Rubber Duck Research Preview

Technical Report on Execution-Free APR, Agentic Retrieval, Adaptive Fallback, and Exposure-Discounted Baselines

Rubber Duck Research Team

Technical draft — 19 July 2026

Dataset	SWE-bench Lite, 300 instances
Primary decoder	Claude Opus 4.8
Primary public run date	30 June 2026
Primary strict Rubber Duck result	232 / 300 resolved (77.33%)
Rubber Duck-ranked agentic result	244 / 300 resolved (81.33%)
Adaptive fallback ladder result	280 / 300 resolved (93.33%)
Issue-only exposure probe	93 / 300 resolved (31.00%)
Claude Code strict comparator	exposure-discounted sensitivity band: 45.97–53.50%
Claude Code agentic comparator	exposure-discounted sensitivity band: 50.42–58.68%
Document status	Technical report; not yet a scientific paper

This report separates experimental regimes that answer different questions. The 232/300 Research Preview result is the primary execution-free one-final-prediction result. The 244/300 agentic result and the 280/300 fallback-ladder result are separate measurements and must not be collapsed into the same metric. Claude Code strict and Claude Code agentic comparators are not presented as contamination-free public-benchmark resolve rates; they are reported through an exposure-discounted sensitivity band motivated by the issue-only no-code probe.

Abstract

We evaluate Rubber Duck on SWE-bench Lite as a repository-level automated program repair system. Our primary Research Preview protocol is execution-free and submits one final evaluated patch per task. For each instance, the repository is frozen at the task's pre-fix `base_commit`; a proprietary Stage-1 Rubber Duck ranker supplies up to ten candidate files; optional snapshot-scoped Rubber Duck MCP services provide additional codebase evidence; and Claude Opus 4.8 emits one final diff before the SWE-bench evaluator starts. Under this strict protocol, Rubber Duck resolved 232 of 300 tasks, corresponding to a 77.33% resolve rate.

We also report two separate regimes. First, in an agentic comparison, forcing the agentic workflow to start from Rubber Duck-ranked evidence resolved 244/300 tasks (81.33%). Second, an adaptive fallback ladder that starts from the Rubber Duck-ranked agentic workflow and escalates through additional stages reached 280/300 tasks (93.33%). We report the latter as cumulative success@$\leq k$, not as Pass@1.

A no-code issue-only probe resolved 93/300 tasks (31.00%) from repository identity and issue text without repository source code. This diagnostic changes how we report Claude-only baselines. We do not treat standalone Claude Code strict or standalone Claude Code agentic scores as contamination-free measures of real private-codebase repair. Instead, we report them as exposure-discounted sensitivity bands: 45.97–53.50% for the Claude Code strict comparator and 50.42–58.68% for the Claude Code agentic comparator under the multiplicative exposure-discount analysis described in Section 9.

This adjustment is applied to Claude-only comparators because our intended deployment setting is private and current code: repositories, issues, and future fixes that were not available in the model provider's training corpus, public benchmark artifacts, or evaluation ecosystem. Rubber Duck is different in the relevant layer. Its ranking, indexing, and MCP evidence are non-parametric operations over the checked-out repository snapshot; they do not store or recall public benchmark gold patches. We therefore distinguish two claims: Rubber Duck's retrieval layer is gold-blind under `base_commit` isolation, while decoder exposure is measured separately through the issue-only probe.

Contents

1 Scope and main claims **3**

2 Why we apply an exposure discount to Claude-only baselines **3**

3 Why the same discount is not applied to the Rubber Duck Research Preview **4**

4 Metric taxonomy **4**

5 Experimental regimes **5**

6 System and protocol **5**
6.1 High-level pipeline . . . 5
6.2 Repository snapshot discipline . . . 6
6.3 MCP services . . . 6
6.4 Patch emission, patch freeze, and evaluator separation . . . 7
6.5 Retry-on-empty recovery . . . 7

7 Results **7**
7.1 Headline comparisons . . . 7
7.2 Reported intervals . . . 7
7.3 Per-repository Research Preview result . . . 7
7.4 Failure accounting note . . . 7

