SWE-Prometheus: Measuring Engineering Governance Improvements in Real-World Repositories

CosmosMind AI Lab

CosmosMind 清華大學 北京大學

Abstract: Large language model based coding agents have made substantial progress on repository-level software engineering tasks. Nevertheless, existing repository-level benchmarks typically start from an issue that has already been identified by a human and evaluate whether a target patch satisfies a relatively explicit functional signal. This setting measures targeted repair well, but does not adequately evaluate the broader engineering work required to make repositories testable, maintainable, reproducible, and safe to modify. We present **SWE-Prometheus**, a benchmark designed to evaluate repository-level engineering governance under realistic constraints. Each task gives an agent a fixed snapshot of a real repository and a general governance objective, but no defect list, base score, oracle patch, or hidden-test description. The agent must discover relevant risks, prioritize interventions, implement a retrofit, and provide executable evidence that the treated repository remains useful and behaviorally compatible. SWE-Prometheus evaluates six governance dimensions—Tests & CI, Code Quality Gates, Documentation & Collaboration, Structure & Maintainability, Reproducible Environment, and Dependency & Security Health—through paired base/treated evidence, clean-environment probes, characterization gates, and multi-teacher scoring. The benchmark contains 60 repositories; ten models are evaluated on a shared 22-repository public subset, where mean Normalized Governance Improvement ranges from 0.0568 to 0.5760 and observed behavior-breakage rates range from 0% to 23%. On a frozen ten-repository auxiliary batch, the no-op noise floor has standard deviation 0.073, while mechanical and rule-based baselines obtain mean NGI of 0.272 and 0.254. These results position SWE-Prometheus as a focused resource for studying whether coding agents can improve the engineering conditions of real repositories rather than merely produce patches that pass a target test.

1. Introduction

Large language model based coding agents have made substantial progress on repository-level software engineering. They can search unfamiliar codebases, execute development commands, edit multiple files, and produce end-to-end changes. Repository-level benchmarks such as SWE-bench made this progress measurable by pairing a real repository snapshot with a concrete issue and executable validation [1]. Agent scaffolds further showed that the interface through which a model searches, edits, and tests a repository can materially affect the outcome [2].

As illustrated in Figure 1, however, most existing evaluations begin after a human has already identified the problem. The issue statement specifies what the agent should investigate, while tests or a reference patch provide a relatively clear success signal. This is appropriate for targeted functional repair, but it leaves a central part of software engineering unmeasured: making a repository easier to test, maintain, reproduce, and safely evolve. In practice, these properties are distributed across tests, CI, tooling, documentation, project structure, environments, and dependencies. They rarely appear as one localized defect, and their improvement cannot be established from a diff alone.

The resulting gap is between engineering artifacts and engineering outcomes. An agent may add a workflow, a lint configuration, or a test file without making the repository more useful; it may also improve one dimension while introducing dependency drift or behavioral regressions elsewhere. A suitable evaluation must therefore give the agent room to diagnose the repository, compare evidence before and after intervention, and verify that improvements survive a clean execution environment.

To address this gap, we introduce **SWE-Prometheus**, a benchmark for autonomous repository-level engineering governance. Each task starts from a fixed repository snapshot and a general retrofit objective. The agent is not given a defect list, base score, hidden-test description, or target patch. It must inspect the repository, identify consequential gaps, prioritize interventions, and implement useful changes under a limited budget. The evaluator

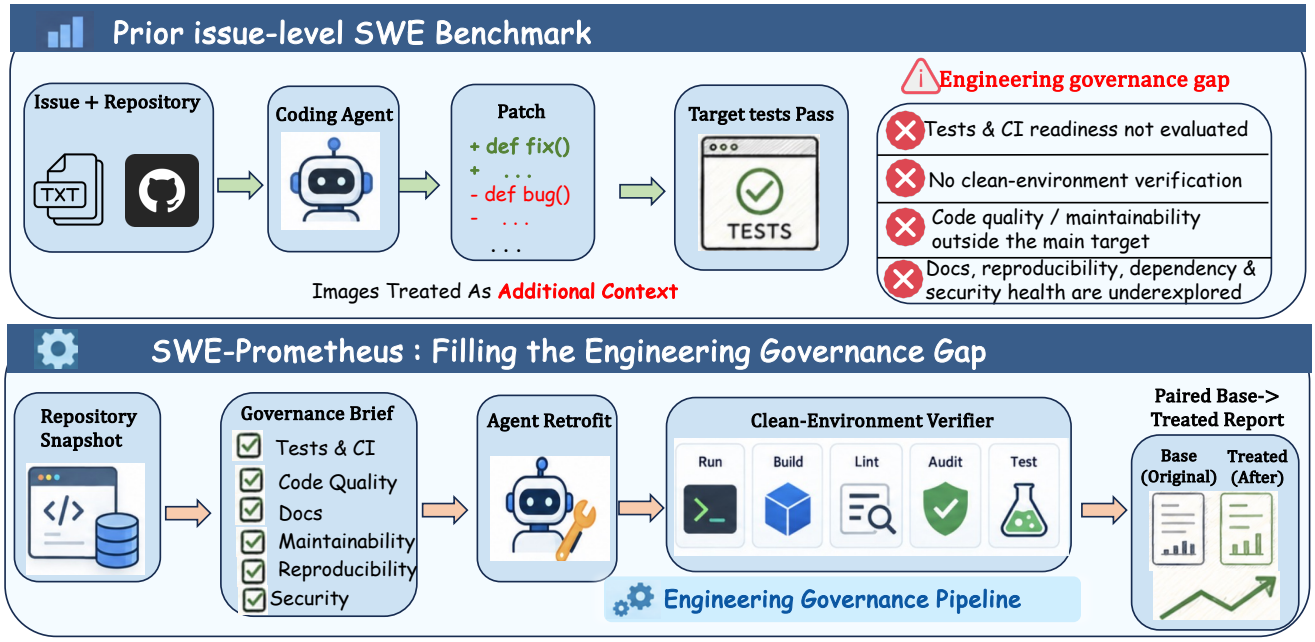


Figure 1: Motivation for SWE-Prometheus. Prior issue-level SWE benchmarks hand the agent a localized defect and score a patch against target tests, leaving test and CI readiness, clean-environment behavior verification, quality and maintainability outside the patch target, and documentation, reproducibility, dependency, and security health unevaluated. SWE-Prometheus replaces the issue with a governance brief, requires the agent to diagnose and retrofit the repository itself, and scores a paired base-to-treated report produced by a clean-environment verifier.

then reconstructs the base and treated states, executes the same probes in a clean environment, and applies behavior gates to identify unsafe interventions. SWE-Prometheus consequently evaluates whether an agent can leave a repository in a more trustworthy engineering state.

The benchmark covers six complementary governance dimensions: Tests & CI, Code Quality Gates, Documentation & Collaboration, Structure & Maintainability, Reproducible Environment, and Dependency & Security Health. Each dimension is scored from executable evidence and structured judgments, while applicability, unavailable evidence, gate strength, and behavior status remain explicit. The current release contains 60 real repositories, including 22 public instances shared across ten model rollouts and 38 private instances reserved for internal evaluation. On this shared public subset, mean Normalized Governance Improvement ranges from 0.0568 to 0.5760, while observed behavior-breakage rates range from 0% to 23%.

The auxiliary study provides controls for interpreting these differences. On a frozen ten-repository batch, the no-op condition has NGI standard deviation 0.073; mechanical and rule-based retrofits reach mean NGI of 0.272 and 0.254. Only two of ten models beat the stronger template baseline on a clear majority of repositories. Gate ablations show model-dependent effects, while measured breakage decreases as mutation-tested gate strength weakens, revealing the importance of verifier quality. These results show that repository-governance evaluation must report improvement, behavior validity, non-agent baselines, and measurement strength together.

In summary, our contributions are threefold:

- We identify repository-level engineering governance as a distinct evaluation setting that complements issue-level functional repair, and formulate it as an open-ended Repository Retrofit task without a defect list or oracle patch.
- We introduce SWE-Prometheus, a repository-centered benchmark with six governance dimensions, paired base/treated evidence, clean-environment probes, behavior gates, public/private splits, and machine-readable release records.

- We benchmark ten coding agents and establish diagnostic controls—including no-op, mechanical, rule-based, matched, and gate-strength analyses—that separate governance improvement from template effects, behavior risk, and verifier limitations.

2. Related Work

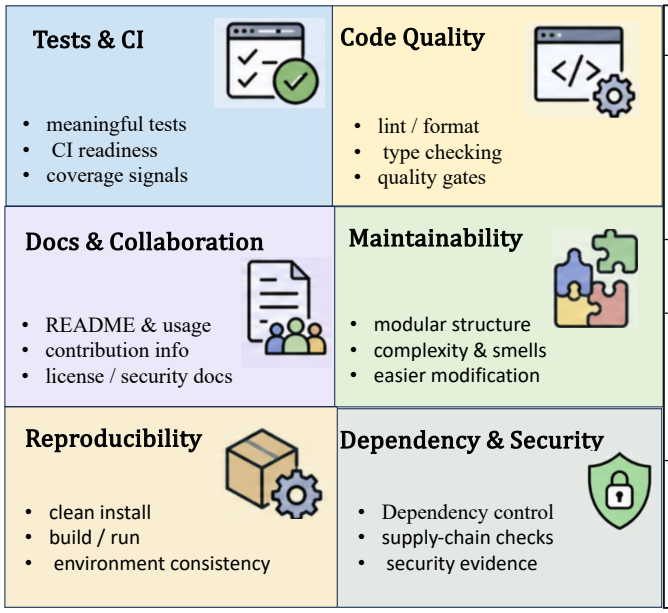
Repository-level software engineering benchmarks. SWE-bench established the now-standard setting in which an agent receives a real issue, a repository snapshot, and executable tests [1]. SWE-bench Verified adds human validation of task solvability and test quality [3], while SWE-bench-java, Multi-SWE-bench, SWE-PolyBench, and SWE-bench Multimodal extend the setting to new languages, broader repository distributions, and visual evidence [4–7]. Recent work also studies freshness and contamination-aware collection through SWE-bench-Live and SWE-rebench [8, 9]. SWE-bench Science and Rust-SWE-bench further show that domain-specific repositories and ecosystems expose failure modes that are hidden by a narrow benchmark distribution [10, 11]. SWE-Prometheus shares fixed revisions, isolated execution, patches, and executable checks, but changes the object of evaluation from fixing a known defect to diagnosing and improving repository engineering state.

Agents for repository exploration and repair. Agent interfaces and repository navigation are themselves important determinants of performance. SWE-agent studies how a purpose-built agent-computer interface supports search, editing, and testing [2]; OpenHands provides a general platform for software-development agents [12]; and Agentless tests a deliberately simple localization, repair, and validation pipeline [13]. RepoBench and RepoCoder focus on repository-level retrieval and completion rather than issue resolution [14, 15], while SWE-Explore isolates repository exploration as a measurable capability [16]. RepairAgent and RepoAgent investigate autonomous program repair and repository-level development workflows [17, 18]. These systems motivate our separation of reconnaissance, intervention, and verification, but they generally assume a known target issue or evaluate a patch-centric endpoint. SWE-Prometheus instead asks the agent to discover the engineering target itself.

Beyond patch correctness. Executable patch success is necessary but insufficient for evaluating software engineering quality. SWT-Bench explicitly tests whether generated bug fixes are validated against real-world behavior [19]; RepoExec studies repository-level executable evaluation [20]; and EvalPlus demonstrates that stronger test suites can substantially change conclusions about functional correctness [21]. Mutation testing provides a complementary way to estimate whether a test suite can detect meaningful changes [22], while studies of flaky tests show that instability in the test infrastructure can contaminate engineering conclusions [23]. Our behavior gates and gate-strength labels build on this line of work by treating preservation evidence as a first-class outcome rather than a hidden prerequisite.

Software quality, maintainability, and technical debt. Software quality models such as ISO/IEC 25010 organize quality into maintainability, reliability, security, and related properties [24]. Complexity metrics [25, 26], technical-debt research [27, 28], and empirical studies of continuous integration [29, 30] provide useful measurement primitives. However, these approaches typically audit a property, a metric, or a development practice in isolation. They do not evaluate whether an autonomous agent can identify the most consequential repository gaps, coordinate changes across several dimensions, and preserve existing behavior. In SWE-Prometheus, these tools are evidence probes inside a paired base-to-treated evaluation, not substitutes for the benchmark objective.

Dependencies, security, and reproducibility. Modern repository health also depends on dependency scope, supply-chain provenance, and reproducible environments. OpenSSF Scorecard, SLSA, and NIST SSDF provide practical security and supply-chain guidance [31–33]. Empirical work shows that dependency alerts do not all have the same production relevance, making naive alert counts an inadequate proxy for risk [34]. These resources inform our Dependency & Security Health dimension and our decision to record applicability and unavailable evidence explicitly. The benchmark evaluates whether an agent can turn such practices into executable repository improvements rather than merely adding configuration files.



