SWE-Prometheus: Benchmarking Repository-Level Engineering Governance under Real-World Constraints

Jiajun Wu[1] **Anonymous Authors**

[1] *Affiliations withheld for working draft*

Abstract: Coding agents have made rapid progress on repository-level software engineering tasks, but existing evaluations usually begin with a known issue and a predefined success signal. This setting measures targeted functional repair while leaving repository governance under-evaluated: discovering undocumented engineering risks, improving tests and quality infrastructure, making projects reproducible, and preserving behavior during maintenance. We introduce **SWE-Prometheus**, a benchmark in which an agent receives a fixed snapshot of a real repository and a general governance objective, but no defect list, base score, or oracle patch. The agent must inspect the repository, prioritize risks, implement a retrofit, and verify its claims under a limited budget. The benchmark evaluates six dimensions—Tests & CI, Code Quality Gates, Documentation & Collaboration, Structure & Maintainability, Reproducible Environment, and Dependency & Security Health—using executable base/treated evidence, hidden characterization tests, and multi-teacher scoring. The dataset60 release contains 60 repositories, including a 22-instance public shared subset and 38 private instances, with one rollout for each of ten models on the public subset. Mean Normalized Governance Improvement ranges from 0.0568 to 0.5760, while observed behavior-breakage rates range from 0% to 23%. Four auxiliary diagnostics calibrate these numbers: a no-op baseline puts the scorer's noise floor at 0.073, so differences below roughly 0.15 are not capability differences; a repository-blind template that adds seven fixed configuration files reaches 0.275 and outscores three of ten models, but earns exactly zero on the two dimensions substantiated by container execution; and measured behavior breakage falls monotonically from 13% to 0% as mutation-tested gate strength degrades, showing that weak gates make agents look safer rather than making them safer. Repository governance therefore yields measurable differences between agents, but only where the evaluator executes—which motivates repeated runs, execution-backed credit for currently gameable dimensions, stronger behavior gates, independent judge calibration, and controlled scaffolds before making stable ranking or production-readiness claims.

Technical report, working draft; artifact links and author affiliations to be finalized.

1. Introduction

Large language model based coding agents can now search unfamiliar repositories, execute development commands, edit multiple files, and produce end-to-end patches. 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], and agent scaffolds such as SWE-agent, Agentless, and OpenHands turned that setting into a standard evaluation harness [2–4]. Later benchmarks improved task curation, repository coverage, language coverage, and execution realism [5–9].

However, as illustrated in Figure 1, most of these evaluations start after a human has already identified the problem. The issue statement tells the agent what to investigate, while tests or a reference patch provide a relatively clear success signal. This setting is well suited to targeted functional repair, but it leaves another important form of software engineering largely unmeasured: *repository governance*.

In real projects, engineering risks are distributed across the repository rather than expressed as one issue. Tests may be absent, shallow, skipped, or disconnected from CI [10]. Lint and type checking may be configured but fail on a clean checkout. Installation may depend on an undocumented local environment, a failure mode documented at scale for research code [11]. Documentation may explain usage but not maintenance or security procedures. Dependencies may be implicit or uncontrolled, exposing the project to known supply-chain risk [12, 13], and complex modules may be difficult to modify safely [14, 15].

This matters because a repository can pass its current functional tests while remaining difficult to reproduce,