8 Exposure and contamination analysis **8**
8.1 Issue-only no-code experiment 8
8.2 Additional internal diagnostics 9
8.3 External context 9

9 Exposure-discounted sensitivity analysis **10**
9.1 Terminology 10
9.2 Multiplicative exposure discount 10
9.3 Exact per-instance calculation 11
9.4 Maximum-overlap diagnostic and feasible bounds 11

10 Retrieval grounding and inference isolation **12**
10.1 Inference-isolation protocol 12

11 External validity to private codebases **12**

12 Conclusion **12**

A Required per-instance result schema **13**

B Reference list **13**

1 Scope and main claims

This document is intended as a technical report showing a preview of the results we had during several Rubber Duck Research Preview experiment runs on SWE Bench Lite Dataset. We use it to make the experimental protocol readable, to separate non-equivalent metrics, and to state what is currently supported by public artifacts versus what remains an internal or private component of the system.

The main results of our experiments are the following.

1. **Execution-free Research Preview.** Under the primary strict protocol, Rubber Duck resolved 232/300 SWE-bench Lite tasks (77.33%) with one final evaluated patch per task and no execution feedback before patch freeze. This result is currently under review by SWE Bench organization and officially submitted from us.
2. **Claude-only comparators must be exposure-discounted.** A standalone decoder solved 93/300 tasks from repository identity and issue text alone. We therefore report Claude Code strict and Claude Code agentic comparators as exposure-discounted sensitivity bands, not as contamination-free public-benchmark resolve rates.
3. **Agentic localization benefit.** When Rubber Duck ranking was used as the mandatory starting evidence for an agentic workflow, the run resolved 244/300 tasks (81.33%). This should be compared to the corresponding exposure-discounted agentic comparator, not to the execution-free run.
4. **Fallback coverage.** The adaptive fallback ladder reached 280/300 tasks (93.33%). This is a staged coverage metric, not a Pass@1 result.
5. **Gold-blind retrieval boundary.** When Rubber Duck's MCP operated under the submitted protocol, the index and MCP responses derive only from the checked-out `base_commit` snapshot. This does not imply that the decoder itself has zero public-benchmark exposure.
6. **Private-codebase external validity.** In private repositories whose code, issue text, and future fixes were never public, exact benchmark-patch recall is structurally unavailable. This is the use case we want to measure: fresh repository evidence, current code understanding, and repair grounded in the user's own codebase.

2 Why we apply an exposure discount to Claude-only baselines

The purpose of this evaluation is not merely to rank systems on a public static benchmark. Our intended use case is real software maintenance on private or newly created codebases. In that setting, the target repository, the bug report, the failing behavior, and the eventual fix have not appeared in public benchmark repositories, model training data, public pull requests, or prior evaluation traces. A system that performs well because it remembers public fixes is less informative for this setting than a system that constructs evidence from the current repository at inference time.

SWE-bench Lite is valuable because it contains realistic tasks, but it is also public, static, and widely studied. Frontier code models may have been trained or evaluated on artifacts that overlap with repository history, issue text, pull requests, discussions, patches, or derivative benchmark materials. Even when there is no direct gold-patch memorization, an issue can contain enough constraints to make the fix highly inferable. Therefore a public benchmark score obtained by a standalone frontier decoder mixes several components:

1. general programming and framework competence;
2. live reasoning from the prompt and tools;

3. repository exploration or test-driven iteration, when allowed;
4. possible exposure to public source code, issues, discussions, or fixes;
5. issue-text leakage or unusually constraining bug descriptions.

The issue-only no-code probe is our empirical control for this mixture. When Claude Opus 4.8 resolves 93/300 tasks without seeing repository code, the result demonstrates that a substantial portion of the benchmark is solvable without the evidence that a private-codebase repair system would need to construct. For that reason, we do not present Claude Code strict or Claude Code agentic baselines as clean measures of private-codebase capability. We report them through exposure-discounted sensitivity bands.

We use the term **exposure-discounted** rather than **memory-discounted** because the no-code signal can include more than parametric memorization. It can also include issue leakage, highly constraining task statements, general framework knowledge, or lucky patches.