Statistic	Number
Total Repositories	60
Public Instances	22
Private Instances	38
Model Rollouts (public)	10
Public Model-Instance Pairs	220
Governance Dimensions	6
Auxiliary Batch Size	10
Detected Gates	24
Blind Gates	27
Vacuous Gates	3
No Gate	6
Best Public NGI	0.5760
No-op Noise SD	0.073
Mechanical Baseline NGI	0.272
Rule-based Baseline NGI	0.254

Figure 2: Dataset and evaluation scope. The benchmark covers six governance dimensions and reports the public/private split, behavior-gate composition, and completed auxiliary-batch statistics.

Evaluation reliability and judge calibration. Weak tests, leakage, incomplete curation, and unstable environments can change model conclusions. Benchmark construction studies show that automated setup and broader repository coverage can produce distributions that differ materially from popular curated suites [35]. Contamination-limited evaluation and benchmark governance therefore require explicit provenance and release controls [36–38]. For subjective dimensions, LLM-as-a-judge methods require calibration and agreement checks rather than unqualified trust [39, 40]. SWE-Prometheus follows these principles by retaining failed runs, measuring characterization-test strength through mutation testing, separating public and private data, and preserving per-run evidence. The remaining gap is an integrated evaluation in which the agent itself must discover the engineering target.

3. Dataset Overview

3.1. Task formulation and scope

Each instance is a fixed snapshot of a real GitHub repository identified by `repo` and `base_commit`. The agent receives a general instruction:

> Inspect this repository and improve its engineering readiness across the applicable governance dimensions. Preserve existing behavior and public interfaces. Make useful changes for future users and maintainers, and verify important claims with executable checks.

The task provides no issue list, base score, hidden test, or reference patch. Multiple governance strategies may be valid. The agent can inspect source code and public documentation, execute development tools, edit project files, and submit a unified patch, but cannot access evaluator evidence or other model results.

Governance dimensions. The six dimensions are scored from 1 to 5; a score of 4 denotes acceptable governance for the repository type and task scope. Figure 2 visualizes their scope: Tests & CI, Code Quality Gates, Documentation & Collaboration, Structure & Maintainability, Reproducible Environment, and Dependency & Security. Configuration presence alone is never sufficient evidence of improvement; each dimension must be supported by executable or structured evidence.

Evidence loop. The evaluator reconstructs the base repository, applies the agent patch to an independent treated snapshot, and runs the same installation, test, quality, complexity, dependency, and security probes on both states. Commands, exit codes, logs, environment versions, timeouts, and unavailable evidence are retained.

Behavior gate. Characterization tests are validated on the base snapshot and then executed on the treated snapshot. A failed treated verification is labeled `behavior_broken`; it is excluded from valid NGI aggregation but retained in behavior-risk analysis. Mutation testing labels a gate as `detected`, `blind`, `vacuous`, or `none`.

Metrics. For instance i and dimension d, raw improvement is $\Delta_{i,d} = s_{i,d}^{treated} - s_{i,d}^{base}$. Normalized Governance Improvement is

$$NGI_i = \frac{1}{|D_i|} \sum_{d \in D_i} \frac{s_{i,d}^{treated} - s_{i,d}^{base}}{5 - s_{i,d}^{base}},$$

where D_i contains defined and scorable dimensions. A dimension with base score 5 receives no improvement reward but is checked for regression. We additionally report treated scores, valid and broken rates, no-regression rate, strict success, verified claim rate, and resource usage.

4. Dataset Creation

4.1. Repository selection and screening

SWE-Prometheus is intentionally repository-centered rather than issue-centered. We first collect candidate repositories with evidence of active development and at least one plausible governance gap. A candidate is retained only when (i) the base commit can be reconstructed, (ii) the project has meaningful executable behavior, (iii) a clean evaluation environment can be built without private credentials, (iv) at least two governance dimensions are applicable, and (v) the task can be described without prescribing a particular patch. Screening records language, build system, test entry points, dependency manager, license, commit age, and external-service requirements.

Discovery signals such as absent workflows, stale metadata, weak tests, and unpinned dependencies are used for high-recall candidate collection, not as final scores. The final score is based on executable evidence. Negative evidence is retained explicitly: an unavailable command, unsupported platform, or inapplicable dimension is recorded rather than silently treated as a zero-quality score.

4.2. Evidence collection and task packaging

For every retained repository, the construction process records a fixed repository identifier and base commit, a governance brief, applicable dimensions, baseline evidence, characterization commands, and environment metadata. The governance brief states the engineering objective without naming a defect or prescribing a patch. This preserves room for multiple valid retrofit strategies while keeping the task reproducible.

The evaluator reconstructs the base repository, runs installation and development probes, and stores commands, exit codes, logs, tool versions, timeouts, and unavailable evidence. The same evidence schema is used after treatment. This paired design prevents a configuration file from being counted as an improvement unless the corresponding command or structured check provides useful evidence.

4.3. Execution protocol

Every rollout starts from a fresh copy of the pinned base commit. The agent receives the same task instruction and repository snapshot for a given comparison set. The harness records tool calls, changed files, patch size, wall-clock time, token usage when exposed by the scaffold, and termination reason. After exit, the harness validates the patch and applies it to a second clean copy. Evaluation runs installation, characterization behavior, governance probes, and post-hoc artifact validation in that order.

Each command has a timeout, an exit-code policy, and an evidence schema. Logs are retained with tool versions and environment identifiers. Monetary cost is reported only when provider accounting is available; it is never

imputed from incomplete accounting. This protocol separates infrastructure failures from engineering failures and makes each reported score auditable.

4.4. Behavior gates and failure taxonomy

Characterization tests are validated on the base snapshot and then executed on the treated snapshot. A failed treated verification is labeled `behavior_broken`; it is excluded from valid NGI aggregation but retained in behavior-risk analysis. Mutation testing labels a gate as `detected`, `blind`, `vacuous`, or `none`. The gate-strength label is reported alongside every behavior outcome because a zero-breakage result from a weak gate should not be interpreted as strong safety evidence.

We distinguish infrastructure failure, environment/dependency failure, invalid or empty patch, behavior regression, preserved behavior without governance improvement, and preserved behavior with positive improvement. A rollout may receive a primary failure label and secondary evidence flags. For example, a patch may preserve characterization behavior while introducing a vacuous CI gate; it should not be counted as a clean success merely because tests pass.

4.5. Scoring and judge calibration

Executable probes provide primary evidence for tests, CI, environment, and dependency dimensions. Documentation and maintainability use dimension-specific rubrics rather than a holistic impression. Each candidate score is accompanied by positive evidence, negative evidence, applicability, and a confidence flag. Two independent judgments are preferred; disagreements are adjudicated only after initial labels are preserved.

The calibration set is scored by independent evaluators who do not know model identity. We report agreement, adjudication rate, and sensitivity to rubric anchors, together with judge-family and prompt-order checks. These safeguards are part of the release protocol and are distinguished from completed benchmark results.

5. Results

5.1. Experimental setting and dataset

The current SWE-Prometheus release contains 60 real repository instances: 22 public instances shared by all current models and 38 private instances reserved for internal evaluation. Ten models have completed one rollout on the public subset, yielding 220 public model-instance pairs. The public subset is a shared comparison set rather than a random sample of all repositories.

Candidate repositories were discovered using signals such as missing or failing CI, thin tests, absent quality gates, incomplete collaboration documentation, unpinned dependencies, and maintenance gaps. These signals support high-recall discovery and stratification, while screening confirms meaningful functionality, a stable base commit, plausible governance headroom, and evaluation without private credentials or paid services.

The evaluation follows a paired base-to-treated design. For every model-instance pair, the harness first records baseline evidence, then applies the submitted patch to a clean copy and reruns the same probes. Characterization behavior is checked before governance scores are aggregated. We report valid and behavior-broken outcomes separately, retain unavailable evidence instead of converting it into a zero, and use the shared public subset for the primary cross-model comparison. Instance-level records for every model and condition, together with the full result tables underlying the figures in this section, are given in Appendix D.

5.2. Auxiliary conditions

Model averages alone cannot separate capability from measurement artifact, so we additionally run a compact auxiliary study on ten repositories sampled from the public set with fixed strata for repository size, baseline quality, and gate availability. Every condition uses the same base snapshots and the same scoring pipeline, and all conditions reuse stored rollouts without issuing new model calls.

The batch includes a *no-op* condition that pushes an empty patch through the full pipeline in order to estimate scorer noise; a *mechanical* retrofit that adds a fixed governance template without inspecting the repository; a

Agent system	Scaffold	Valid	Broken	Breakage (%)	NGI mean
GPT-5.6-Sol	pi	21	1	4.5	0.2956
Claude-Opus-5	Claude Code	18	4	18.2	0.5293
Kimi-K3	pi	21	1	4.5	0.5760
GLM-5.3-Flash	pi	17	5	22.7	0.5760
Qwen3.8-Max	pi	18	4	18.2	0.4630
GLM-5.2	pi	20	2	9.1	0.4438
DeepSeek-V4-Pro	pi	18	4	18.2	0.3803
GLM-5.3	pi	19	3	13.6	0.3198
DeepSeek-V4-Flash	pi	22	0	0.0	0.2083
MiniMax-M3	pi	22	0	0.0	0.0568

Table 1: Public shared-subset results. Valid and broken counts are out of 22 instances. NGI is aggregated only over valid runs, which preserve characterization behavior and contain scorable evidence. Claude used Claude Code; the other models used pi.

rule-based retrofit that first detects which artifacts are missing and adds only those, generating a real import smoke test rather than a vacuous assertion; and the existing model rollouts evaluated on the same instances. A separate behavior-gate ablation uses all 60 instances because the public subset contains no `none` gate cases. Appendix C.5 documents the strata, the condition matrix, and the patch contents of each baseline.

5.3. Governance improvement varies widely across agents

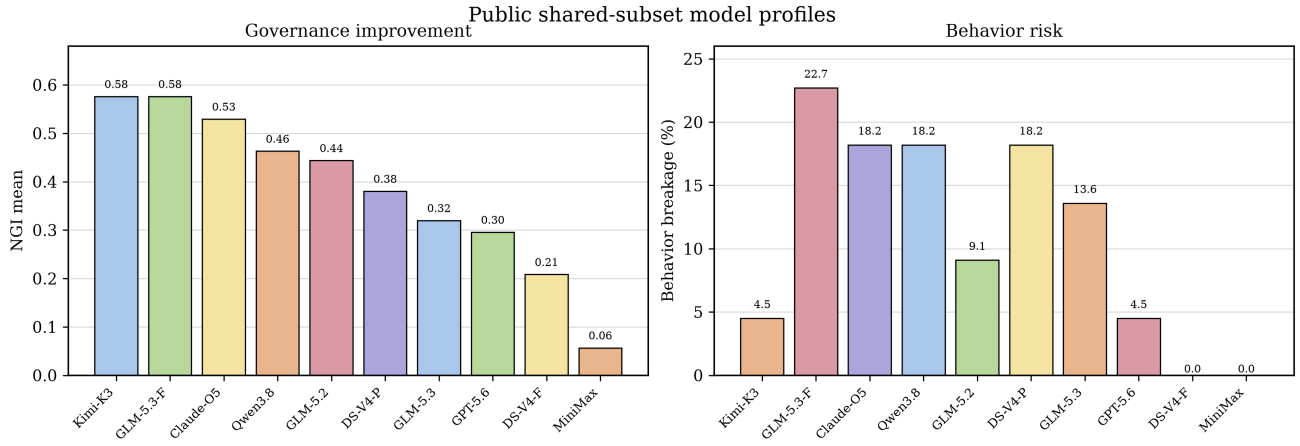
Figure 3a reports the headline outcome on the public shared subset. Mean NGI spans roughly an order of magnitude, from 0.0568 to 0.5760, while measured behavior breakage spans 0% to 22.7%. Kimi-K3 and GLM-5.3-Flash obtain the highest mean NGI, but they sit at opposite ends of the safety axis: Kimi-K3 breaks behavior on one of 22 instances while GLM-5.3-Flash breaks it on five. MiniMax-M3 produces little normalized improvement despite zero observed behavior breaks. The contrast is the reason improvement, safety, and absolute treated quality are reported as separate axes rather than folded into one leaderboard number. The companion coverage view separates valid runs from behavior-broken runs before any NGI aggregation, making clear that a high score supported by fewer valid runs should be interpreted cautiously.

