	35.34	23.09	30.28	15.46	30.22	20.66	26.61	27.48	19.11
Entry effects (%)	45.93	39.35	38.84	39.64	21.63	48.98	39.82	34.97	23.85
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.20%				HY3: 0.01%			Kimi K2.7 Code: 0.09%	

Table A31: Testability: Behavior monolith. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Largest function dominance (%)	16.57	5.534	7.168	3.662	9.656	7.843	7.295	15.18	9.470
Largest branch-function dominance (%)	17.21	4.407	12.23	7.008	13.72	4.023	10.41	16.66	10.48
Largest behavior-function dominance (%)	17.27	9.720	11.96	6.077	11.91	5.686	8.904	15.71	10.63
Construction tokens Partial R^2	DeepSeek V4 Flash: 1.56%				HY3: 3.99%			Kimi K2.7 Code: 0.31%	

Table A32: Testability: Control-state complexity. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
CCN p90	18.89	18.24	17.30	11.70	18.51	14.53	18.83	24.59	11.77
Argument count p90	3.571	4.443	3.029	4.014	4.486	4.086	3.971	3.229	2.929
Mutable global state/KLOC	1.660	1.181	1.775	0.436	1.166	0.000	1.521	0.845	1.579
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.50%				HY3: 0.00%			Kimi K2.7 Code: 0.37%	

Table A33: Testability: Dependency topology. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Dependency density	0.000	0.048	0.000	0.059	0.020	0.000	0.064	0.000	0.056
Module fanout (%)	0.000	14.29	0.000	40.82	14.29	0.000	14.29	0.000	42.60
Cycle nodes (%)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Shared identifiers (%)	6.355	2.249	0.000	21.79	10.25	0.000	10.45	0.000	28.60
Construction tokens Partial R^2	DeepSeek V4 Flash: 0.68%				HY3: 0.09%			Kimi K2.7 Code: 0.55%	

Table A34: Testability: Lexical complexity. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Mean line length	29.71	30.18	34.34	41.48	32.65	36.06	32.56	45.92	27.57
Maximum line length	150.4	171.0	173.7	1,650	223.1	605.3	156.6	537.3	899.0
Maximum brace depth	5.429	4.143	4.286	7.000	5.429	3.143	5.143	1.714	8.000
Duplicate lines (%)	19.34	22.87	17.66	13.59	17.77	15.68	16.53	4.368	18.57
Construction tokens Partial R^2	DeepSeek V4 Flash: 3.58%				HY3: 0.00%		Kimi K2.7 Code: 0.42%		

Table A35: Testability: Machine-command surface. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Machine commands	2.714	0.714	0.857	0.857	1.857	1.429	0.714	1.143	1.571
Construction tokens Partial R^2	DeepSeek V4 Flash: 9.86%				HY3: 0.04%		Kimi K2.7 Code: 0.54%		

Table A36: Testability: Effect-behavior locality. Component means and block associations.

Component	DeepSeek V4 Flash	DeepSeek V4 Pro	GLM 5.1	GLM 5.2	HY3	Kimi K2.7 Code	MiniMax M3	GPT-5.6 Sol	Claude Opus 5
Effect mixed logic (%)	81.55	96.43	95.92	94.05	79.37	94.76	94.13	98.57	76.79
Construction tokens Partial R^2	DeepSeek V4 Flash: 2.33%				HY3: 0.02%		Kimi K2.7 Code: 3.33%		

A7 Concrete Downstream Task Examples

This appendix gives one complete downstream task instance for each of the three maintenance demands. Each example states the repository context, the request supplied to the agent, the required processing steps, and the resulting interface. The examples use the same task contracts as the evaluation runs.

A7.1 Analysability: Melody Pattern Compilation and Validation

Background. `yoav-lavi/melody` provides a readable pattern language that is compiled into regular expressions and then used to match strings. The task examines the observable relationship between a pattern, its compiled expression, and a collection of samples.

Request. The input contains an ordered array of policy rows. Each row has an identifier `id`, a Melody pattern `pattern`, and an array of string `samples`. Patterns may contain literal route text, pattern operators, and wildcard constructs. Samples include strings that match the pattern and strings that do not.

Processing. For each policy, the executable first compiles the Melody pattern. Every sample is then tested against the same compiled pattern. The policy order and sample order are retained. Compilation output is normalized by removing its trailing line ending. A compilation failure or a sample-evaluation error terminates the request with a failure status.

Output. The response contains the policies in input order. Each policy contains its identifier, compiled regular expression, and sample results. A sample result contains the original sample string and a Boolean `matched` field. The response also contains `all_compiled`, which is true when every policy has been compiled successfully.

A7.2 REUSABILITY: RHIT INCIDENT-TRIAGE CONSUMER

Background. `canop/rhit` reads Nginx access logs and provides filtering and projection operations over the recorded requests. The downstream task packages these operations in a JSON interface for incident triage.

Request. The consumer receives one JSON object containing a log path and an ordered array of queries. A query selects records by HTTP status or request method and has the form {`kind`, `value`}, where `kind` is `status` or `method`.

Processing. Each query is executed independently through the existing Rhit executable. Selected records are normalized to string `method`, string `path`, string `status`, and integer `bytes_sent`. Records returned by multiple queries are merged and duplicate rows are removed. The merged records are sorted by path, method, and status. The consumer then counts the records and checks whether any selected status belongs to the HTTP 5xx class.

Output. The success response contains exactly `incidents`, `incident_count`, and `has_server_error`. `incidents` is the normalized, deduplicated, sorted record array. `incident_count` is its length, and `has_server_error` is a Boolean derived from the selected status values. The consumer reads one JSON document from standard input, writes one JSON document to standard output, and returns a non-zero status for malformed or schema-invalid input.

A7.3 TESTABILITY: TOKEI STREAMING AND FILTERING TESTER

Background. `xampprocky/tokei` counts code, comment, and blank lines by programming language. Its command-line interface exposes both a line-oriented streaming JSON surface and a document-oriented report surface. File visibility, filtering, and symbolic-link handling are part of the observable behavior.

Fixture and scenarios. The tester creates a fresh fixture containing files from multiple languages, an empty file, hidden files, ignore files, an excludeable directory, a Markdown file, and a symbolic link. Seven scenarios run against this fixture: full streaming output, full report output, ignore-rule comparison, exclude-pattern filtering, language allow-list filtering, symbolic-link handling, and the corresponding output-surface comparisons.

Processing. For every scenario, the tester invokes the supplied executable with the specified arguments and captures stdout, stderr, and the process exit code. Streaming output is parsed line by line. Report output is parsed as one JSON document. File records are normalized to relative paths and retain language, blank-line, code-line, and comment-line counts. Language-level file counts and totals are derived from the normalized records. The tester computes path and statistic differences introduced by ignore rules, exclude patterns, language filters, and the two output surfaces.

Output. The result records the scenario order and, for every scenario, its process outcome, parsed records, language totals, report language keys, and parse-error count. Relation fields contain the ordered path sets and statistic differences for the declared scenario pairs. The tester emits one JSON result document after all seven scenarios have been processed.