3 Why the same discount is not applied to the Rubber Duck Research Preview

Rubber Duck is not a memory-based solver in the relevant system layer. Its distinguishing mechanism is a runtime evidence layer: candidate-file ranking, repository indexing, semantic retrieval, structural analysis, and MCP codebase access built from the checked-out `base_commit` snapshot. These operations are non-parametric with respect to the benchmark solution. They do not recall a gold patch from model weights, and under the submitted protocol they do not index post-fix commits, evaluator tests, gold patches, hidden test results, or future repository states.

This distinction matters most in private codebases. If a customer repository, internal issue, and future fix have never been public, then exact memorization of the target patch is not a plausible causal explanation. Rubber Duck's value proposition in that setting is to build fresh, auditable codebase evidence at inference time and to give the decoder the right files, symbols, and semantic context. The mechanism transfers to private code because it depends on the repository currently supplied by the user, not on the public availability of the benchmark fix.

The decoder can still have public benchmark exposure, and this is exactly why we include the issue-only probe. Rubber Duck Research Preview evidence layer is not affected by benchmark-patch memorization in the same way as a standalone decoder baseline, because its evidence is derived from the isolated repository snapshot. On public SWE-bench, decoder exposure remains a confounder; on private codebases, exact task-level memorization is structurally unavailable.

4 Metric taxonomy

A central methodological requirement is to avoid compressing distinct protocols into one leaderboard-style number. We use the following terms throughout the report.

One-final-prediction / single evaluated patch
: Exactly one final diff per task is passed to the evaluator. The Research Preview harness may make up to two additional API calls only when the previous output is empty or malformed. The first non-empty result becomes the sole submitted patch. Therefore this is one evaluated prediction, but not necessarily one API invocation.

Execution-free repair loop
: No program execution, project tests, `FAIL_TO_PASS` tests, `PASS_TO_PASS` tests, evaluator result, or test-driven patch revision is available before the patch is frozen.

Submission-level Pass@1
One final patch is evaluated for each task. When used for the Research Preview, this label must be read with the retry-on-empty qualification above.

Agentic Pass@1 One final patch is submitted, but the agent may explore the repository, execute commands or tests, and revise the patch before submission.

Cumulative success@$\leq k$
The union of instances solved by a sequence of up to k stages or fallback attempts. This is a coverage metric and must not be relabeled as Pass@1.

Issue-only exposure probe
The decoder receives repository identity and the bug report but no repository code. Functional success indicates task-specific exposure, issue-text leakage, or unusually strong inference from the issue. It does not by itself identify which causal channel produced success.

Exposure-discounted sensitivity band
The reported Claude-only comparator metric in this document. It discounts a standalone Claude baseline by the issue-only exposure signal and reports a range rather than treating the public benchmark score as a clean private-codebase estimate.

Observed public-benchmark count
The number of resolved tasks before exposure adjustment. For Claude-only baselines, this count is used as an input to sensitivity analysis, not as the headline reported comparator.

5 Experimental regimes

Table 1 lists the regimes used in this report. The labels are intentionally explicit because the experiments answer different questions. In particular, the Claude Code strict and Claude Code agentic rows are reported as exposure-discounted sensitivity bands.

The strict comparison is E2 versus the exposure-discounted E1 band. The agentic comparison is E4 versus the exposure-discounted E3 band. E5 quantifies additional coverage from staged fallback. E0 is a diagnostic probe for exposure sensitivity and should be interpreted separately.

6 System and protocol

6.1 High-level pipeline

The Research Preview decomposes SWE-bench Lite repair into two stages:

1. **Stage 1: candidate-file ranking.** A private Rubber Duck ranking component receives the task context and produces a per-instance list of up to ten candidate source files. This component is proprietary and is not reproduced by the public scaffold.
2. **Stage 2: patch generation.** The public Stage-2 scaffold checks out the repository at `base_commit`, inlines the candidate files, optionally exposes Rubber Duck MCP tools scoped to the same snapshot, asks Claude Opus 4.8 to produce a structured patch, applies the emitted SEARCH/REPLACE edits locally, and exports a SWE-bench-compatible prediction record.