5.4. Instance-level dispersion is large relative to the model spread

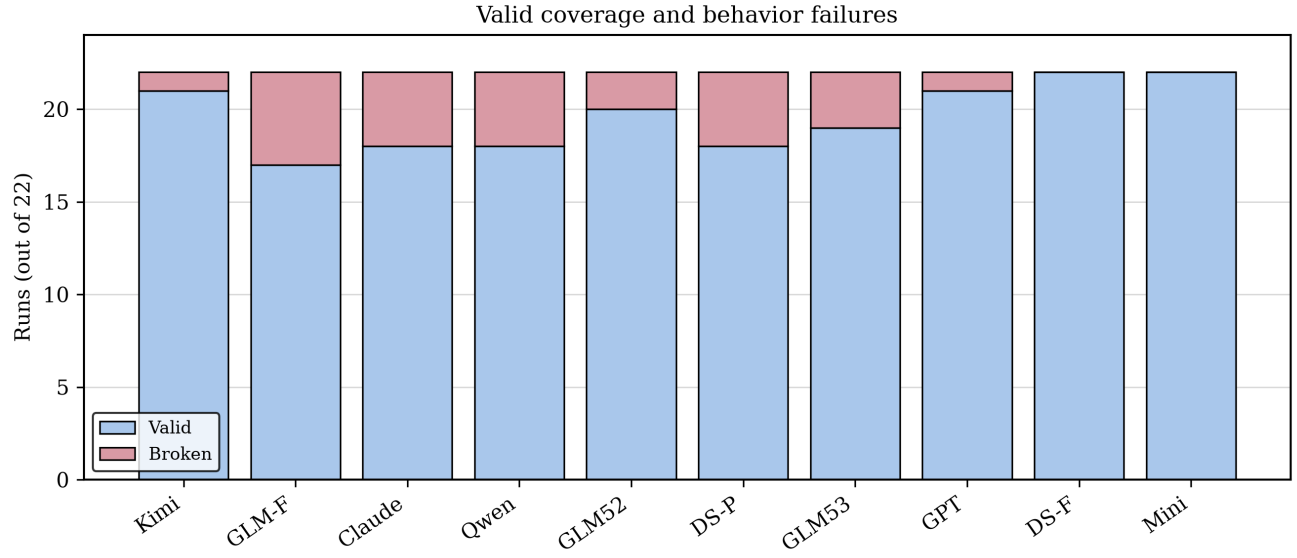
Aggregates hide how unevenly governance improvement is distributed. Figure 6a shows the per-instance NGI distribution behind each mean. Every model spans a wide interquartile range, several have valid instances at or below zero, and mean and median diverge substantially for models whose gains are concentrated in a few repositories—Claude-Opus-5 in particular has a median of 0.722 against a mean of 0.457 on the auxiliary batch, indicating a right-skewed profile driven by a subset of instances. Because each model-instance pair has been run once, this dispersion is a joint effect of repository heterogeneity, rollout variability, and scorer noise, and it is what limits the resolution of the current comparison (Section 5.9).

5.5. Diagnostic result analysis

This section reports the auxiliary conditions. Each subsection asks whether a specific part of the headline result survives a control.



(a) Public model profiles.



(b) Valid coverage and behavior failures.

Figure 3: Public outcomes and validity. Model-level NGI and behavior risk are shown together with the valid/broken decomposition of the same 22-instance pool. Claude-Opus-5 ran under Claude Code while the other nine models used pi.

5.6. Non-agent controls and the scoring noise floor

Figure 4a places the non-agent conditions inside the model ranking on the frozen ten-repository batch. The no-op condition has mean NGI -0.009 and median 0.000, with five of ten instances receiving a non-zero score change and a standard deviation of 0.073. The median of exactly zero is reassuring—the scorer does not systematically reward an unchanged repository—but the dispersion is real and provides a practical tolerance scale: observed differences of this size should be interpreted cautiously. Drift concentrates in Structure & Maintainability (three instances), Documentation and Reproducible Environment (two each); the environment cases are attributable to installation nondeterminism, while the structural cases should be deterministic on an unchanged diff and are the clearest symptom of judge variability in the current scorer.

The two template conditions sit inside the model range rather than below it. The mechanical baseline obtains mean NGI 0.272 and the rule-based baseline 0.254, both above DeepSeek-V4-Flash and MiniMax-M3. The ordering between the two baselines is itself informative: the deterministic rule baseline, which conditions its patch on

detected repository state, does *not* dominate a fixed template that ignores the repository entirely. Their separation, 0.018, is a quarter of the noise floor.

5.7. Matched comparison against the template baselines

Comparing means across conditions conflates repository difficulty with agent capability, because an agent system and a baseline may succeed on different repositories. We therefore compare each agent system to the stronger of the two templates *per instance*, counting an instance as a win only when the gap exceeds the empirical no-op noise floor. Figure 4b reports the resulting outcome switches.

The matched view is considerably less flattering than the mean ranking. Only Kimi-K3 (9 wins, 1 loss) and GLM-5.3-Flash (8–1) beat the template on a clear majority of repositories. Claude-Opus-5 and Qwen3.8-Max are close to even at 5–4 and 4–4. GPT-5.6-Sol is indistinguishable from the template on half of the batch, and the two weakest models are behind it: DeepSeek-V4-Flash loses on six of ten repositories and MiniMax-M3 on seven. In other words, most of the middle of the agent-system comparison is not reliably doing better, repository by repository, than a patch that never reads the repository.

5.8. Verifier quality control

A behavior gate is only as informative as it is discriminative, so we stratify outcomes by the mutation-testing label. Figure 5a shows the result over all 60 instances and ten models. Measured breakage falls monotonically as gate strength degrades—13% under detected gates, 8% under blind gates, 0% under vacuous gates—while median NGI moves by only 0.06 across the same strata. Weak gates therefore do not observe safer agents; they fail to observe unsafe ones. A zero observed breakage rate on a gate that cannot detect breakage is a measurement artifact, and any headline breakage number must be reported with its gate-strength breakdown.

The gate ablation itself is more nuanced than a simple inflation correction. Removing the gate and scoring every instance changes the median by at most 0.021 in either direction: it is positive for Kimi-K3 ($+0.014$), Qwen3.8-Max ($+0.021$), and GPT-5.6-Sol ($+0.021$), negative for GLM-5.3-Flash (-0.017), GLM-5.2 (-0.008), and GLM-5.3 (-0.007), and unchanged for the rest; the model ordering is identical with the gate on and off. Instances caught by the gate therefore tend to score near or below their model's average rather than above it, which suggests that behavior breakage is more often a symptom of a botched intervention than the price of an aggressive high-scoring one. The gate remains necessary—it identifies which scores are untrustworthy—but its effect should be reported as a sensitivity analysis, not assumed to be a monotone bias.

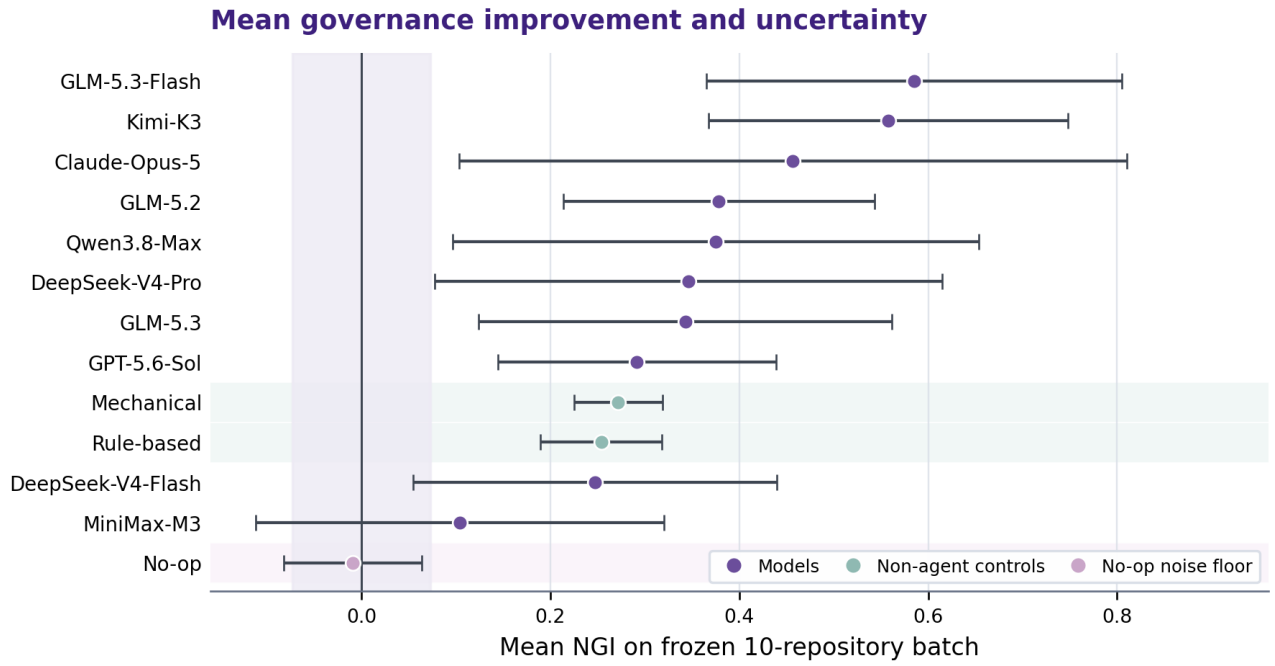
5.9. Sensitivity of the current design

Given the dispersion in Figure 6a, we ask what difference the current design can actually resolve. Across all 45 model pairs on their common-valid public instances, the paired instance-level NGI difference has median standard deviation 0.270 over a median of 18 shared instances. Figure 5b converts this into detection power. At 80% power and $\alpha = 0.05$, the public subset resolves a true mean difference of about 0.19, and the ten-repository auxiliary batch about 0.27.

This has a direct consequence for how Figure 3a should be read. Adjacent models in the ranking—Kimi-K3 and GLM-5.3-Flash, or GLM-5.2 and DeepSeek-V4-Pro—are separated by less than the resolution limit and are not distinguished by this instrument. The result supports a coarse grouping into a high band, a middle band, and a low band rather than a ten-way ordering. Reaching a resolution of 0.10 requires about 110 paired observations, or roughly five repeated rollouts per model-instance pair on the current public subset; this is the concrete cost of the repeated-seed experiment we defer to future work.

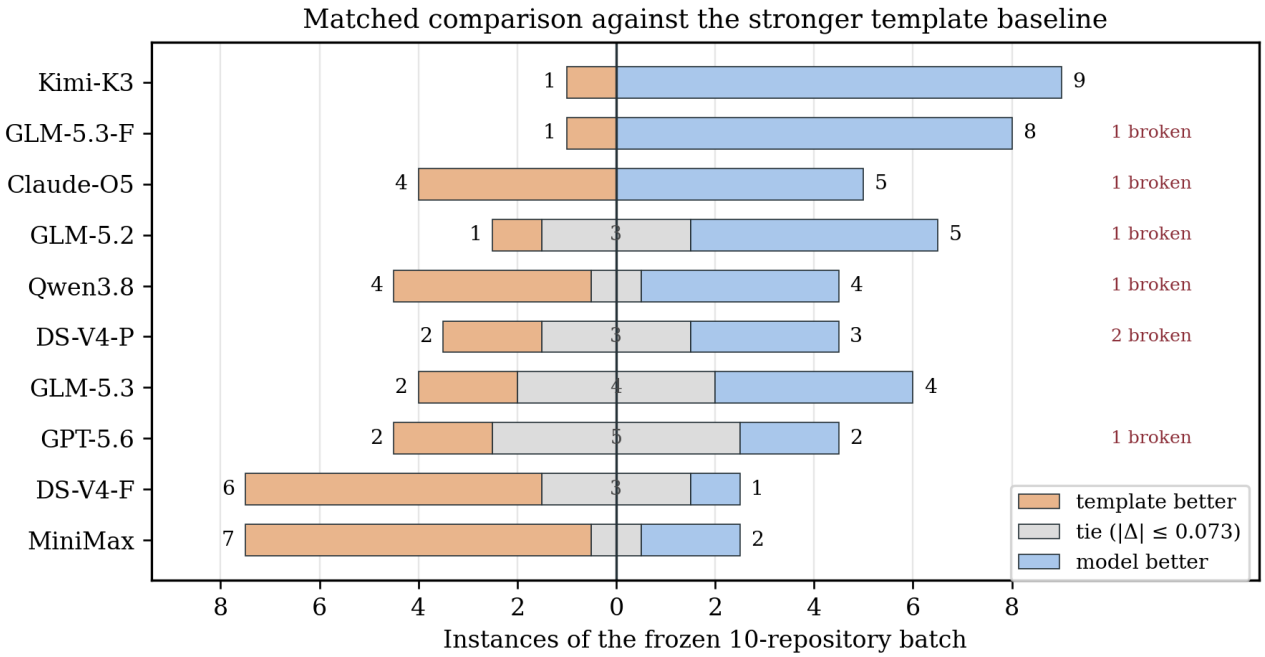
5.10. Per-dimension decomposition

Aggregate NGI can hide a condition that improves one dimension while neglecting another. Figure 6b decomposes improvement across the six governance dimensions. Two patterns are consistent across models. Tests & CI improves on the large majority of valid runs for every model except MiniMax-M3, whereas Structure & Maintainability is the weakest or joint-weakest dimension for eight of ten models, falling to 19% for GPT-5.6-Sol and 14% for



Points show means; whiskers show one standard deviation across repositories. The shaded band marks the No-op noise scale (±0.073).