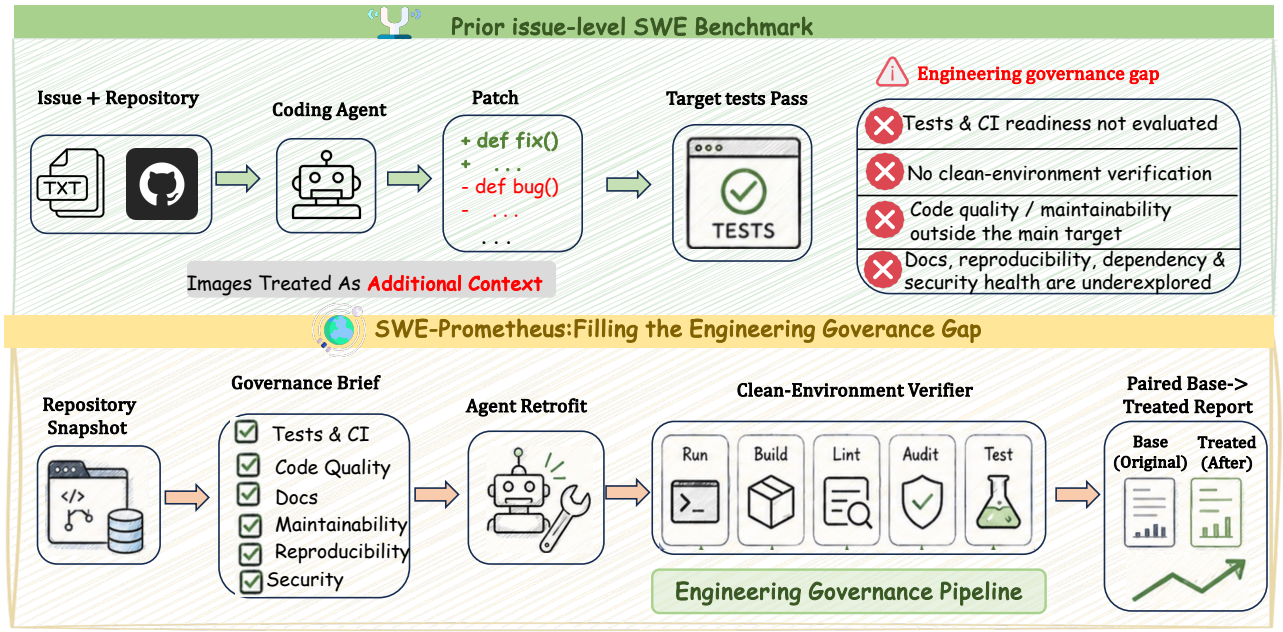


Figure 1: Motivation for SWE-Prometheus. Prior issue-level SWE benchmarks hand the agent a localized defect and score a patch against target tests, leaving testing and CI readiness, clean-environment behavior verification, quality and maintainability outside the patch target, and documentation, reproducibility, 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.

extend, audit, or hand over. Conversely, an agent can add modern-looking configuration while introducing broken commands, ineffective tests, dependency drift, or behavioral regressions. File-presence checks and agent self-reports therefore do not provide sufficient evidence of engineering improvement. Our own template baselines make this concrete: a fixed set of seven configuration files, added without reading the repository at all, reaches a normalized improvement of $+0.275$ and outscores three of ten evaluated models (Section 6).

We address this gap with SWE-Prometheus, a benchmark for autonomous repository governance. The key design is to make the agent responsible for both diagnosis and intervention. Given a repository and a general retrofit objective, the agent must decide what is important, allocate a limited budget, implement changes, and demonstrate that the changes work. We compare the repository before and after intervention and separately verify whether existing behavior is preserved.

Research questions.

1. Can an agent discover important engineering risks without an explicit issue list?
2. Can it prioritize useful governance work under a limited budget?
3. Can it turn superficial engineering changes into executable infrastructure?
4. Can it improve repository health without breaking existing behavior?

Contributions.

- We formulate *Repository Retrofit*, an issue-free repository-level task for autonomous engineering governance.
- We define a six-dimensional governance model spanning testing, quality gates, documentation, maintainability, reproducibility, and dependency/security health.
- We establish an evidence-backed protocol combining base/treated execution, hidden behavior verification, structured probes, and multi-teacher scoring.

- We release dataset60 with 60 real repositories, public and private splits, instance-level results, patches, evidence, and rollout artifacts.
- We provide an initial ten-model analysis showing differentiated governance improvement and behavior-regression risk.
- We report a set of auxiliary diagnostics—a no-op scoring floor, two non-LLM template baselines, a behavior-gate ablation, and a gate-strength stratification—that calibrate how much of the reported signal is capability rather than measurement artifact.

2. Related Work

2.1. Repository-level software engineering benchmarks and agents

SWE-bench evaluates agents on real repository issues using executable validation [1], and its descendants improve task reliability, cost, language coverage, and freshness: a human-filtered subset removes underspecified and mis-validated instances [5], multimodal and multilingual variants broaden the input and language distribution [6–8], and completion-oriented suites measure cross-file context use [9]. On the agent side, SWE-agent introduced the agent–computer interface abstraction [2], Agentless showed that much of the reported gain comes from localization and repair structure rather than agency [3], and OpenHands packaged the setting as an open execution platform [4].

All of these evaluations share one assumption: the engineering target is given. SWE-Prometheus shares their fixed revisions, isolated execution, patch format, and executable checks, but changes the task objective from fixing a known defect to *diagnosing and improving repository engineering state*. The agent receives no issue, no failing test, and no reference patch.

2.2. Repository health and software quality

Static analysis, code-quality systems, dependency auditors, community-standards checklists, and supply-chain tools such as OpenSSF Scorecard [16] provide useful repository-health signals, and an empirical literature quantifies why these signals matter: continuous integration changes project outcomes and costs [10], complexity and technical debt shape maintainability [14, 15], dependency networks propagate vulnerabilities [12, 13], and research artifacts frequently fail to re-execute from a clean checkout [11].