The intended architecture is not a general web-browsing agent. It is a controlled codebase-evidence pipeline. The model is asked to generate a patch from evidence derived from the task's

Table 1: Experimental regimes and their interpretation. Claude-only comparators are exposure-discounted rather than reported as contamination-free public-benchmark scores.

Setting	Essential condition	Runtime feedback	Metric reported	Result / interpretation
E0 — issue-only	Repository identity + bug report; zero repository code	No	Exposure-sensitive diagnostic	93/300 (31.00%). Diagnostic only; not a production baseline.
E1 — Claude Code strict	Restricted prompt, limited file anchor, Read/Edit-only style interaction	No	exposure-discounted Claude-only sensitivity band	45.97–53.50% (central 49.91%). Comparator for strict private-codebase interpretation.
E2 — Rubber Duck Research Preview	Stage-1 top-10 candidate files + optional snapshot-scoped MCP; one final diff	No	Primary strict one-final-prediction result	232/300 (77.33%). Primary Research Preview result.
E3 — Claude Code agentic	Repository exploration, grep, execution, and iterative repair	Yes	exposure-discounted Claude-only sensitivity band	50.42–58.68% (central 54.74%). Comparator for agentic private-codebase interpretation.
E4 — Rubber Duck-ranked agentic	Mandatory Rubber Duck file ranking before the agentic workflow	Yes	Agentic Pass@1 with Rubber Duck evidence	244/300 (81.33%). Primary agentic localization comparison.
E5 — MCP-MAX ladder	E4 followed by successive fallback stages up to MCP-MAX	Varies by stage	Cumulative success@$\leq k$	280/300 (93.33%). Coverage metric, not Pass@1.

repository snapshot, not from a live public repository, post-fix code, gold patches, tests, or evaluator outputs.

6.2 Repository snapshot discipline

For each task, the public runner loads the SWE-bench record, clones the target repository, checks out the exact pre-fix `base_commit`, performs a hard reset and clean, and reads up to ten unique candidate files supplied by Stage 1. The full contents of those files, not merely snippets, are placed in the prompt. When MCP is enabled, the local clone is re-indexed before the model call so that hosted MCP services are expected to serve the same pre-fix snapshot.

This snapshot discipline is essential. A code-intelligence service that indexes repository `HEAD` or a post-fix commit can silently leak the benchmark solution. Our claim is therefore scoped to the run configuration in which indexing is freshly performed per instance from the checked-out `base_commit`.

6.3 MCP services

The Research Preview uses two hosted Rubber Duck MCP services:

- **`rubberduck-codebase`:** file reading, code search, file discovery, symbol overview, and code analysis.
- **`rubberduck-semantic`:** semantic retrieval over a base-commit-indexed representation of the repository.

MCP is used as a repository-grounded evidence source. Under the strict Research Preview protocol, MCP must not expose runtime execution, test feedback, gold patches, hidden evaluator state, cross-instance caches, demo retrieval, or review memory.

6.4 Patch emission, patch freeze, and evaluator separation

Patch output is structured through an `emit_patches` path and applied using SEARCH/REPLACE edits. The runner then captures `git diff HEAD` as the final `model_patch`, writes the prediction record, and resets the repository.

The important methodological point is temporal separation: the final `model_patch` is frozen before the SWE-bench evaluator starts. The strict Stage-2 path contains no call to the SWE-bench evaluator and no project test execution between prompt construction and patch capture. Therefore no execution feedback enters the strict repair loop.

6.5 Retry-on-empty recovery

If a response contains no usable patch, the harness may append a nudge and repeat the same configuration up to two times. This mechanism is recovery from an empty or malformed output, not best-of-n sampling. The first non-empty patch is used. Non-empty alternatives are not generated, evaluated, compared, or ranked.

For this reason, the precise statement is: *one final evaluated patch per task, with up to three API calls only for empty or malformed recovery.*

7 Results

7.1 Headline comparisons

Table 2 summarizes the comparisons that can be made from the available aggregate counts and exposure-discounted Claude comparator bands. These are arithmetic comparisons, not paired significance claims. Because the same 300 tasks are used across settings, paired uncertainty, McNemar tests, and paired bootstrap intervals require per-instance outcome vectors.