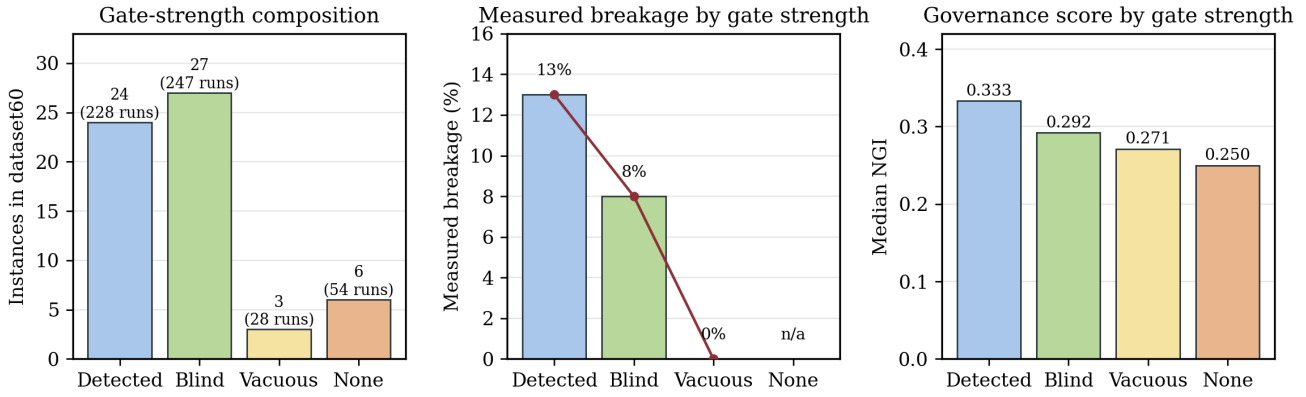
(a) Model gains against non-agent controls.



Per instance the model is compared against max(mechanical, rule-based); ties fall inside the no-op noise floor.

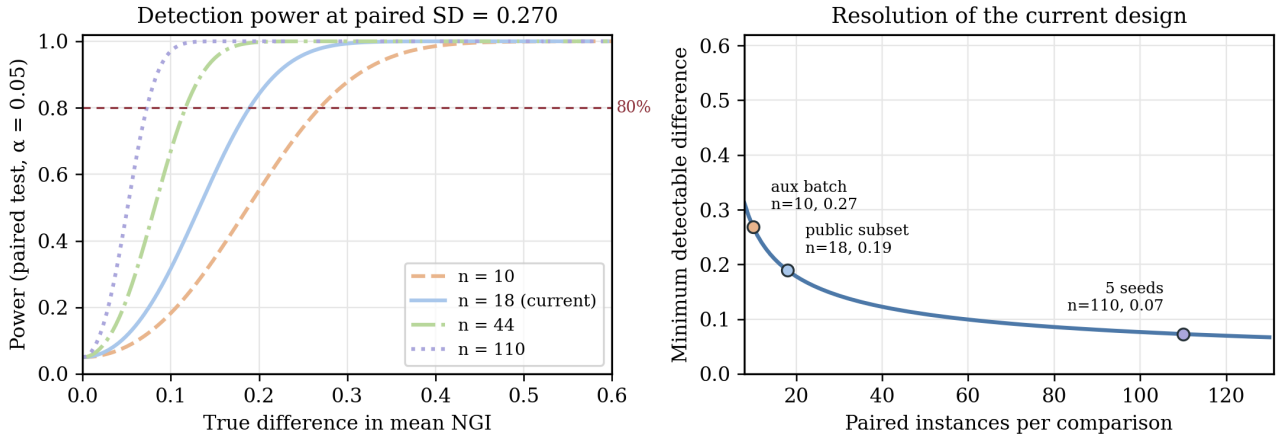
(b) Matched comparison against the stronger template baseline.

Figure 4: Auxiliary controls for interpreting model gains. Mean NGI, template baselines, the no-op noise floor, and matched per-instance outcomes show that aggregate improvement does not always imply repository-specific diagnosis.



Breakage falls monotonically as gates weaken while governance scores barely move: weak gates fail to observe unsafe patches rather than observing safer agents.

(a) Verifier quality control by mutation-testing gate strength.



Paired instance-level SD is estimated from all 45 model pairs on their common-valid public instances (median n = 18).

(b) Resolution of the current evaluation design.

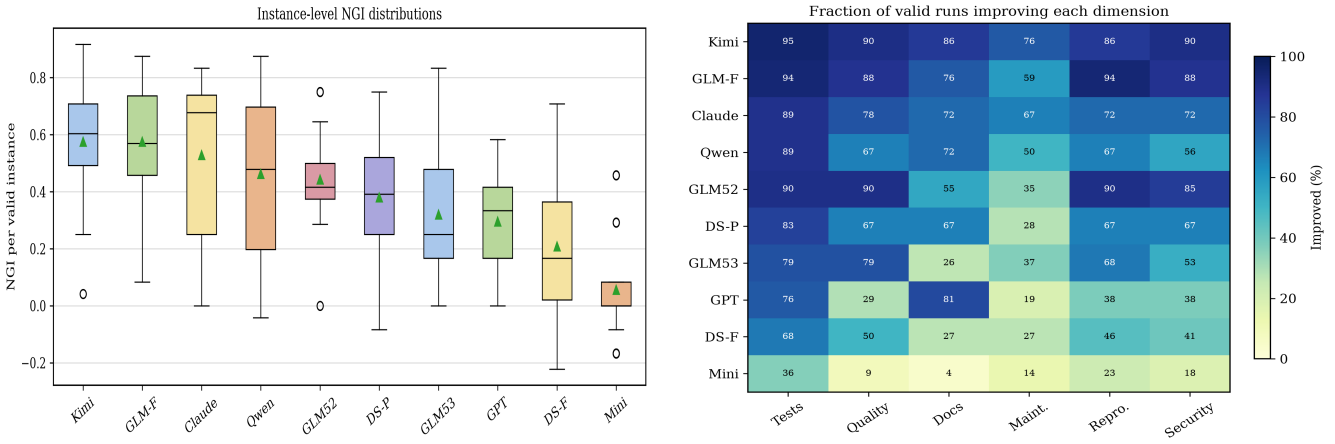
Figure 5: Verifier quality and statistical resolution. Weak gates observe fewer unsafe patches, while the current public subset has a finite resolution for detecting NGI differences.

MiniMax-M3. Only Kimi-K3 (76%) and Claude-Opus-5 (67%) reach it on a clear majority of their valid runs, and Claude-Opus-5 is notable for reaching it while scoring lower than several competitors on the configuration-facing dimensions—the profile of a model that edits structure rather than adding files.

The same decomposition applied to the template baselines is what explains their standing in Section 5.7. Both templates improve Tests & CI, Code Quality Gates, and Documentation & Collaboration on essentially every instance of the auxiliary batch, and improve Reproducible Environment and Dependency & Security Health on *none* of them, with Structure & Maintainability at 20%. The three dimensions a template can reach are exactly the three judged primarily from configuration and content; the three it cannot reach are those substantiated by container execution and static analysis of the repository itself. Adding a workflow file does not make a project install cleanly, and adding a security policy does not produce dependency evidence.

5.11. Key insight: governance improvement is not repository diagnosis

The controls converge on a single claim. Repository governance scores rise for reasons that have little to do with whether the agent understood the repository. A fixed template that never reads the project reaches mean NGI 0.272 and beats most models on a per-instance basis; a rule-based patch that does read the project adds 0.018 over that fixed template, a quarter of the noise floor; and the dimensions where models genuinely separate from both are precisely the ones the evaluator executes rather than inspects.



(a) Instance-level NGI distributions.

(b) Per-dimension improvement coverage.

Figure 6: Model heterogeneity and dimension coverage. The paired views show both the spread of repository-level gains and the governance dimensions improved by each model.

This is the governance analogue of a familiar benchmark failure: an evaluation can register the artifact of a capability without registering the capability. What distinguishes the top models here is not that they add more governance files, but that their changes survive a clean rebuild—that the project installs, audits, and remains structurally maintainable afterwards. Any future version of this benchmark should therefore extend execution-backed credit to the three currently template-reachable dimensions, and should report the template baselines alongside model scores as a standing lower bound.

6. Analysis

6.1. What the benchmark makes visible

The benchmark occupies a useful middle ground between issue resolution and static repository inspection. An issue benchmark asks whether a particular defect is fixed; a repository-health audit asks whether a project exhibits a property at one point in time. SWE-Prometheus asks an agent to diagnose a project-wide gap, intervene under an open-ended objective, and provide evidence that the intervention is useful and safe. This makes the task closer to maintenance work, but it also makes construct validity and evaluator transparency more important, because there is no oracle patch to fall back on.

The central distinction the evidence loop enforces is between engineering *presence* and engineering *effectiveness*. A repository can contain a test suite, CI workflow, lint configuration, or dependency file without obtaining the intended benefit. The loop tests whether the associated command runs, whether its scope is meaningful, and whether the treated repository remains behaviorally compatible. Section 5.10 shows this is not a theoretical concern: the three dimensions a repository-blind template can saturate are exactly the three judged without a clean rebuild.

The six-dimensional view also exposes asymmetric progress. Tests and environment files are often easy to add, whereas dependency security, maintainability, and collaboration practices require understanding project intent. Reporting per-dimension intervention rates rather than aggregate NGI alone is therefore not optional presentation detail; it is what prevents a template-shaped result from being read as a diagnosis-shaped one.

6.2. Improvement, safety, and absolute quality are separate axes

Higher NGI does not automatically imply lower risk. An agent that makes many changes may improve more dimensions and also create more opportunities for regressions. Conversely, a zero-breakage model may simply make few changes—MiniMax-M3 has no observed breaks and the lowest improvement in the release. The combination

of NGI, behavior-breakage rate, no-regression rate, and verified claim rate is therefore more informative than any single leaderboard number.

The gate analysis in Section 5.8 refines this. Removing the behavior gate raises the median for some models and lowers it for others rather than producing a universal inflation term, and the model ordering is unchanged. The gate's value is not that it corrects scores but that it marks which scores cannot be trusted, and gate strength determines how much confidence a zero-breakage observation can carry at all.

6.3. Implications for agent design

The task suggests three capabilities that are not captured by patch pass rates alone: repository reconnaissance, risk-aware intervention, and evidence-oriented verification. The matched comparison in Section 5.7 suggests that repository-specific diagnosis and prioritization remain weakly demonstrated by current agent systems. An agent that beats a fixed template only by adding the same artifacts the template adds has not demonstrated repository-specific diagnosis; it has demonstrated familiarity with a convention.

Agents should therefore construct a compact repository map first, prioritize the gaps whose repair requires the project to actually build and run, and avoid changing stable source behavior when configuration or documentation is sufficient. They should then verify both positive claims and negative side effects. This motivates future baselines with explicit diagnosis, planning, patching, and audit phases, and it suggests that diagnosis quality—not patch volume—is the capability the next iteration of this benchmark should isolate.

6.4. Validity threats and mitigations

Construct validity is threatened if the six dimensions become a checklist. We mitigate this with executable probes, mutation-based gate checks, and evidence requirements, and we quantify the residual exposure with template baselines rather than assuming it away. Internal validity is threatened by scaffold differences, one-shot sampling, and judge variability; Section 5.9 bounds the last of these by showing that the current design resolves differences of about 0.19, and we mitigate the others through paired reruns, fixed manifests, anonymized scoring, and calibration. External validity is limited by the current 60-repository sample and its language and ecosystem distribution. Releasing a larger, temporally refreshed, contamination-audited split is a direct next step.

The public table is a descriptive shared-subset result. Claude-Opus-5 was evaluated with Claude Code while the other models used pi, so the cross-scaffold ordering should not be read as a causal model comparison. Even among the pi runs, each model-instance pair has one rollout; repeated runs are required before discussing ranking stability or model-scale effects.

7. Limitations and Future Work

Only 24 of 60 instances currently have behavior gates whose discriminative power is demonstrated through mutation testing. Blind, vacuous, and none instances remain useful for governance-state analysis but provide weaker behavioral evidence, and Section 5.8 quantifies how much weaker: measured breakage falls from 13% to 8% to 0% as gate strength degrades with almost no change in governance scores.

The current scorer uses multiple teacher models but has not completed independent expert calibration. Subjective dimensions such as documentation and maintainability may contain shared judge errors; the no-op condition makes one symptom concrete, since Structure & Maintainability drifts on three of ten instances where an unchanged diff should be scored deterministically. Future evaluation should add blinded expert annotations, adjudication, cross-family judges, and adversarial calibration.

Three of the six dimensions are also demonstrably reachable by a repository-blind template (Section 5.10). This is a property of the current rubric rather than only of the models: Tests & CI, Code Quality Gates, and Documentation & Collaboration should be tightened so that credit requires demonstrated scope and execution, bringing them closer to how Reproducible Environment and Dependency & Security Health are already substantiated.