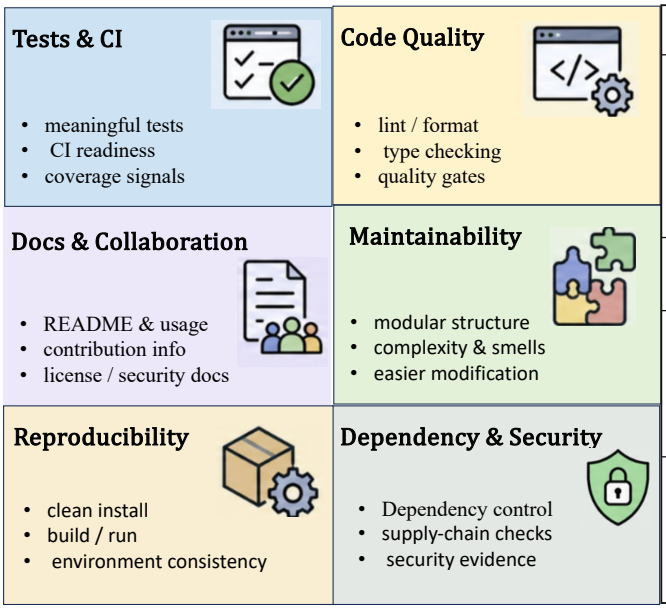
These tools measure individual properties or configuration states. They do not evaluate whether an autonomous agent can decide which weaknesses matter, coordinate changes across dimensions, and preserve the original behavior. In SWE-Prometheus such tools are *evidence probes* rather than the benchmark objective: a configuration contributes only when its scope and execution result support an effectiveness claim. Section 6 shows this distinction is load-bearing—template baselines saturate the three configuration-facing dimensions while scoring exactly zero on the two dimensions backed by container execution.

2.3. Behavior preservation and verification strength

Retrofitting a repository without a specification is the classic legacy-code setting, where characterization tests pin current behavior before it is changed [17]. Because a passing test suite is not by itself evidence of a discriminative one, we adopt mutation testing to measure gate strength [18], following evidence that mutants are a usable proxy for real fault detection [19]. This matters directly: our gate-strength stratification shows measured breakage rates falling monotonically as gates get weaker, which is a property of the instrument rather than of the agents.

2.4. Benchmark validity and LLM judges

Recent benchmark-validation work shows that weak tests can change conclusions about code-generation capability [20], while leakage and incomplete curation can affect repository-level comparisons. SWE-Prometheus follows this principle by retaining failed runs, measuring characterization-test strength through mutation testing, separating public and private data, and preserving per-run evidence. Because part of our rubric is judged by



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: Overview of SWE-Prometheus dataset60. Left: the six governance dimensions and the concrete properties each one targets. Right: release statistics, gate-strength composition, and the reference points that bound interpretation—the best public normalized governance improvement, the no-op scoring noise floor, and the two non-LLM template baselines.

models rather than by probes, we also inherit the known limitations of LLM-as-a-judge evaluation—position and verbosity bias, and correlated errors among judges from one family [21, 22]. We therefore report a no-op scoring floor that bounds judge noise empirically (Section 6) instead of assuming the judge is exact. The remaining gap is an integrated evaluation of repository governance in which the agent itself must discover the engineering target.

3. Benchmark Overview

3.1. Characteristics of SWE-Prometheus

The dataset60 release contains 60 real GitHub repositories at fixed base commits: 22 public instances shared by all current models and 38 private instances reserved for internal evaluation. Ten models have completed one rollout each on the public subset, yielding 220 public model–instance pairs. Figure 2 summarizes the governance dimensions and the release statistics, including the distribution of behavior-gate strengths and the headline reference points established by our auxiliary baselines.

3.2. Task specification

Each instance is a real repository at a fixed `base_commit`. The agent receives the following 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 agent receives no issue list, base score, hidden test, or reference patch. Multiple governance strategies are valid, and the agent chooses which dimensions to address. Formally, given a repository snapshot $\mathcal{R}$ and a governance brief, the agent produces a patch $\mathcal{P}$ under a budget $\mathcal{B}$; the evaluator scores the pair $(\mathcal{R}, \mathcal{R} \oplus \mathcal{P})$ under the rubric $\mathcal{G}$ and separately checks behavior preservation with a hidden verification kernel $\mathcal{V}$.

Information boundary. The agent can access the repository, public documentation, repository metadata, and development tools. It cannot access base or treated scores, hidden characterization tests, verification patches,

Table 1: Six governance dimensions. Configuration presence alone is never sufficient evidence of improvement. The rightmost column reports how each dimension is primarily substantiated, which turns out to predict which dimensions a template patch can and cannot game (Table 4).

Dimension	Evaluation target	Primary evidence
D1 Tests & CI	Meaningful tests for important behavior and executable CI paths.	Config + execution
D2 Code Quality Gates	Effective lint, formatting, type-checking, and related gates.	Config + execution
D3 Documentation & Collaboration	Actionable usage, maintenance, contribution, license, and security information.	Content review
D4 Structure & Maintainability	Module boundaries, dependency direction, complexity, and maintainable organization.	Static probes
D5 Reproducible Environment	Clean installation, construction, execution, and version consistency.	Container execution
D6 Dependency & Security	Controlled dependencies and available supply-chain and security evidence.	Container execution

evaluator evidence, or other model results. The evaluator records the task prompt, model snapshot, scaffold, container, tool versions, and run metadata.

3.3. Governance dimensions

The benchmark scores six dimensions from 1 to 5. A score of 4 denotes acceptable governance for the repository type and task scope.

3.4. Evaluation pipeline

Figure 3 shows the end-to-end path from task materials to the paired governance report. The evaluator first collects executable baseline evidence on the untouched snapshot, then runs the agent retrofit, then rebuilds the treated repository from scratch in a clean container and re-runs the same probes together with the hidden behavior gate. Only evidence that survives this clean rebuild enters the score.