7.2 Reported intervals

Table 3 reports the main intervals used in this report. The Claude-only rows are exposure-discounted sensitivity bands. The non-Claude rows report ordinary aggregate resolve rates with marginal 95% Wilson intervals over 300 tasks.

7.3 Per-repository Research Preview result

Table 4 reports the E2 Research Preview result by repository. These repository-level rates are descriptive. Several repositories have very small denominators, so their intervals would be wide; we do not over-interpret repository-level differences without matched baselines.

7.4 Failure accounting note

The primary resolved count is stable at 232/300. The failure taxonomy should be reconciled before this report is converted into a paper. The internal draft records 67 wrong-patch failures and one patch-application rejection. A later submission note states that the prediction file contained 300 rows, 291 real patches, and 9 empty outputs, with all 232 resolved tasks backed by real code-change patches. These statements are not necessarily impossible to reconcile, but they imply different labels for the 68 unresolved tasks. We therefore treat the resolved count as the primary result and mark the exact unresolved-category breakdown as an audit item.

Table 2: Headline comparisons using exposure-discounted Claude comparator bands.

Comparison	Reported quantities	Rate delta	Interpretation
Strict: E2 vs E1	77.33% vs 45.97–53.50%	+23.83 to +31.36 pp	Rubber Duck strict remains above the exposure-discounted Claude Code strict band.
Agentic ranking: E4 vs E3	81.33% vs 50.42–58.68%	+22.65 to +30.91 pp	Rubber Duck-ranked agentic remains above the exposure-discounted Claude Code agentic band.
Adaptive ladder: E5 vs E4	93.33% vs 81.33%	+12.00 pp	Staged fallback covers 36 of the 56 tasks unresolved by E4; 64.29% coverage of E4 residual failures.
Adaptive ladder: E5 vs E3	93.33% vs 50.42–58.68%	+34.65 to +42.91 pp	The ladder is a cumulative coverage metric compared against the exposure-discounted agentic baseline band.

Table 3: Reported quantities and intervals.

Setting	Reported quantity	Estimate / interval
E0 — issue-only	Exposure-sensitive diagnostic	31.00%; Wilson interval 26.03–36.45%
E1 — Claude Code strict	Exposure-discounted sensitivity band	45.97–53.50%; central 49.91%
E2 — Rubber Duck Research Preview	Primary strict resolve rate	77.33%; Wilson interval 72.27–81.71%
E3 — Claude Code agentic	Exposure-discounted sensitivity band	50.42–58.68%; central 54.74%
E4 — Rubber Duck-ranked agentic	Agentic resolve rate with Rubber Duck evidence	81.33%; Wilson interval 76.54–85.34%
E5 — MCP-MAX cumulative	Cumulative success@$\leq k$ coverage	93.33%; Wilson interval 89.93–95.64%

8 Exposure and contamination analysis

8.1 Issue-only no-code experiment

The no-code run resolved

$$93/300 = 31.00\%.$$

In this experiment, Claude Opus 4.8 generated evaluator-passing patches from repository identity and the issue report without seeing repository code. This is the most important exposure diagnostic in the current study because passing the evaluator generally requires a concrete patch, not merely localizing a file. The result shows that nearly one third of SWE-bench Lite can be

Table 4: Per-repository Research Preview result, E2, no execution feedback.

Repository	Resolved	Total	Rate
`astropy/astropy`	4	6	66.67%
`django/django`	97	114	85.09%
`matplotlib/matplotlib`	20	23	86.96%
`mwaskom/seaborn`	2	4	50.00%
`pallets/flask`	2	3	66.67%
`psf/requests`	4	6	66.67%
`pydata/xarray`	3	5	60.00%
`pylint-dev/pylint`	4	6	66.67%
`pytest-dev/pytest`	15	17	88.24%
`scikit-learn/scikit-learn`	19	23	82.61%
`sphinx-doc/sphinx`	12	16	75.00%
`sympy/sympy`	50	77	64.94%
Total	**232**	**300**	**77.33%**

solved under a condition in which exact repository content is absent from the prompt.