Finally, the single-rollout design bounds what any ranking claim can mean. At the observed paired dispersion the public subset resolves differences of about 0.19 NGI (Section 5.9), so adjacent positions in the current ordering are not separated by this instrument.

Different scaffolds expose different tools, prompts, context handling, and termination behavior. Current budget artifacts also include cases where monetary cost is unavailable; efficiency conclusions should therefore use tokens, turns, wall-clock time, and tool calls until cost provenance is complete.

The public subset is defined by common model coverage rather than random sampling. Public GitHub repositories may also have appeared in model training data or later repository versions. Stronger releases require repository-family separation, near-duplicate scanning, contamination audits, and an access-controlled private evaluator.

Finally, a public distribution must not include private tasks, private verification patches, private results, or other hidden evaluation artifacts. Every model-instance pair should have an artifact manifest so that missing traces or evidence are explicit rather than represented by empty directories. D6 measures available dependency and security evidence, not comprehensive vulnerability absence.

Future work will add repeated rollouts, expert baselines, stronger characterization tests, same-scaffold reruns, expert judge calibration, and a genuinely isolated private holdout. The template and deterministic baselines are now included as auxiliary measurements, but should be expanded across repository families before being treated as general lower bounds.

8. Conclusion

Repository-level agents are commonly evaluated as issue resolvers, while real teams also need them to improve the conditions under which a repository can be trusted. SWE-Prometheus makes this broader capability measurable by asking an agent to inspect a real repository, identify engineering risks, choose a retrofit strategy, and demonstrate that its changes work without breaking existing behavior.

The current SWE-Prometheus release shows that repository governance produces measurable differences across agent systems. The benchmark contains 60 real repositories, while ten agent systems are evaluated on the shared 22-repository public subset; it exposes variation in governance improvement, behavior-regression risk, and dimension-specific engineering preferences. It also shows why executable evidence is necessary: a change that looks correct in a diff may fail when executed or may damage existing behavior.

The completed auxiliary study adds the controls needed to read those differences. On a frozen ten-repository batch, the no-op noise floor has NGI standard deviation 0.073, and mechanical and rule-based retrofits obtain mean NGI of 0.272 and 0.254. A matched per-instance comparison shows that only two of ten models beat the stronger template on a clear majority of repositories, and that the two weakest models lose to it outright. Gate removal has mixed effects across models rather than a uniform inflation effect, while measured breakage declines monotonically as gate strength degrades—a property of the instrument, not of the agents. At the observed dispersion, the current public subset resolves differences of roughly 0.19 NGI.

Together these findings sharpen the design lesson. Governance scores can rise for reasons unrelated to whether the agent understood the repository, and the dimensions where models genuinely separate from a template are exactly those the evaluator executes rather than inspects. Benchmark conclusions in this setting therefore require explicit non-agent baselines, gate-strength labels, instance-level evidence, and a stated resolution limit.

SWE-Prometheus is therefore a resource for studying repository stewardship, complementing issue-level repair benchmarks with a setting that emphasizes diagnosis, prioritization, intervention, and verification. Its central question is not only whether an agent can change code, but whether it can leave a repository in a more trustworthy and maintainable state.

References

[1] 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.

[2] John Yang, Carlos E. Jimenez, Alexander Wettig, Konstantin Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *International Conference on Learning Representations*, 2024.

[3] OpenAI. Swe-bench verified. *OpenAI Research*, 2024. URL https://openai.com/index/swe-bench-verified/.

[4] Daoguang Zan, Zhirong Huang, Ailun Yu, Shaoxin Lin, Yifan Shi, Wei Liu, Dong Chen, Zongshuai Qi, Hao Yu, Lei Yu, Dezhi Ran, Muhan Zeng, Bo Shen, Pan Bian, Guangtai Liang, Bei Guan, Pengjie Huang, Tao Xie, Yongji Wang, and Qianxiang Wang. SWE-bench-java: A GitHub issue resolving benchmark for Java. *arXiv preprint arXiv:2408.14354*, 2024.

[5] Daoguang Zan, Zhirong Huang, Wei Liu, et al. Multi-SWE-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.

[6] Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. SWE-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*, 2025.

[7] John Yang, Carlos E. Jimenez, Alexander Wettig, Shunyu Yao, and Shun Narayanan. Swe-bench multimodal: Do ai systems generalize to visual software engineering tasks? In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[8] Linghao Zhang, Shilin He, Chaoyun Zhang, et al. Swe-bench goes live! *arXiv preprint arXiv:2505.23419*, 2025.

[9] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, et al. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems*, 2025.

[10] Zhipeng Xu, Jiahao Lu, Yining Zheng, Yuxin Wang, and Xipeng Qiu. SWE-bench science: Can coding agents resolve engineering tasks in science? *arXiv preprint arXiv:2608.19799*, 2026.

[11] Jiahong Xiang, Wenxiao He, Xihua Wang, Hongliang Tian, and Yuqun Zhang. Evaluating and improving automated repository-level rust issue resolution with llm-based agents. *arXiv preprint arXiv:2602.22764*, 2026.

[12] Xue Wang et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

[13] Chunqiu Steven Xia, Yihan Deng, et al. Agentless: Demystifying llm-based software engineering agents. In *International Conference on Learning Representations*, 2025.

[14] Tianyang Liu et al. Repobench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations Workshop*, 2023.

[15] Fang Zhang, Bei Chen, Yue Zhang, et al. Repocoder: Repository-level code completion through iterative retrieval and generation. In *Empirical Methods in Natural Language Processing*, 2023.

[16] Shaoqiu Zhang, Yuhang Wang, Jialiang Liang, Yuling Shi, Wenhao Zeng, Maoquan Wang, Shilin He, et al. SWE-explore: Benchmarking how coding agents explore repositories. *arXiv preprint arXiv:2606.07297*, 2026.

[17] Jiawei Liu et al. Repairagent: An autonomous, llm-based agent for program repair. In *International Conference on Software Engineering*, 2024.

[18] Shuai Wang et al. Repoagent: An llm-powered repository-level software development agent. In *International Conference on Software Engineering*, 2024.

[19] Niels Mundler, Mark Niklas Muller, Jingxuan He, and Martin Vechev. Swt-bench: Testing and validating real-world bug-fixes with code agents. *arXiv preprint arXiv:2406.12952*, 2024.

[20] Benjamin Miller et al. Repoexec: Evaluate code generation with repository-level executable benchmark. *arXiv preprint arXiv:2406.03095*, 2024.

[21] Jiawei Liu, Zhe Zhuo, et al. Evalplus: Rigorous evaluation of neural code generation. *arXiv preprint arXiv:2305.01210*, 2023.

[22] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[23] Giovanni Grano, Raffaele Ventura, and Harald C. Gall. On the prevalence of flaky tests in open source software. In *2018 IEEE International Conference on Software Maintenance and Evolution*, 2018.

[24] International Organization for Standardization. *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)*. ISO, 2011.

[25] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

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

[27] Zengyang Li, Paris Avgeriou, and Peng Liang. Technical debt in software systems: A systematic review. *Journal of Systems and Software*, 101:193–220, 2015.

[28] Jean-Louis Letouzey. The sqale method for managing technical debt. *Managing Technical Debt*, 2012.

[29] Bogdan Vasilescu, Yida Yu, Hua Wang, Prem Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[30] Moritz Beller, Georgios Gousios, and Andy Zaidman. The trivial matters: An empirical study of the impact of build and test failures on software development. In *International Conference on Software Engineering*, 2017.

[31] OpenSSF. Openssf scorecard. Technical specification, 2024.

[32] SLSA. Supply-chain levels for software artifacts specification, 2023.

[33] National Institute of Standards and Technology. Secure software development framework version 1.1, 2022.

[34] Rabe Abdalkareem et al. Not all dependencies are equal: An empirical study on production dependencies in npm. In *IEEE/ACM International Conference on Automated Software Engineering*, 2022.

[35] Konstantinos Vergopoulos, Mark Niklas M"uller, and Martin Vechev. Automated benchmark generation for repository-level coding tasks. *arXiv preprint arXiv:2503.07701*, 2025.

[36] Sayash Kapoor and Arvind Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 2023.

[37] Tianle Li et al. Livebench: A challenging, contamination-limited llm benchmark. In *International Conference on Learning Representations*, 2025.

[38] Fernando Martinez-Plumed et al. Position: Benchmarking machine learning systems is a sociotechnical problem. *Proceedings of Machine Learning Research*, 2021.

[39] Lianmin Zheng et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

[40] Yang Liu et al. G-eval: Nlg evaluation using gpt-4 with better human alignment. *Proceedings of EMNLP*, 2023.

[41] Yuntian Zhang et al. Swe-bench multimodal: Do ai systems generalize to visual software engineering tasks? *arXiv preprint arXiv:2410.03859*, 2024.

[42] Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

[43] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. In *ACM Computing Surveys*, 2018.

[44] Shunyu Yao et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[45] Timo Schick et al. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

[46] John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *International Conference on Learning Representations*, 2025.

[47] Linhao Zhang, Daoguang Zan, Quanshun Yang, Zhirong Huang, Dong Chen, Bo Shen, Tianyu Liu, Yongshun Gong, Pengjie Huang, Xudong Lu, Guangtai Liang, Lizhen Cui, and Qianxiang Wang. CodeV: Issue resolving with visual data. In *Findings of ACL*, pages 7350–7361, 2025.

[48] Shaoxiong Zhan, Shi Hu, Boyu Feng, Hai Lin, Andrew Gong, Zhengda Zhou, Jiaying Zhou, Yunyun Hou, Hao Su, and Hai-Tao Zheng. MM-IssueLoc: A controlled benchmark for evaluating visual evidence in multimodal repository-level issue localization. *arXiv preprint arXiv:2607.15205*, 2026.

[49] Dongfu Jiang, Xuan He, Huaye Zeng, Cong Wei, Max Ku, Qian Liu, and Wenhu Chen. MANTIS: Interleaved multi-image instruction tuning. *Transactions on Machine Learning Research*, 2024.

[50] Bingchen Zhao, Yongshuo Zong, Letian Zhang, and Timothy Hospedales. Benchmarking multi-image understanding in vision and language models. *arXiv preprint arXiv:2406.12742*, 2024.

[51] Fei Wang, Xingyu Fu, James Y. Huang, Zekun Li, Qin Liu, et al. MuirBench: A comprehensive benchmark for robust multi-image understanding. In *International Conference on Learning Representations*, 2025.

[52] Sergio Castro, Coen De Roover, Andy Kellens, Angela Lozano, Kim Mens, and Theo D'Hondt. Diagnosing and correcting design inconsistencies in source code with logical abduction. *Science of Computer Programming*, 76:1113–1129, 2011.

[53] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[54] Lianghong Guo, Wei Tao, Runhan Jiang, Yanlin Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, and Zibin Zheng. OmniGIRL: A multilingual and multimodal benchmark for GitHub issue resolution. *Proceedings of the ACM on Software Engineering*, 2025.

[55] Zhen Pan et al. Swe-gym: Training software engineering agents with real-world tasks. In *International Conference on Learning Representations*, 2025.

[56] Marco Tulio Ribeiro, Tong Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. *ACL*, 2020.

[57] Fan Cui, Hongyuan Hou, Zizhang Luo, Chenyun Yin, and Yun Liang. Hwe-bench: Benchmarking llm agents on real-world hardware bug repair tasks. *arXiv preprint arXiv:2604.14709*, 2026.

[58] Ion George Dinu, Marian Cristian Mihaescu, and Traian Rebedea. Smellbench: Evaluating llm agents on architectural code smell repair. *arXiv preprint arXiv:2605.07001*, 2026.

[59] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. In *arXiv preprint arXiv:2107.03374*, 2021.

[60] Jacob Austin, Augustus Odena, Maxwell Nye, et al. Program synthesis with large language models. In *arXiv preprint arXiv:2108.07732*, 2021.

[61] Xinyun Chen, Maxwell Lin, Nathanael Scharli, and Denny Zhou. Self-debugging: Teaching large language models to debug their own code. *arXiv preprint arXiv:2304.05128*, 2023.

[62] Noah Shinn, Federico Cassano, Ash Berman, et al. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[63] Zhou Zheng et al. Sweet: A framework for evaluating code agents with software engineering tasks. *arXiv preprint arXiv:2405.16203*, 2024.

[64] Qiguang Chen, Chengyu Luan, Jiajun Wu, Qiming Yu, Yi Yang, et al. Omibench: Benchmarking olympiad-level multi-image reasoning in large vision-language model. *arXiv preprint arXiv:2604.20806*, 2026.

[65] Wendong Bu, Yang Wu, Qifan Yu, et al. What limits virtual agent application? omnibench: A scalable multi-dimensional benchmark for essential virtual agent capabilities. In *International Conference on Machine Learning*, 2025.