4. Dataset Creation and Evaluation Protocol

4.1. Repository selection

Candidate repositories are discovered using signals such as missing CI, weak tests, absent quality gates, incomplete collaboration documentation, unpinned dependencies, and maintenance gaps. These signals support high-recall discovery and stratification; they are not treated as ground truth. Repositories are screened for meaningful software functionality, a stable base commit, usable distribution status, plausible governance headroom, and evaluability without private credentials or paid services.

4.2. Base and treated states

For each rollout, the evaluator reconstructs the base repository and creates an independent treated snapshot by applying the agent patch. The same probes run on both states: installation and environment checks; test collection and execution; lint, formatting, type checking and build checks; complexity and maintainability probes; and dependency and security probes. Commands, exit codes, logs, environment versions, timeouts, and unavailable evidence are retained.

4.3. Behavior gate

Each usable verification patch is validated on the base snapshot and then run on the treated snapshot. If treated verification fails, the rollout is marked `behavior_broken`: it is excluded from valid NGI aggregation but remains in

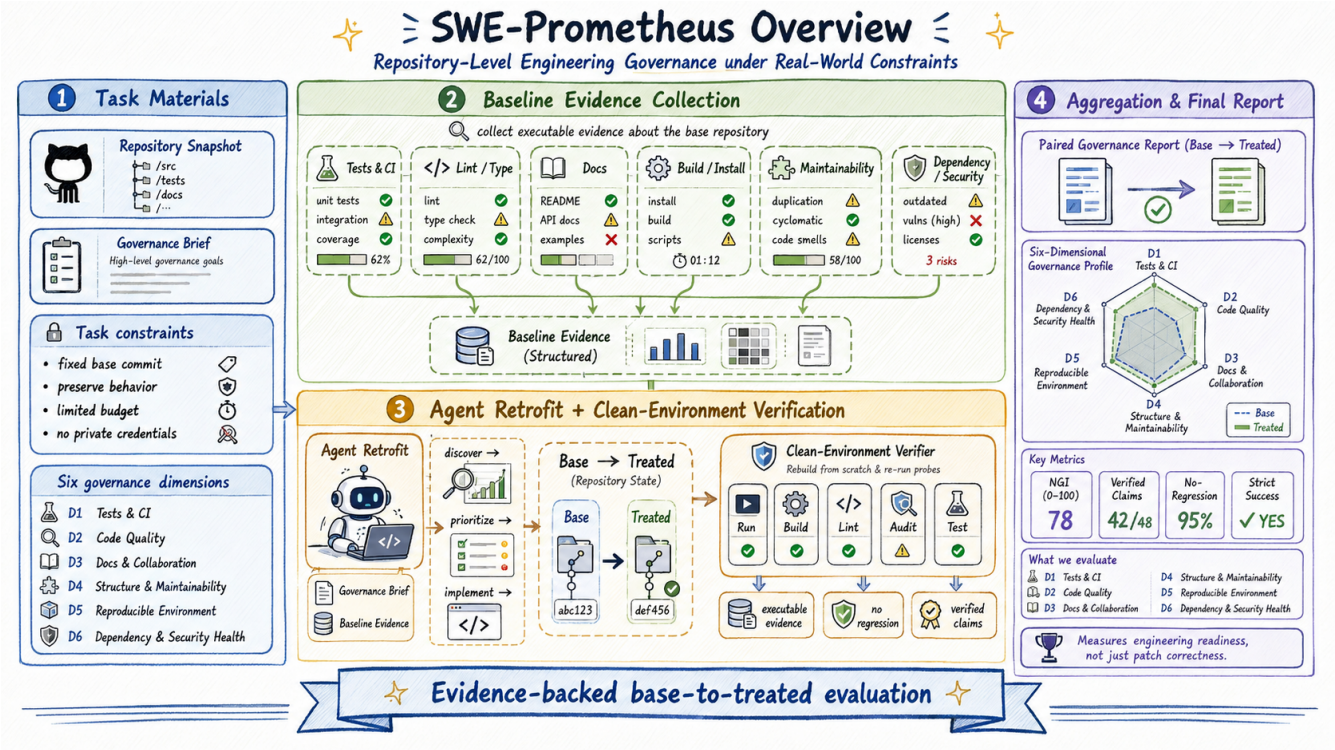


Figure 3: SWE-Prometheus construction and evaluation pipeline. (1) The agent receives a repository snapshot, a governance brief, and task constraints—fixed base commit, behavior preservation, limited budget, no private credentials. (2) Deterministic probes collect structured baseline evidence across the six dimensions. (3) The agent retrofits the repository; a clean-environment verifier rebuilds from scratch and re-runs run/build/lint/audit/test probes plus the hidden behavior gate. (4) Three teacher models score base and treated states under the fixed rubric, producing a paired report with per-dimension profile, NGI, verified-claim rate, no-regression rate, and strict success.

behavior-breakage analysis. Because a passing gate is only as informative as the gate is discriminative, mutation testing [18, 19] provides a second label for gate strength: `detected`, `blind`, `vacuous`, or `none`. In dataset60 these labels cover 24, 27, 3, and 6 instances respectively; Section 6 shows they must be reported alongside any breakage number.