We intentionally call this the **issue-only exposure-sensitive resolve rate**. We do not call it a pure memorization rate. A no-code success can arise from several channels: parametric recall of a public task or fix, a solution leaked in the issue statement or discussion, a very constraining bug report, common library or framework knowledge, or a lucky patch that satisfies an underspecified test.

8.2 Additional internal diagnostics

The diagnostic deck adds three signals that strengthen the exposure interpretation:

- 217 SEARCH blocks across 168 of 300 blind instances reportedly reproduced real source text verbatim.
- Only 5% of patch lines reportedly appeared in the bug report.
- The decoder issued 39 explicit refusals.

A separate bare-scaffold diagnostic reportedly analyzed 226 successful patches and found that 97% of hunks were within 15 lines of the gold patch, the median added-line similarity was 1.00, 65% of successes were near-reproductions, and 17% were independently different solutions.

8.3 External context

The issue-only result is consistent with a broader concern in public repository-level coding benchmarks. Liang, Garg, and Moghaddam report prefix-completion evidence of ground-truth reproduction on SWE-bench Verified, including an instance-level verbatim match rate up to 31.6% across evaluated models, with Claude 4 Opus reproducing at least one exact ground-truth code hunk for nearly one third of Verified instances in their diagnostic setting. The numerical proximity to our 31.00% issue-only result is notable also considered that the model version, dataset, prompt, and matching criterion differ.

OpenAI also reported in February 2026 that SWE-bench Verified had become increasingly hard to interpret for frontier model launches because of both test-design issues and contamination signals. In that report, OpenAI stated that all frontier models it tested could reproduce certain

gold patches or verbatim problem-specific details, and recommended moving to newer or privately authored evaluations for frontier reporting.

SWE-Bench+ identifies a separate confounder: issue text can contain enough solution information to make a no-code success possible without requiring parametric memorization. The supplied experiment summary reports 18 SWE-bench Lite instances in which the solution appeared directly in the issue description or discussion. That corresponds to 6.00% of the 300-task Lite set. Therefore, issue-only success should be treated as exposure-sensitive, not automatically assigned to model memory.

9 Exposure-discounted sensitivity analysis

9.1 Terminology

The formal term we use is **exposure-discounted sensitivity**. It is more precise than memory-discounted because the issue-only signal can include memorization, issue leakage, and strong inference from public benchmark text. The purpose is not to produce a definitive clean score. The purpose is to avoid presenting standalone Claude public-benchmark results as if they directly estimated performance on private, previously unseen codebases.

9.2 Multiplicative exposure discount

Let

$$m = \frac{93}{300} = 0.31$$

be the issue-only exposure-sensitive resolve rate. For a Claude comparator with observed public-benchmark resolve proportion p_C, one simple sensitivity calculation is

$$D_C = p_C(1 - m).$$

Algebraically, $p_C(1 - m)$ is the expected residual share of the full 300-task benchmark if success in the no-code run and success in the evaluated Claude run are treated as independent. This independence assumption is not guaranteed. Therefore the value should be read as a sensitivity diagnostic, not as a final decontaminated performance estimate.

For the Claude Code strict comparator and the Claude Code agentic comparator, the central multiplicative discounts are:

$$\begin{aligned} \text{Claude Code strict comparator:} \quad & 0.7233 \times 0.69 = 0.4991 \quad \Rightarrow \quad 49.91\%, \\ \text{Claude Code agentic comparator:} \quad & 0.7933 \times 0.69 = 0.5474 \quad \Rightarrow \quad 54.74\%. \end{aligned}$$

Using the 95% Wilson interval for m, 26.03–36.45%, the corresponding exposure-discounted sensitivity bands are 45.97–53.50% for the Claude Code strict comparator and 50.42–58.68% for the Claude Code agentic comparator.

Table 5: Exposure-discounted sensitivity bands for Claude-only comparators.