A. Dataset Construction Details

The appendix is divided into a small number of top-level modules following the organization of benchmark papers such as OMIBench. Appendix A covers dataset construction and instance records; Appendix B gives the detailed main experiment and agent lifecycle; Appendix C defines evaluation and scoring; Appendix D reports detailed analysis and case studies; and Appendix E documents reproducibility, release, and the data dictionary. Auxiliary experiments and statistical templates are kept as subsections of the relevant modules rather than promoted to independent appendices.

A.1. Artifacts and accounting

Each instance is identified by repository, base commit, split, license, baseline scores, verification status, and gate-strength label. A complete model-instance record contains the unified patch, agent output, base evidence, treated evidence, verification result, budget record, and score record. Missing artifacts should be represented in a manifest with an explicit reason.

The evaluator distinguishes infrastructure failure, environment or dependency failure, missing or invalid patch, behavior regression, preserved behavior with no governance improvement, and preserved behavior with positive improvement. Behavior-broken runs are excluded from valid NGI aggregation but remain in failure statistics. Monetary cost is reported only when provider accounting is available; otherwise it is marked unavailable.

A.2. Rubric anchors

Scores of 1–2 indicate absent or non-functional governance, 3 indicates partial coverage with material gaps, 4 indicates an executable and reviewable standard, and 5 indicates broad coverage with evidence of maintainability and low regression risk.

Table 2: Compact rubric anchors used for each governance dimension.

Score	Interpretation
1	Absent, misleading, or actively broken practice
2	Minimal artifact with major gaps or unreliable execution
3	Partial coverage; useful in common cases but material gaps remain
4	Executable, reviewable, and appropriate for the repository
5	Broad, durable coverage with evidence of maintainability

A.3. Instance Records and Behavior Gates

A.3.1. Instance schema

Table 3 lists the minimum fields required for a reproducible instance. Fields are separated into task, evidence, and result records so that a missing trace cannot be mistaken for a zero score.

Table 3: Minimum instance schema for the release records.

Record	Required fields
Task	repository, base commit, split, language, license, applicability mask, task text
Environment	image or lockfile identifier, installation command, tool versions, timeout policy
Evidence	probe name, command, exit code, stdout/stderr digest, timestamp, applicability
Behavior	characterization tests, mutation sample, gate-strength label, pre/post status
Result	patch digest, changed files, score vector, NGI, breakage label, budget record
Provenance	evaluator version, prompt version, scaffold, model identifier, artifact completeness

A.3.2. Detailed gate-strength accounting

The current 60-instance release contains 24 instances with mutation-demonstrated gates, 27 with blind gates, 3 with vacuous gates, and 6 without a usable behavior gate. This distribution motivates reporting gate strength as a first-class variable. The reported results distinguish the full set, the mutation-demonstrated subset, and the no-gate subset separately.

Table 4: Behavior-gate categories and their interpretation.

Category	Count	Interpretation
Detected	24	Mutation is caught by the characterization gate
Blind	27	Mutation is not caught; risk evidence is weak
Vacuous	3	Gate executes but has insufficient discriminative power
None	6	No usable behavior gate is available

A.3.3. Reproducibility checklist

The artifact release includes: (1) a manifest of repository commits and licenses; (2) a container or environment lock for every public instance; (3) deterministic scorer configuration; (4) command-level logs and exit codes; (5) patch and trace completeness checks; (6) a separation policy for private tasks and verification evidence; (7) evaluator and judge version identifiers; (8) a script that reconstructs every main-paper table; and (9) a changelog for repository, rubric, or scoring updates.

A.3.4. Auxiliary experiment worksheet

For every auxiliary experiment, this appendix records the sampling frame, model and scaffold, paired comparison unit, and machine-readable result source. The ten-repository batch is frozen; repeated-seed, same-scaffold, and expert rows are reported separately from the completed measurements.

A.3.5. Expanded rubric prompts

For D1, evaluators should ask whether tests exercise important behavior and whether CI runs the relevant tests on a clean checkout. For D2, they should ask whether the quality command is meaningful, scoped, and enforced rather than merely configured. For D3, they should ask whether a new contributor can install, use, modify, and report security issues. For D4, they should ask whether changes reduce unnecessary coupling and preserve understandable boundaries. For D5, they should ask whether another user can reproduce installation and execution from the declared environment. For D6, they should ask whether dependencies are controlled and whether security checks produce inspectable evidence. In every dimension, a file's presence without successful execution is insufficient for a score of 4 or 5.

A.3.6. Auxiliary-results manifest

The following manifest records which auxiliary values are already incorporated and which analyses remain open.

The measured values are summarized in Figure 4a; rows outside the primary comparison are clearly identified and are not cited as empirical evidence. The archived result package contains the exact ten-instance selection and run metadata.

A.4. Candidate Dataset and Quality Control

This appendix expands the construction description by separating the benchmark object, the construction funnel, the evaluation protocol, and the quality-control evidence. It is explicit about what is fixed, what is hidden, and how quality control is applied.

A.4.1. Dataset construction overview

The current SWE-Prometheus release was constructed as a repository-governance benchmark rather than sampled from a synthetic checklist. The construction unit is a repository at a fixed commit. The final package contains

Table 5: Status of auxiliary-result fields.

Artifact	Available content	Reported measure
10-repository manifest	Frozen stratified batch	Recorded in machine-readable manifest
Main rerun	Existing model rollouts on the batch	Observed per-instance scores
No-op baseline	Measured scorer stability	NGI mean, median, and SD
Mechanical baseline	Measured template intervention	Patch and evidence results
Rule baseline	Measured deterministic intervention	Rule-based results
Gate ablation	Measured on all 60 instances	Gated/ungated paired statistics
Repeated seeds	Not included in the primary comparison	Reserved for follow-up evaluation
Expert treatment	Not included in the primary comparison	Reserved for follow-up evaluation

60 instances, divided into 22 public instances with shared model coverage and 38 private instances reserved for internal evaluation. The public/private distinction is applied after task validation and does not change the scoring definition.

The construction workflow is:

1. discover repositories with plausible engineering-health gaps;
2. freeze a reconstructable base commit and collect project metadata;
3. run baseline probes and assign the six-dimension applicability mask;
4. construct or validate a behavior characterization gate;
5. execute a dry-run of installation, probes, and scoring;
6. record artifacts, provenance, and exclusion reasons;
7. assign the instance to the public or private split and freeze the manifest.

The workflow intentionally separates discovery from inclusion. A repository can be useful for finding candidates because it lacks a workflow or has sparse tests, but it enters the final dataset only after its behavior, environment, and governance headroom are independently checked.

Table 6: Dataset-construction stages and concrete outputs.

Stage	Operation	Output
Discovery	Search active open-source repositories for governance signals	Candidate repository list
Pinning	Select a reachable pre-treatment commit	Repository and commit manifest
Baseline	Run installation, tests, quality, dependency, and security probes	Base evidence and score vector
Applicability	Decide which of D1–D6 are meaningful	Applicability mask and rationale
Behavior	Validate characterization tests and mutation sensitivity	Verification patch and gate-strength label
Dry run	Rebuild the environment and execute the full evaluator	Environment and evaluator log
Packaging	Hash artifacts and remove protected material from public outputs	Instance package and release manifest

A.4.2. Construction principles

SWE-Prometheus treats a repository as a maintained artifact rather than a collection of isolated code snippets. The benchmark therefore follows four principles. First, each task starts from a reconstructable repository snapshot.

Second, the objective is open-ended enough to admit multiple valid engineering interventions. Third, each claimed improvement is connected to a probe or a structured judgment. Fourth, behavior preservation is tested independently from governance improvement.

These principles distinguish three artifacts that can otherwise be conflated: the task defines the engineering objective, the agent patch represents the intervention, and the evaluator evidence determines whether the intervention is useful and safe. The developer or construction patch, when present, is a construction asset and is not exposed as an oracle to the agent.

A.4.3. Candidate discovery funnel

Candidate discovery proceeds in four stages: broad repository retrieval, automated mechanical screening, manual validity inspection, and evaluator dry runs. Broad retrieval favors recall and may use repository metadata, visible workflow gaps, test sparsity, packaging inconsistencies, documentation gaps, and dependency hygiene signals. Mechanical screening removes repositories that cannot be cloned or pinned. Manual inspection verifies that the project has a meaningful functional core and that its governance gaps are not artifacts of a broken checkout.

The final dry run reconstructs the base snapshot, executes all applicable probes, and records unavailable evidence. A candidate is rejected when the environment cannot be made reproducible, when the task requires private services, when no dimension has plausible headroom, or when behavior cannot be characterized at all. This funnel prevents the benchmark from equating "many missing files" with "high-value engineering work."

A.4.4. Baseline assessment and headroom

For every retained candidate, the baseline evaluator runs before any model treatment. The baseline record contains the raw probe outputs, the normalized score vector, the set of applicable dimensions, and the reason for every score. Headroom is computed relative to the dimension ceiling, but high headroom alone is not sufficient for inclusion: a repository with a completely non-functional environment is not a meaningful governance task.

The baseline pass also identifies which commands are safe to run without network access and which dependencies are merely preinstalled in the evaluator image. This distinction is particularly important for D5 and D6. Declared project dependencies are audited as project artifacts; evaluator convenience tools are not counted as evidence that the repository itself is reproducible or secure.

A.4.5. Behavior-gate construction

Behavior gates are constructed from repository-native tests whenever possible. If the native suite is too broad, a small characterization patch targets stable public behavior and records the command used to execute it. The patch is validated on the base snapshot before any model treatment. We then assess gate strength by injecting behavior-preserving or behavior-changing mutations where the repository and language permit it. A gate is labeled detected only when the mutation is caught; a passing but mutation-blind test remains blind rather than being promoted to strong evidence.

This construction step is kept separate from governance scoring. The gate protects against destructive interventions, while the six-dimensional scorer measures engineering readiness. A strong gate therefore increases confidence in a treatment outcome but does not directly increase the treatment score.

A.4.6. Split construction and leakage control

The public split contains the instances used for shared comparison and trace inspection. The private split retains tasks, verification patches, and outcomes that should not be available for targeted optimization. Split assignment is recorded in the manifest, and private artifacts are excluded from public result aggregation. A broader public split should additionally use repository-family separation, near-duplicate scanning, and temporal contamination checks.

A.4.7. Instance packaging

An instance package contains task metadata, a repository identity, a base commit, a dimension applicability mask, base evidence, a verification asset, and evaluator configuration. Public packages may include task descriptions and public traces, but private verification patches, hidden test details, and private model outcomes remain excluded. Every package receives a stable identifier and a manifest digest.

Table 7: Candidate screening checklist used before an instance enters the benchmark.

Gate	Pass condition	Recorded evidence
Identity	Repository and base commit are uniquely identified	URL, commit hash, license
Buildability	Snapshot can be installed or inspected with declared limits	Environment log, failure reason
Functionality	Project contains meaningful runnable or testable behavior	Entry points and test sample
Applicability	At least two governance dimensions are relevant	Dimension mask
Headroom	Baseline is below the dimension ceiling without being pathological	Baseline score vector
Behavior	A characterization test or gate can be constructed	Patch, command, gate strength
Isolation	No private credential or construction artifact is required	Network and secret policy

Packaging is performed before model assignment. The task text does not disclose a target score, a preferred file, or a reference implementation. If a repository contains generated files or large assets, the package records whether they are retained, omitted, or reconstructed. Any omission that can affect execution is treated as a validity concern rather than an undocumented convenience.

A.4.8. Quality-control passes

We define three independent quality-control passes. The first checks package integrity: identifiers, commit reachability, file presence, manifest hashes, and license metadata. The second checks evaluator integrity: base evidence is reproducible, the characterization test runs on the base state, and the treated-state harness accepts a known valid patch. The third checks interpretation: the score record includes evidence, the behavior result is consistent with the logs, and missing data are labeled explicitly.

The passes are deliberately separated because a package can be technically executable but conceptually invalid, or conceptually sound but impossible to reproduce. The release checklist records pass, fail, or not-applicable for every pass and preserves the reason for every exception.

B. Detailed Main Experiment

B.1. Task tuple

We represent an instance as

$$T_i = (R_i, c_i, M_i, A_i, E_i, V_i),$$

where R_i is the repository, c_i the pinned base commit, M_i the task metadata and applicability mask, A_i the agent-facing instruction, E_i the executable environment, and V_i the verification package. The agent receives $(R_i, c_i, M_i, A_i, E_i)$ but not protected parts of V_i. A rollout produces a patch $P_{i,m}$ and an execution trace $\tau_{i,m}$ for model m.

The evaluator computes base evidence B_i and treated evidence $B_i(P_{i,m})$. For each dimension d, the scorer returns a score in $\{1,2,3,4,5\}$ or an explicit inapplicable value. The result record is therefore a tuple of evidence, scores, behavior status, and budget rather than only a scalar.

B.2. Evidence record

Every probe should be represented by the following fields: probe identifier, command, working directory, environment identifier, start and end time, exit code, timeout status, output digest, normalized result, applicability, and human-readable interpretation. The raw log may be stored separately, but the normalized record must be sufficient to reconstruct why a score was assigned.