4.4. Judge

Deterministic probes generate execution evidence. Three teacher models independently score base and treated states using the fixed governance rubric. The release retains individual scores, justifications, evidence references, aggregate scores, and disagreement information. Expert calibration is not yet complete; therefore, teacher agreement is reported as internal scorer agreement rather than external validity, and the empirical noise floor of Section 6 is the operative bound on how small a difference may be interpreted.

4.5. 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

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

Table 2: Public shared-subset results. Valid counts and behavior-breakage counts are out of 22 instances. Claude used Claude Code; the other models used pi, so its row is a cross-scaffold reference rather than a controlled comparison.

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

where D_i contains the defined and scorable dimensions. A dimension with base score 5 receives no improvement reward, but is still checked for regression.

We report NGI mean, median, and dispersion; per-dimension improvement; treated absolute scores; valid and behavior-broken rates; no-regression rate; strict governance success; verified claim rate; and token, turn, time, and tool-call budgets. NGI is a normalized change metric, not an absolute readiness score or a production-failure probability.

4.6. Auxiliary diagnostic protocol

The main table alone cannot say whether a reported gap is capability or measurement artifact. We therefore run four diagnostics on a fixed 10-repository subset drawn from the public 22 by stratified sampling that balances repository size (3 large / 4 medium / 3 small), gate strength (6 detected / 3 blind / 1 vacuous), and base governance level (4 low / 4 medium / 2 high). No instance in the public subset has gate strength `none`, so the gate-strength stratification instead uses all 60 instances, where `none` occurs 6 times.

- **No-op baseline.** An empty patch is pushed through the full evidence and scoring pipeline. Any nonzero NGI is measurement noise, so its dispersion is an empirical floor on interpretable differences.
- **Mechanical baseline.** A fixed set of seven files—CI workflow, pre-commit config, `mypy.ini`, `CONTRIBUTING`, `SECURITY`, issue templates, and an `assert True` smoke test—is added without inspecting the repository.
- **Rule-based baseline.** A non-LLM script first detects which artifacts are missing, then adds only those, and generates a real `importlib.import_module` smoke test against the inferred package name instead of a vacuous assertion.
- **Behavior-gate ablation.** NGI is recomputed with the gate disabled (all scored instances counted) and stratified by mutation-testing gate strength.

All four diagnostics reuse existing rollouts and add no new model calls, so they are directly comparable to the main results rather than a separate experiment.

5. Results

5.1. Public shared-subset outcomes

Table 2 reports the headline result: ten models, one rollout each, on the 22 public instances. Mean NGI spans an order of magnitude, from 0.0568 to 0.5760, while measured behavior breakage spans 0 to 5 of 22 instances.

The results support three observations. First, repository governance yields measurable differences between agents: the spread is far larger than the 0.073 scoring noise floor established in Section 6. Second, aggressive modification is not automatically better—high NGI coexists with behavior regressions, and the two models with zero measured breakage are also the two lowest scorers. Third, models display different dimension-specific profiles, with configuration-heavy improvements generally easier than deep structural maintenance and security evidence. These are descriptive observations from a single rollout, not stable rankings.

5.2. Governance improvement is not one scalar capability

An agent may improve tests and environments while leaving security or structure weak, or may make broad changes at the cost of behavioral safety. This is why the benchmark reports six dimensions and behavior outcomes separately rather than reducing every rollout to a pass/fail label. Table 4 makes the profile explicit and, read together with the template baselines, separates the dimensions that reward file addition from those that require working software.

5.3. Reading the ranking conservatively

The public subset is a shared comparison set, not a random sample of GitHub repositories, and each model–instance pair has been run once. Claude's row is produced under a different scaffold. Differences below roughly 0.15 in NGI—twice the measured noise floor—should not be read as capability differences at all; by that standard Table 2 supports a coarse grouping into a high band (Kimi-K3, GLM-5.3-Flash, Claude-Opus-5), a middle band (Qwen3.8-Max, GLM-5.2, DeepSeek-V4-Pro, GLM-5.3, GPT-5.6-Sol), and a low band (DeepSeek-V4-Flash, MiniMax-M3), rather than a ten-way ordering.

6. Analysis

This section reports the four auxiliary diagnostics of Section 4.6. All of them reuse existing rollouts on the fixed 10-repository subset (or, for gate strength, all 60 instances) and add no new model calls.

6.1. The scorer has a noise floor of 0.073

Pushing an empty patch through the full pipeline should yield NGI $= 0$ everywhere. It does not. Five of ten instances drift, with a largest single deviation of -0.167 and a standard deviation of $\sigma = 0.073$. The median is exactly 0.000, so the scorer has no systematic bias—it does not reward an unchanged repository—but the dispersion is real.

Drift concentrates in D4 (three instances), D3 and D5 (two each). D5 drift is attributable to environment nondeterminism: the installation probe can succeed on one run and fail on the next. D4 drift is the more troubling case, because structural probes on an unchanged diff should be fully deterministic.