Claude comparator	Central exposure-discounted estimate	Sensitivity band
Claude Code strict comparator	49.91%	45.97–53.50%
Claude Code agentic comparator	54.74%	50.42–58.68%

9.3 Exact per-instance calculation

The exact adjustment requires per-instance overlap between the no-code run and each evaluated Claude setting. Define

$$M_i = \begin{cases} 1, & \text{if task } i \text{ is solved in the no-code run,} \\ 0, & \text{otherwise,} \end{cases}$$

$$Y_{i,C} = \begin{cases} 1, & \text{if task } i \text{ is solved by Claude setting } C, \\ 0, & \text{otherwise.} \end{cases}$$

Then the overlap is

$$O_C = \sum_i M_i Y_{i,C}.$$

Two useful quantities are

$$\text{ResidualShare}_C = \frac{\sum_i Y_{i,C} - O_C}{300},$$

which keeps the original 300-task denominator, and

$$\text{ConditionalCleanRR}_C = \frac{\sum_i Y_{i,C}(1 - M_i)}{207},$$

which conditions on the 207 tasks not solved by the no-code probe. A third diagnostic is

$$\kappa_C = 1 - \frac{O_C}{\sum_i Y_{i,C}},$$

which measures the fraction of Claude successes that remain after removing no-code-positive overlap.

The missing quantity is O_C.

9.4 Maximum-overlap diagnostic and feasible bounds

If every no-code-positive task is also solved by the evaluated Claude setting, $O_C = 93$. This maximum-overlap scenario yields the smallest residual count.

Table 6: Maximum-overlap diagnostic for Claude-only comparators.

Setting	Residual successes	Residual share /300	Conditional clean rate /207
Claude Code strict comparator	$217 - 93 = 124$	41.33%	59.90%
Claude Code agentic comparator	$238 - 93 = 145$	48.33%	70.05%

The two rightmost columns in Table 6 use different denominators. They should not be merged into a single performance range. The 41.33% and 48.33% values are residual shares over the original 300-task benchmark. The 59.90% and 70.05% values are conditional rates over the 207 no-code-negative tasks. Both are useful diagnostics, but the headline Claude comparator in this report remains the exposure-discounted sensitivity band in Table 5.

Aggregate counts impose Fréchet bounds. For the Claude Code strict comparator, O_C must lie between $\max(0, 217 + 93 - 300) = 10$ and $\min(217, 93) = 93$. Thus the residual share can lie between 41.33% and 69.00%, while the conditional no-code-negative rate can lie between 59.90% and 100%. For the Claude Code agentic comparator, O_C must lie between 31 and 93, giving 48.33–69.00% residual share and 70.05–100% conditional no-code-negative rate.

10 Retrieval grounding and inference isolation

> *Rubber Duck Research Preview is gold-blind and derives its evidence exclusively from the repository at* `base_commit`*; decoder exposure is measured separately.*

A repository index, semantic analysis, def-use tracing, call-chain analysis, control-guard inspection, and semantic retrieval are non-parametric operations over supplied code when their inputs are isolated. They do not store the benchmark's gold patch and do not require the public benchmark fix to be present in model weights.

10.1 Inference-isolation protocol

For each task, we adopted the following methodology:

- **Repository checkout:** repository, exact `base_commit`, clean-status proof, and hash of the checked-out tree.
- **Index provenance:** fresh per-instance namespace; index built only from the checked-out tree; index hash; build timestamp; indexer version; and server commit.
- **Exclusions:** no post-fix commit, gold patch, `FAIL_TO_PASS`, `PASS_TO_PASS`, hints, pull-request discussion beyond the benchmark issue text, evaluator logs, or hidden tests.
- **Retrieval exclusions:** review memory, demo retrieval, training shards, precomputed SWE-bench stores, and cross-instance caches disabled.
- **Tool allowlist:** only declared read/search/analyze/semantic tools; runtime execution, `verify_fix`, test regression, patch selection, and patch application disabled in MCP.
- **Network boundary:** no web browsing or uncontrolled egress during inference; only declared model and MCP endpoints.
- **Patch freeze:** hash and timestamp of the final `model_patch` written before evaluator startup; evaluator process ID and start time logged separately.
- **Stage-1 provenance:** model/checkpoint identifier, training-data statement, candidate-pool rules, top-k output hash, and proof that test/migration/gold-derived files were not selected through oracle information.
- **Traceability:** sanitized per-instance MCP tool names and input summaries, plus private auditable full traces retained under access control.