Table 8: Evidence fields and their role in auditability.

Field	Purpose
Probe ID	Stable name for the check across evaluator versions
Command	Exact executable command, including relevant flags
Environment	Runtime, image, dependency, and tool version identifiers
Exit status	Distinguishes pass, failure, timeout, and unavailable execution
Output digest	Links the normalized record to the retained log
Applicability	Indicates whether the check is meaningful for this repository
Interpretation	Explains the mapping from evidence to rubric score

B.3. Applicability and missingness

An inapplicable dimension is not equivalent to a failed dimension. For example, a repository without a releasable package may not receive a packaging score, while a repository that declares a package but cannot install it receives negative evidence. The scorer therefore records applicability separately from outcome and prevents missing evidence from being converted into an unexplained zero.

Missingness is categorized as not applicable, unavailable due to environment, unavailable due to timeout, or malformed artifact. This coding supports sensitivity analyses. Complete-case summaries retain unavailable evidence as a separate outcome rather than converting it to a zero.

B.4. Normalization and aggregation

Raw dimension scores are retained alongside normalized improvement. For an instance with applicable dimensions D_i, the headroom-normalized score is computed only over defined dimensions. We report the number of dimensions contributing to each aggregate, the raw score sum, the treated score sum, and the normalized value. This avoids over-interpreting an identical NGI computed from very different numbers of applicable dimensions.

B.5. Agent Protocol and Run Lifecycle

B.5.1. Agent-facing instruction template

The following template describes the stable task-level contract. The actual task-specific metadata and repository path are filled by the runner. It does not prescribe which files to edit or which governance intervention to choose.

> Inspect the provided repository and improve its engineering readiness across the applicable governance dimensions. Preserve existing behavior and public interfaces. Make useful changes for future users and maintainers. Use repository evidence to prioritize changes, verify important claims with executable checks, and submit the resulting patch. Do not use hidden evaluator assets or any reference patch.

The instruction is intentionally open-ended. The agent must discover the repository's build system, tests, documentation conventions, and dependency structure rather than receiving a checklist of missing files. The runner separately provides tool descriptions, time limits, and submission semantics.

B.5.2. Run lifecycle

Each run follows the lifecycle below:

1. reconstruct the pinned base snapshot;
2. verify base environment and characterization behavior;
3. launch the model with the fixed task instruction;
4. archive the model output and candidate patch;

5. apply the patch to an independent clean snapshot;
6. run treated-state probes and behavior verification;
7. score the evidence and write the result manifest.

The agent-side container is discarded after submission. The evaluator does not reuse the agent's mutable working tree because doing so could allow unrecorded caches or generated files to influence the result.

B.5.3. Canonical-run policy

Retries are allowed for infrastructure failures such as a failed container launch, a transient model service error, or a verifier crash. A patch that is valid but incorrect is not retried merely to obtain a better result. When a retry occurs, the original record remains archived and the manifest identifies the canonical run with an explicit reason.

B.5.4. Budget accounting

Budget records contain model identifier, scaffold, maximum wall-clock time, actual elapsed time, input and output token counts when available, number of turns, tool calls, termination reason, and provider cost when available. A missing cost field is represented as unavailable. We do not compare providers using estimated prices from incomplete ledgers.

Table 9: Run-level budget fields for later auxiliary experiments.

Field	Definition
Model/scaffold	Exact model and execution framework
Wall time	Start-to-stop elapsed time and configured timeout
Turns/tools	Number of agent turns and external tool calls
Tokens	Input, output, and total tokens when exposed
Termination	Completed, timeout, service failure, invalid patch, or other reason
Cost	Provider-reported cost; unavailable otherwise

C. Evaluation and Scoring

C.1. Dimension-specific score anchors

The general 1–5 anchors are expanded below to reduce holistic scoring. A score of 4 requires executable or inspectable evidence appropriate to the repository; a configuration file by itself is not sufficient.

C.2. Evidence sufficiency rules

For D1, a passing test command must be paired with evidence that the tests exercise meaningful behavior. For D2, the existence of a formatter or linter configuration does not establish that it runs on the project's relevant files. For D3, prose is evaluated for actionability rather than length. For D4, a large refactor is not rewarded if it increases churn without improving boundaries. For D5, a lockfile is useful only when it agrees with the declared project metadata. For D6, a dependency report must distinguish project dependencies from tools preinstalled in the evaluator image.

C.3. No-regression rules

A treated repository receives a regression flag when characterization behavior fails, an important public interface changes unexpectedly, or the patch causes the evaluator to lose the ability to run applicable probes. A governance improvement cannot compensate for a confirmed behavior break in the primary valid-run aggregate. This asymmetric rule reflects the practical cost of damaging a working project while adding maintenance metadata.

Table 10: Dimension-specific interpretation of an acceptable score.

Dim.	Score 3	Score 4–5
D1	Tests cover selected behavior but important paths or CI execution are missing	Important behavior is exercised in a clean, repeatable test/CI path
D2	Tool configuration exists but scope or enforcement is incomplete	Quality checks run, target meaningful files, and expose failures
D3	Basic usage or contribution information exists with material omissions	A new maintainer can use, change, and report issues responsibly
D4	Organization is understandable locally but coupling or duplication remains	Boundaries and dependency direction are maintainable and justified
D5	Installation works only with undocumented assumptions	Clean setup and execution are reproducible from declared metadata
D6	Dependencies are declared but audit or update evidence is incomplete	Dependency and security posture are controlled and inspectable

C.4. Judge calibration worksheet

Before the final scoring pass, evaluators should independently score a calibration set spanning low, medium, and high baseline quality. The worksheet records score, evidence cited, uncertainty, and whether the evaluator would request a maintainer review. Agreement is computed before adjudication; adjudication changes are retained as a separate field.

C.5. Auxiliary Experiment Design

C.5.1. Fixed 10-repository batch

The completed auxiliary conditions use one frozen batch of ten repositories. The manifest contains repository, commit, language, size band, base mean, applicable dimensions, gate strength, and public/private status. No experiment may silently substitute an easier repository after execution begins.

Table 11: Required fields for the frozen auxiliary batch manifest.

Field	Requirement
Instance ID	Stable repository-level identifier
Commit	Exact base revision used by every condition
Strata	Size, baseline-quality, and gate-strength bins
Applicability	D1–D6 mask and reason for exclusions
Environment	Image or lockfile and evaluator version
Selection rationale	Why the instance represents the intended stratum

C.5.2. Condition matrix

The completed condition matrix contains no-op, model rollouts, mechanical retrofit, and rule-based baseline on the same ten repositories. Behavior-gate ablation uses all 60 instances because the public subset has no “none” gate cases. Optional repeated-seed, same-scaffold, and expert rows are outside the primary comparison.

C.5.3. Pre-registered hypotheses

H1: the model treatment achieves higher NGI than the two non-semantic baselines but exhibits larger variance. H2: the behavior gate reduces apparent success by removing unsafe patches. H3: detected gates yield higher no-regression confidence than blind or vacuous gates. H4: improvement is concentrated in D1–D3, while D4–D6 require more repository-specific reasoning. H5: broad source edits correlate with behavior risk more strongly than targeted governance edits.

Table 12: Unified condition matrix for the next experimental package.

Condition	Model	Repositories	Main comparison
No-op	None	10	Scoring and environment stability
Main	Kimi-K3/pi	10	Open-ended repository governance
Mechanical	None	10	Template-only intervention
Rule	Deterministic	10	Checklist-oriented automation
Gate ablation	Re-score	10	Effect of behavior filtering
Repeated seed	Kimi-K3/pi	5 or 10	Within-model variance

These hypotheses are directional and should be evaluated with paired instance-level records. If an estimate is not supported, the final paper will report the negative result rather than modify the hypothesis after seeing the data.

C.6. Statistical Analysis and Reporting Templates

C.6.1. Primary estimands

The primary estimand is the paired difference in NGI between the Kimi-K3 treatment and each baseline on the same ten repositories. Secondary estimands are the paired difference in raw dimension score, behavior-breakage rate, no-regression rate, verified-claim rate, and resource usage. We report per-instance values before any mean or median aggregation.

C.6.2. Mean–median reporting rationale

The main result table reports mean NGI as its single aggregate improvement measure. This choice keeps the leaderboard compact and makes the quantity directly interpretable as average normalized governance improvement across valid runs. Because the current public comparison contains 22 instances and one rollout per model-instance pair, the mean is still descriptive rather than an estimate of a stable population-level capability.

Median NGI is retained as a secondary diagnostic in the appendix rather than the main table. It is useful for identifying whether a model's mean is driven by a small number of unusually large improvements: a mean substantially above the median indicates a right-skewed outcome distribution, while a close mean–median pair suggests more uniform instance-level behavior. The median is not used to replace the mean because it does not reflect the magnitude of improvements above or below the typical instance and can hide the contribution of a few high-value but valid interventions.

We therefore interpret the two statistics jointly when performing detailed analysis, but use only mean NGI in the compact main table. Any future claim about ranking stability will additionally report per-instance values, bootstrap intervals, valid coverage, and breakage rate. In particular, a high median cannot compensate for a high behavior-breakage rate, and a high mean cannot be interpreted as robust if it is supported by only a small valid denominator.

Model-level outcome profiles are reported in the body as Figure effig:model-profiles.

Instance-level NGI distributions are reported in the body as Figure effig:ngi-distribution.

C.6.3. Uncertainty reporting

For ten repositories, asymptotic significance tests are not persuasive. We therefore emphasize exact paired differences and bootstrap intervals over repository units, while clearly identifying the exploratory nature of the small batch. Repeated seeds, when available, add a second source of uncertainty and should be summarized with a hierarchical table separating between-repository and within-model variation.

C.6.4. Per-dimension reporting

For each condition, the final appendix should include a six-column D1–D6 table with base mean, treated mean, mean raw change, fraction improved, fraction unchanged, and fraction regressed. The table should also report the

Table 13: Compact auxiliary-result summary.

Condition	Valid	Mean	Median	SD	Broken	Notes
No-op	10	-0.009	0.000	0.073	0	scorer stability
Kimi-K3	10	0.558	0.615	0.190	0	primary treatment
Mechanical	10	0.272	0.275	0.047	0	template baseline
Rule	10	0.254	0.283	0.064	0	deterministic baseline
Gate-off	all 60	mixed	mixed	n/a	n/a	paired ablation

number of applicable instances per dimension. This prevents aggregate NGI from hiding a condition that improves one dimension while damaging another.

The per-dimension improvement heatmap is reported in the body as Figure effig:dimension-heatmap.

C.6.5. Missing and excluded runs

The analysis script must print a list of excluded runs with an exclusion code before generating any aggregate table. Codes include infrastructure, invalid patch, behavior broken, missing evidence, and inapplicable dimension. The paper should report both the denominator before exclusion and the denominator used for valid NGI.

D. Full Result Tables

The body reports every result as a figure. This appendix gives the underlying numbers.

D.1. Public shared-subset results

Table 14: Public shared-subset results (underlying Figure 3a). Valid and broken counts are out of 22 instances. Valid runs preserve characterization behavior and contain scorable evidence; NGI is aggregated only over valid runs. Breakage rate is broken/(valid+broken). Claude used Claude Code; the other models used pi.

Model	Scaffold	Valid	Broken	Breakage	NGI mean	NGI median
GPT-5.6-Sol	pi	21	1	4.5%	0.2956	0.3333
Claude-Opus-5	Claude Code	18	4	18.2%	0.5293	0.6771
Kimi-K3	pi	21	1	4.5%	0.5760	0.6042
GLM-5.3-Flash	pi	17	5	22.7%	0.5760	0.5694
Qwen3.8-Max	pi	18	4	18.2%	0.4630	0.4792
GLM-5.2	pi	20	2	9.1%	0.4438	0.4167
DeepSeek-V4-Pro	pi	18	4	18.2%	0.3803	0.3917
GLM-5.3	pi	19	3	13.6%	0.3198	0.2500
DeepSeek-V4-Flash	pi	22	0	0.0%	0.2083	0.1667
MiniMax-M3	pi	22	0	0.0%	0.0568	0.0000

D.2. Frozen ten-repository auxiliary batch

D.3. Behavior-gate ablation and gate-strength stratification

D.4. No-op drift and per-instance baseline comparison

Table 15: Observed results on the frozen ten-repository auxiliary batch (underlying Figure 4a). Models and non-agent conditions are ranked jointly by mean NGI. NGI statistics exclude behavior-broken runs; "strict" counts valid runs passing the strict-success criterion. Rows separated by less than the no-op standard deviation are not distinguished by this instrument.