We therefore adopt the following reading rule throughout the paper: **NGI differences smaller than about 0.15 (2σ) should not be interpreted as capability differences.**

6.2. A repository-blind template scores 0.275 and beats three models

Table 3 places the two non-LLM baselines into the model ranking on the fixed 10-repository subset. The Mechanical baseline—seven fixed files, added without ever reading the repository—reaches a median NGI of $+0.275$. That is above GLM-5.3 ($+0.271$), DeepSeek-V4-Flash ($+0.174$), and MiniMax-M3 ($+0.042$), and statistically indistinguishable from DeepSeek-V4-Pro ($+0.281$).

This is the most important self-check in the paper, and it cuts both ways. Table 4 shows where the template score comes from: the Mechanical and Rule-based baselines improve D1, D2, and D3 on essentially every instance (100%, 100%, 90–100%), and score *exactly zero* on D5 and D6, with only 20% on D4.

The correct conclusion is therefore not that the benchmark rewards adding files. It is sharper: **three of six dimensions (D1, D2, D3) are gameable by a template, and three (D4, D5, D6) are not.** Reproducible environment and dependency/security health are substantiated by container execution, so a configuration file

Table 3: Auxiliary diagnostics on the fixed 10-repository subset. Models and non-LLM baselines are ranked jointly by median NGI. The No-op row is the scoring noise floor, not a method. Rows within 0.15 of one another are not separated by this instrument.

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

Table 4: Per-dimension improvement rate on the fixed 10-repository subset. Fraction of scored instances on which each dimension improved. Template baselines saturate the three configuration-facing dimensions and score zero on the two dimensions substantiated by container execution.

Method	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%
Baseline: Mechanical	100%	100%	90%	20%	**0%**	**0%**
Baseline: Rule-based	100%	100%	100%	20%	**0%**	**0%**
Baseline: No-op	0%	0%	10%	20%	10%	0%

that does not make the project install or audit cleanly earns nothing. This is also the clearest available separation between models and templates: the top models improve D5 and D6 on 55–100% of instances where both templates improve them on none.

6.3. Repository-aware rule following adds nothing over a fixed template

The Rule-based baseline detects what is missing before adding it and emits a real `importlib.import_module` smoke test rather than `assert True`. It scores +0.283 against Mechanical's +0.275—a gap of 0.008, an order of magnitude inside the noise floor. Under the current rubric, conditioning the patch on repository state buys nothing. The gap between agents and templates lives entirely in work a template cannot do: restructuring modules, repairing dependency specifications, and making a clean environment actually build and run.

Table 5: Behavior-gate ablation and gate-strength stratification. Left: NGI median with the gate off (all scored instances) versus on (current protocol), over all instances available per model. Right: outcomes stratified by mutation-testing gate strength over all 60 instances × 10 models.

Model	Gate off	Gate on	Infl.	Broken
Claude-Opus-5	+0.677	+0.677	+0.000	18%
Kimi-K3	+0.517	+0.531	−0.014	7%
GLM-5.3-Flash	+0.476	+0.458	+0.017	12%
Qwen3.8-Max	+0.375	+0.396	−0.021	13%
GLM-5.2	+0.375	+0.367	+0.008	11%
DeepSeek-V4-Pro	+0.367	+0.367	+0.000	15%
GPT-5.6-Sol	+0.229	+0.250	−0.021	6%
GLM-5.3	+0.215	+0.208	+0.007	11%
DeepSeek-V4-Flash	+0.125	+0.125	+0.000	6%
MiniMax-M3	+0.000	+0.000	+0.000	6%

Strength	Inst.	NGI med.	Brk.	No-reg.
detected	24	+0.333	**13%**	86%
blind	27	+0.292	**8%**	88%
vacuous	3	+0.271	**0%**	86%
none	6	+0.250	n/a	85%

Measured breakage falls monotonically as the gate weakens.

Vacuous and absent gates cannot catch a regression, so their 0% is a measurement artifact, not safer behavior.

6.4. The behavior gate does not change the ranking—it flags untrustworthy scores

Table 5 recomputes NGI with the gate disabled. All ten rank positions are unchanged, and the inflation attributable to the gate is small and frequently *negative*: instances caught by the gate tend to score below their model's average, not above it.

This is counterintuitive but informative: **when an agent breaks behavior, it usually also fails to improve the six dimensions.** Behavior breakage is not the price of an aggressive high-scoring rewrite; it is mostly a symptom of a botched one. The gate's value therefore is not to stop score inflation but to mark which scores are untrustworthy.

The stratification on the right of Table 5 is the direct argument for mutation testing. Measured breakage falls monotonically as gates get weaker—13% under detected gates, 8% under blind gates, 0% under vacuous gates—while median NGI barely moves. Weak gates do not observe safer agents; they fail to observe unsafe ones. Any headline breakage number must therefore be reported with its gate-strength breakdown.