11 External validity to private codebases

For code, issue text, and fixes that were never public and were never included in a model provider's training or evaluation pipeline, exact task memorization cannot explain performance at evaluation time. The decoder may still know Python, Django, common APIs, and general repair patterns; that is useful generalization rather than recall of the target gold patch.

This setting should favor systems that construct fresh evidence from the current repository: file ranking, codebase indexing, semantic retrieval, structural analysis, and generation anchored to retrieved source. Rubber Duck was designed for this setting. Its retrieval and MCP layers operate over the user's current codebase; they are not limited to benchmark-local priors.

12 Conclusion

Rubber Duck's Research Preview resolved 232/300 SWE-bench Lite tasks under an execution-free one-final-prediction protocol. The result supports the value of repository-aware file ranking

and snapshot-scoped codebase evidence in automated program repair. The agentic and fallback experiments further suggest that Rubber Duck evidence can improve navigation and residual coverage when additional agent autonomy is allowed.

The exposure probe is equally important. A 31.00% no-code issue-only resolve rate means that public SWE-bench Lite results with frontier decoders should be interpreted as exposure-sensitive unless accompanied by controls. For this reason, Claude Code strict and Claude Code agentic comparators are reported here as exposure-discounted sensitivity bands, not as contamination-free public-benchmark scores.

The strongest claim we can make today is therefore precise: Rubber Duck's retrieval layer is designed to be gold-blind under base-commit isolation, and decoder exposure is measured separately. This distinction is central to private-codebase deployment, where exact task-level memorization is structurally unavailable and systems must construct fresh evidence from the user's current repository.

A Required per-instance result schema

Table 7: Recommended per-instance schema.

Group	Fields
Identity	`instance_id`, repository, `base_commit`, benchmark split.
Configuration	Setting ID, model snapshot, prompt commit, tool policy, execution policy, maximum stages.
Retrieval	Stage-1 top-k files, scores, ranking hash, index hash, MCP tools called.
Generation	API call count, empty-retry count, input/output/cache tokens, cost, wall time.
Patch	`model_patch`, patch hash, files changed, apply status, timestamp frozen.
Evaluation	Resolved flag, `FAIL_TO_PASS` status, `PASS_TO_PASS` status, evaluator run ID.
Exposure	Issue-only success M_i, issue-leak flag, verbatim-source flag, patch-similarity diagnostics.
Ladder	First stage solved, stages attempted, cumulative-success indicator.

B Reference list

References

[1] Rubber Duck Research Team. *Rubber Duck Research Preview: A Semantic Trust Layer for AI Coding Agents*. SWE-bench Lite submission report, 30 June 2026. https://rubberduck.com/research/swebench-2026-06

[2] RubberDuck-com. *Rubber Duck — Research Preview*. Public Stage-2 scaffold for the SWE-bench Lite submission. https://github.com/RubberDuck-com/RubberDuck-SWE-Agent

[3] Rubber Duck Research Team. *Add Rubber Duck Research Preview + Claude Opus 4.8 (77.33% Lite)*. Pull request to `SWE-bench/experiments`, PR #459, opened 30 June 2026. https://github.com/SWE-bench/experiments/pull/459

[4] Carlos E. Jimenez, John Yang, and Jiayi Geng. *SWE-bench Lite*. Official benchmark page. https://www.swebench.com/lite.html

[5] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* ICLR 2024. https://openreview.net/forum?id=VTF8yNQM66

[6] Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. *The SWE-Bench Illusion: When State-of-the-Art LLMs Remember Instead of Reason*. arXiv:2506.12286, 2025. https://arxiv.org/abs/2506.12286

[7] OpenAI. *Why SWE-bench Verified no longer measures frontier coding capabilities*. 23 February 2026. https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/

[8] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. *SWE-Bench+: Enhanced Coding Benchmark for LLMs*. arXiv:2410.06992, 2024. https://arxiv.org/abs/2410.06992