Condition	Valid	Mean	Median	SD	Broken	Strict
GLM-5.3-Flash	9	0.585	0.583	0.220	1	0
Kimi-K3	10	0.558	0.615	0.190	0	2
Claude-Opus-5	9	0.457	0.722	0.354	1	0
GLM-5.2	9	0.378	0.390	0.165	1	0
Qwen3.8-Max	9	0.375	0.375	0.278	1	0
DeepSeek-V4-Pro	8	0.346	0.281	0.269	2	1
GLM-5.3	10	0.343	0.271	0.219	0	0
GPT-5.6-Sol	9	0.292	0.333	0.147	1	0
Mechanical (template)	10	0.272	0.275	0.047	0	0
Rule-based (deterministic)	10	0.254	0.283	0.064	0	0
DeepSeek-V4-Flash	10	0.247	0.174	0.193	0	0
MiniMax-M3	10	0.104	0.042	0.216	0	0
No-op (noise floor)	10	−0.009	0.000	**0.073**	0	0

Table 16: Matched per-instance comparison against the stronger template baseline (underlying Figure 4b). Each agent system is compared against $\max(\text{mechanical}, \text{rule-based})$ on the same repository; ties are defined heuristically as differences within the empirical no-op noise floor of 0.073, not as formal equivalence tests. Behavior-broken instances are excluded from the win/tie/loss counts.

Model	Wins	Ties	Losses	Broken	Mean Δ	SD Δ
Kimi-K3	9	0	1	0	+0.265	0.195
GLM-5.3-Flash	8	0	1	1	+0.298	0.221
Claude-Opus-5	5	0	4	1	+0.170	0.360
GLM-5.2	5	3	1	1	+0.092	0.181
Qwen3.8-Max	4	1	4	1	+0.089	0.268
GLM-5.3	4	4	2	0	+0.051	0.221
DeepSeek-V4-Pro	3	3	2	2	+0.066	0.283
GPT-5.6-Sol	2	5	2	1	+0.004	0.137
DeepSeek-V4-Flash	1	3	6	0	−0.045	0.213
MiniMax-M3	2	1	7	0	−0.188	0.231

D.5. Baseline patch contents

The mechanical condition writes seven fixed artifacts regardless of repository content: a GitHub Actions CI workflow, a `.pre-commit-config.yaml`, a `mypy.ini`, `CONTRIBUTING.md`, `SECURITY.md`, an issue-template directory, and a smoke test containing `assert True`. The rule-based condition first probes which of these are absent, adds only those, and replaces the vacuous smoke test with `importlib.import_module` against the package name inferred from the repository layout. Neither condition invokes a language model and neither modifies existing source files, so both are guaranteed not to break behavior—which the gate confirms, with zero broken runs out of ten for each.

D.6. Additional analyses

Beyond model averages, useful stratifications include repository language, project size, baseline score, number of applicable dimensions, test-suite maturity, and dependency-manager type. These analyses distinguish absolute

Table 17: Per-dimension improvement rate on the frozen ten-repository batch. Fraction of scored instances on which each dimension improved. The template baselines saturate the three dimensions judged from configuration and content, and score zero on the two substantiated by container execution.

Condition	D1 Tests/CI	D2 Quality	D3 Docs	D4 Structure	D5 Repro.	D6 Dep./Sec.
Kimi-K3	100%	100%	80%	60%	90%	100%
GLM-5.3-Flash	100%	77%	88%	55%	88%	88%
Claude-Opus-5	88%	55%	55%	**77%**	55%	55%
GLM-5.2	88%	77%	44%	33%	88%	77%
GLM-5.3	90%	70%	20%	40%	70%	60%
DeepSeek-V4-Pro	87%	62%	50%	25%	62%	62%
DeepSeek-V4-Flash	90%	60%	30%	30%	60%	40%
Qwen3.8-Max	100%	33%	55%	55%	55%	33%
GPT-5.6-Sol	77%	33%	77%	11%	33%	44%
MiniMax-M3	40%	10%	10%	10%	40%	30%
Mechanical (template)	100%	100%	90%	20%	**0%**	**0%**
Rule-based (deterministic)	100%	100%	100%	20%	**0%**	**0%**
No-op (noise floor)	0%	0%	10%	20%	10%	0%

Table 18: Behavior-gate ablation over all instances available per model. "Gate off" counts every scored instance; "gate on" is the current protocol. The model ordering is identical under both settings, and the inflation term is small and signed in both directions.

Model	Scored	Valid	NGI gate off	NGI gate on	Inflation	Breakage
Claude-Opus-5	22	18	+0.677	+0.677	+0.000	18%
Kimi-K3	60	56	+0.517	+0.531	−0.014	7%
GLM-5.3-Flash	58	52	+0.476	+0.458	+0.017	12%
Qwen3.8-Max	59	52	+0.375	+0.396	−0.021	13%
GLM-5.2	59	53	+0.375	+0.367	+0.008	11%
DeepSeek-V4-Pro	59	51	+0.367	+0.367	+0.000	15%
GPT-5.6-Sol	60	57	+0.229	+0.250	−0.021	6%
GLM-5.3	60	54	+0.215	+0.208	+0.007	11%
DeepSeek-V4-Flash	60	57	+0.125	+0.125	+0.000	6%
MiniMax-M3	60	57	+0.000	+0.000	+0.000	6%

Table 19: Gate-strength stratification over all 60 instances and ten models (underlying Figure 5a). Vacuous and absent gates cannot observe a regression, so their breakage rate is a measurement artifact rather than evidence of safer behavior.

Gate strength	Instances	Scored runs	NGI median	NGI mean	Breakage	No-regression
detected	24	228	+0.333	+0.321	13%	86%
blind	27	247	+0.292	+0.315	8%	88%
vacuous	3	28	+0.271	+0.289	0%	86%
none	6	54	+0.250	+0.257	n/a	85%

change from headroom-normalized change and relate both to patch size and verification effort. Intervention locality across tests, workflows, documentation, packaging, dependency manifests, and source modules distinguishes targeted governance edits from broad source rewrites.

Table 20: Instances that drift under the no-op condition. Dimension changes are base → treated on an identical repository. Five of ten instances drift; the median is 0.000 and the standard deviation is 0.073.

Instance	NGI	Dimension drift
DreamWall-Animation/dwpicker	−0.1667	D4 4 → 3
jiajun613/Efficient-WAM	−0.1111	D5 2 → 1
TsingZ0/HtFLlib	+0.0972	D4 3 → 4, D5 1 → 2
fblissjr/ComfyUI-QwenImageWanBridge	+0.0556	D4 3 → 4
rivitna/Malware	+0.0333	D3 2 → 3

Table 21: Per-instance NGI on the frozen batch, one representative model against the three non-agent conditions. Bold marks the best of the four on each repository. Qwen3.8-Max loses to at least one template on four of ten repositories.

Instance	Qwen3.8-Max	Mechanical	Rule-based	No-op
sleeepeer/PoisonedRAG	+**0.750**	+0.264	+0.292	+0.000
rivitna/Malware	+**0.708**	+0.236	+0.275	+0.033
THUNLP-MT/StreamingBench	+**0.708**	+0.347	+0.208	+0.000
SkyworkAI/Skywork-Skills	+**0.667**	+0.333	+0.333	+0.000
DreamWall-Animation/dwpicker	+**0.500**	+0.286	+0.250	−0.167
avbor/HomeAssistantConfig	+**0.375**	+0.308	+0.292	+0.000
fblissjr/ComfyUI-QwenImageWanBridge	+0.167	+**0.292**	+0.208	+0.056
jiajun613/Efficient-WAM	+0.167	+0.236	+**0.292**	−0.111
dama-cyber/magic-distillation	+0.083	+**0.208**	+0.097	+0.000
TsingZ0/HtFLlib	−0.042	+0.208	+**0.292**	+0.097

The public subset provides one rollout per model-instance pair, so inferential claims are limited accordingly. Section 5.9 quantifies the cost of reducing this limitation: reaching a resolution of 0.10 NGI requires roughly 110 paired observations, about five rollouts per model-instance pair on the current subset. Private instances and unbalanced model coverage are not treated as interchangeable with the shared public subset, and exclusions are reported before aggregation.

Repeated seeds, same-scaffold reruns, and expert treatments are outside the primary comparison. Repeated seeds should cover small and large repositories, strong and weak base governance, and each gate-strength category. Same-scaffold reruns are important because one model used Claude Code while the other nine used pi.

D.7. A corrected accounting bug

An earlier version of the pipeline recorded all six `gate=none` instances as behavior-broken, because applying a nonexistent verification patch exits with a nonzero status. The gate had in fact never run on those instances. Correcting this lowered per-model breakage rates over all 60 instances from 15–17% to 6–7% for the affected models; the rates in Table 18 are post-correction. The public 22-instance subset is unaffected, since every public instance carries a verification patch, so Table 1 required no revision.

E. Detailed Analysis and Case Studies

E.1. Case-study selection

The qualitative case-study protocol selects five cases: one high-NGI safe intervention, one high-NGI risky intervention, one low-NGI conservative intervention, one baseline-sensitive case, and one gate-strength disagreement.

Selection is performed from the frozen result table using predefined criteria, not by choosing only visually persuasive examples.

E.2. Case-study record

Each case study contains repository context, base governance profile, model diagnosis, changed-file map, executed verification, dimension-level score changes, behavior outcome, and an auditor's assessment of whether the patch solved a meaningful maintenance problem. The developer or construction patch is not needed to explain a governance case and should remain hidden if it is part of protected evaluation.

Table 22: Qualitative case-study record.

Field	Recorded content
Repository context	Purpose, ecosystem, size, and baseline governance gaps
Agent diagnosis	Evidence cited before editing
Intervention	Files and governance dimensions changed
Verification	Commands, outcomes, and behavior status
Score change	D1–D6 base and treated scores
Audit judgment	Useful, superficial, risky, or mixed, with rationale

E.3. Failure vignettes

For broken or low-value runs, the paper should show the earliest failure: environment misunderstanding, unsupported dependency change, ineffective test addition, vacuous CI, broad source churn, or insufficient verification. These vignettes are important because the benchmark evaluates safe maintenance rather than only the presence of new files.

F. Reproducibility and Release

F.1. Directory-level release layout

The release mirrors the separation between public tasks, protected verification assets, results, and traces. Public artifacts are sufficient to reconstruct the announced experiments without revealing private outcomes or hidden answers. A manifest at the root maps every paper table to a script and every script output to a versioned input.

Table 23: Recommended release package layout.

Path	Content
tasks/	Public task metadata, commits, licenses, and applicability masks
environments/	Container definitions or environment locks
probes/	Versioned governance and behavior probe definitions
runs/	Canonical traces, patches, budgets, and evidence records
results/	Machine-readable per-instance and aggregate scores
analysis/	Scripts that regenerate paper tables and figures
docs/	Construction notes, scoring rubric, changelog, and limitations

F.2. Artifact completeness rules

An artifact is complete only when its manifest exists, its digest matches the stored file, and the downstream table can identify whether it was used. An empty directory is not treated as evidence of absence. If a run is interrupted, the manifest records the last completed stage and the reason for interruption.

F.3. Security and privacy

Repository licenses, credentials, private issue data, evaluator patches, and model-provider traces require separate review for release. Secret scanning is performed on artifacts and logs, but a positive scanner result is manually adjudicated because repository content may contain intentional examples. Private tasks remain excluded from public traces, score files, and aggregate tables.

F.4. Final paper update procedure

The auxiliary settings and results are incorporated in the main text and Figure 4a. Updates should regenerate summaries from the machine-readable records, preserve the frozen manifest, distinguish valid from broken runs, and rerun the full PDF visual check.

F.5. Data Dictionary and Researcher Checklist

F.5.1. Core identifiers

The following identifiers should remain stable across the pilot and final releases. The instance identifier names a repository snapshot, the run identifier names one model execution, and the evidence identifier names one probe invocation. A score record must reference all three identifiers.

Table 24: Data dictionary for the benchmark records.

Name	Type	Definition
instance_id	string	Repository and pinned-snapshot identifier
run_id	string	Unique model/scaffold execution identifier
model	string	Exact model name or provider label
scaffold	string	Agent framework and runner version
base_commit	string	Commit used to reconstruct the untreated state
patch_digest	string	Digest of the submitted unified patch
behavior	enum	preserved, broken, invalid, or unavailable
gate_strength	enum	detected, blind, vacuous, or none
ngi	float	Headroom-normalized governance improvement
evidence_status	enum	pass, fail, timeout, unavailable, or not applicable

F.5.2. Researcher checklist

Before claiming an auxiliary result, confirm that the ten repositories were frozen in advance, every condition used the same base commit, no hidden evaluator asset was exposed, no-op results are stable, broken behavior runs are not silently dropped, score denominators are printed, and every aggregate number can be regenerated from the archived per-instance records. The ten-repository baselines and gate-ablation values are measured; repeated seeds, same-scaffold reruns, and expert treatments are outside the primary comparison.

F.5.3. Interpretation checklist

When comparing conditions, first check paired coverage, then behavior validity, then per-dimension change, and only then aggregate NGI. A higher aggregate score is not interpreted as an improvement if it is caused by a smaller denominator, a weaker gate, or a higher breakage rate. This ordering is especially important for the small ten-repository pilot, where one instance can materially affect the mean.