6.5. Key insight: presence is not effectiveness

Taken together, the diagnostics support one claim. A repository can contain a test suite, CI workflow, lint configuration, or dependency file without obtaining the intended engineering benefit, and a scoring rubric that inspects artifacts rather than executing them cannot tell the difference—a repository-blind template outscores three of ten models on exactly the three dimensions where inspection substitutes for execution. What separates agents from templates is the execution-backed remainder: making the project install cleanly, audit cleanly, and remain structurally maintainable. Those dimensions, plus a behavior gate whose discriminative power has been demonstrated, are what make the measurement mean anything.

7. Limitations and Future Work

Verification coverage. Only 24 of 60 instances currently have behavior gates whose discriminative power is demonstrated through mutation testing. The remaining instances support governance-state analysis but provide weaker behavioral evidence, and Table 5 quantifies exactly how much weaker: measured breakage drops from 13% to 8% to 0% as gate strength degrades, with no corresponding change in governance scores.

Single rollout and scoring noise. Each model–instance pair has been run once, so rollout variance, infrastructure flakiness, and rank stability are not yet estimated. The no-op baseline bounds the scorer's contribution at $\sigma = 0.073$, which already tells us the ten-way ordering in Table 2 is really a three-band grouping, but it does not bound the agent's own run-to-run variance. Repeated rollouts are the highest-value next experiment: they would attach error bars to every row and refine the noise estimate per model rather than pooling it.

Judge calibration. The current judge is multi-teacher but has not completed independent expert calibration, and all three teachers come from one model family that overlaps with part of the evaluated set. Subjective dimensions such as documentation and maintainability may therefore contain correlated judge errors [21, 22]; the D4 drift observed under the no-op baseline (Section 6.1) is a concrete symptom, since structural probes on an unchanged diff should be deterministic. Future evaluation should add blinded expert annotations, adjudication, cross-family judges, and adversarial calibration.

Rubric gameability. Three of six dimensions are demonstrably reachable by a repository-blind template (Section 6.2). This is a property of the current rubric, not only of the models. D1, D2, and D3 should be tightened so that credit requires demonstrated scope and execution—a CI workflow that runs, a lint gate that reports on real files, a smoke test that exercises real behavior—bringing them closer to how D5 and D6 are already substantiated.

Scaffold confounds. Different scaffolds expose different tools, prompts, context handling, and termination behavior. Claude-Opus-5 ran under Claude Code while all other models ran under pi, so cross-scaffold comparisons are not causal model comparisons. Claude also shows the largest dispersion of any method on the 10-repository subset (SD $= 0.354$) with only nine valid samples, so its position should be read with particular caution. Some budget artifacts also have unavailable monetary cost, so current efficiency analysis should use tokens, turns, wall-clock time, and tool calls.

Release hygiene. For external release, private tasks, private verification patches, private results, and other hidden artifacts must remain access-controlled. Public traces should include an artifact manifest so that missing evidence, patches, or logs are explicit. Finally, dependency and security scores measure available probe evidence, not comprehensive vulnerability absence.

Next steps. In priority order: repeated rollouts with confidence intervals; expert reference treatments and blinded human annotation; stronger characterization tests to convert blind and vacuous gates into detected ones; same-scaffold reruns to remove the Claude confound; cross-family judges; and a genuinely isolated private evaluator. The mechanical and rule-based template baselines called for in earlier drafts are now included and reported in Section 6.

8. Conclusion

SWE-Prometheus introduces repository-level engineering governance as a benchmark task for coding agents. Instead of telling an agent which defect to fix, it asks the agent to inspect a real repository, identify engineering risks, choose a retrofit strategy, make changes under a limited budget, and demonstrate that the changes work without breaking existing behavior.

The dataset60 release shows that this capability is measurable. Across 60 real repositories and 10 models, agents differ substantially in governance improvement, behavior-breakage risk, and dimension-specific preferences, and the spread is an order of magnitude larger than the measured scoring noise floor. The results also show why executable evidence and behavior verification are necessary: engineering changes that look correct in a diff may fail when executed or may damage existing behavior.

The auxiliary diagnostics sharpen this into a design lesson for governance benchmarks generally. A repository-blind template that adds seven fixed files outscores three of ten models, but only on the three dimensions judged from artifacts; it earns nothing on the two dimensions substantiated by container execution. Behavior gates do not change the ranking, yet weakening them makes agents appear safer purely as a measurement artifact. Both findings point the same way: governance can only be measured where the evaluator executes, and any dimension scored from presence alone will eventually be scored from presence alone.

The current benchmark is best understood as a research resource for studying repository stewardship, not as a definitive measure of production readiness. Repeated rollouts, independent judge calibration, stronger hidden gates, controlled scaffolds, execution-backed credit for the currently gameable dimensions, complete artifact

manifests, and a genuinely isolated private holdout are the next steps toward reliable comparison of coding agents' repository-governance capabilities.

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, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

[3] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[4] Xingyao Wang, Boxuan Chen, Yufan Song, Frank F. Li, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

[5] OpenAI. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

[6] John Yang, Carlos E. Zhang, Carlos E. Jimenez, Ofir Press, Karthik Narasimhan, and Shunyu Yao. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *International Conference on Learning Representations*, 2025.

[7] Daoguang Zan, Zhirong Huang, Ailun Yu, Shaoxin Lin, Yifan Shi, Wei Liu, et al. SWE-bench-java: A GitHub issue resolving benchmark for Java. *arXiv preprint arXiv:2408.14354*, 2024.

[8] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, et al. Multi-SWE-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.

[9] Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, 2024.

[10] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 426–437, 2016.

[11] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of Jupyter notebooks. In *IEEE/ACM International Conference on Mining Software Repositories*, pages 507–517, 2019.

[12] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *IEEE/ACM International Conference on Mining Software Repositories*, pages 181–191, 2018.

[13] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43, 2020.

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

[15] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6): 18–21, 2012.

[16] Open Source Security Foundation. OpenSSF scorecard. https://github.com/ossf/scorecard, 2024.

[17] Michael C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall, 2004.

[18] 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.

[19] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 654–665, 2014.

[20] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.

[21] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.

[22] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023.

A. Release and Accounting Details

Release structure. The current internal release contains public tasks, verification patches, results, traces, and leaderboards, together with a private directory for internal evaluation. A public distribution should contain only the public subset. Private task files, verification patches, results, and hidden evaluation artifacts must remain outside the distributed package.

Per-run artifacts. A complete model–instance record contains the patch, agent output, stderr, base evidence, treated evidence, verification result, budget record, and score record. Each artifact should be keyed by model, instance, run identifier, evaluator version, and container version. Missing artifacts must be represented in a manifest with an explicit reason.

Accounting. The valid NGI denominator excludes behavior-broken runs and other runs that cannot produce a valid treated score. Behavior-broken, environment-failed, no-patch, and infrastructure-failed runs remain in the corresponding failure statistics. Monetary cost is reported only when provider accounting is available; otherwise the result is marked unavailable rather than imputed.

Verification strength. Mutation-testing labels are reported separately from verification outcome. A preserved result under a detected gate, a blind gate, a vacuous gate, and no gate are not equivalent evidence. All headline claims should provide both the overall number and the gate-strength breakdown of Table 5.

B. Auxiliary Diagnostics

B.1. Stratified 10-repository subset

Table 6 lists the fixed subset used for every auxiliary diagnostic. It is drawn from the public 22 so that three stratifications hold simultaneously: repository size (3 large / 4 medium / 3 small), mutation-testing gate strength (6 detected / 3 blind / 1 vacuous), and base governance level (4 low / 4 medium / 2 high). The one stratum that cannot be filled is gate strength `none`: every public instance carries a verification patch. The gate-strength stratification in Table 5 therefore uses all 60 instances, where `none` occurs 6 times.

Table 6: Fixed 10-repository auxiliary subset. Base mean is the mean base score over the six dimensions.

Instance	Size	Gate	Base level	Base mean
avbor/HomeAssistantConfig	large	blind	low	1.67
rivitna/Malware	large	detected	medium	1.83
TsingZ0/HtFLlib	large	detected	high	2.17
DreamWall-Animation/dwpicker	medium	vacuous	low	1.67
THUNLP-MT/StreamingBench	medium	detected	medium	1.83
fblissjr/ComfyUI-QwenImageWanBridge	medium	detected	medium	1.83
sleeepeer/PoisonedRAG	medium	blind	medium	1.83
SkyworkAI/Skywork-Skills	small	detected	low	1.67
dama-cyber/magic-distillation	small	detected	low	1.67
jiajun613/Efficient-WAM	small	blind	high	2.17

B.2. No-op drift detail

Table 7 lists every instance on which the empty patch produced a nonzero score. Five of ten instances drift; the median is 0.000 and the standard deviation is 0.073.

Table 7: Instances that drift under the no-op baseline. Dimension changes are base → treated on an identical repository.

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

B.3. Per-instance comparison against the template baselines

Table 8 compares one representative model against all three baselines instance by instance. Qwen3.8-Max loses to at least one template baseline on four of ten repositories, including a negative score on `HtFLlib`. These cases are the natural target for manual inspection: they distinguish model misjudgment from repositories that genuinely suit templated treatment.

Table 8: Per-instance NGI, Qwen3.8-Max versus the three baselines. Bold marks the best of the four on each instance.

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

B.4. Baseline patch contents

The Mechanical baseline 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 baseline 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 baseline invokes a language model, and neither modifies existing source files, so both are guaranteed not to break behavior—which the gate confirms (0 broken out of 10 for both).

B.5. A corrected accounting bug

An earlier version of the pipeline recorded all six `gate=none` instances as `behavior_broken`, because `git apply` against a nonexistent verification patch exits with status 99. 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 reported in Table 5 are post-correction. The public 22-instance subset is unaffected, since every public instance carries a verification patch, so Table 2 required no revision.

B.6. Reproducing the diagnostics

All four diagnostics reuse stored rollouts and add no model calls. The release ships per-instance evidence, scores, and patches for each baseline (`noop/`, `mechanical/`, `rule/`), the machine-readable summary of Table 3, the ablation record backing Table 5, the fixed subset of Table 6, and the scripts that produce each table.