Budget-Bounded Software Repair with AI Agents

Termination, tree-bound acceptance, and benchmark validity in the Gemma 4 challenge

Author: Aridio Silva — Independent Researcher, Brazil
Preferred citation: Silva, Aridio (2026) · **ORCID:** 0009-0008-2411-6995
Copyright: © 2026 Aridio Silva
DOI: https://doi.org/10.34740/KAGGLE/W/125650

Abstract

We propose a budget-bounded repair controller with tree-bound acceptance. The abstract controller requires a separate validator's evidence for the exact frozen tree before delivery, preventing hallucinated claims of test success from authorizing release. Under explicit supervision assumptions, the abstract controller terminates within 4M + 1 transitions for an attempt budget of M. A finite-state check explored 556 states and 953 transitions and found a path to acceptance when the model allowed false evidence. An audit of the public benchmark identified one task with incompatible mandatory test requirements and three false acceptances in nine synthetic test-identity cases. On Kaggle, 109 executions covered 86 of 87 selected bug tasks across seven rounds. The verifier recorded 37 resolutions (43.0% of executed tasks and 42.5% of selected tasks), including 20 of 54 in S54M, the batch run under a common configuration.

1 Introduction

A repair agent examines a repository and proposes a diff. The controller then budgets attempts, supervises validation, and ties authorization to the code tree examined by the validator. We apply principles from the SGAEIA research program [1] to software repair with Gemma 4 [2].

We make three contributions.

1. **Termination and tree-bound acceptance.** We prove a transition bound and an invariant linking acceptance to the exact validated tree. A finite-state check produces a counterexample when the controller allows false evidence.
2. **Benchmark validity.** We document a task whose mandatory tests are jointly unsatisfiable under the fixed public profile and show how a test-identity comparator can accept results for the wrong test.
3. **Evaluation on public development tasks.** We report 37 resolutions across 86 executed bug tasks in three repositories, with a separate result for 54 tasks run under a common configuration and an analysis of verifier outcomes.

2 Related work

Generate-and-validate repair systems such as GenProg use tests to screen candidate patches [15]. Qi et al. distinguish test-suite success from broader patch correctness [4]. SWE-agent provides an interface for repository and tool interactions [16]. Agentless stages localization, repair, and validation [17], and AutoCodeRover uses program structure to guide code search [18]. Our controller formalizes the release decision for a budgeted repair workflow by binding validator evidence to the delivered tree.

Certifying algorithms accompany results with checkable evidence [5], SWE-Proof studies machine-checked repair contracts [6], and in-toto links artifacts to software supply-chain steps [20]. The property here governs when a controller may deliver a repair. SWE-bench evaluates repair on

real repository issues [3], and SWE-bench Verified screens tasks for evaluation quality [19]. For the Gemma 4 public development profile, we examine whether mandatory tests can pass together and whether the verifier matches the required test identities.

3 Repair controller

The evaluation profile fixes the task, base commit, allowed paths, dependencies, and acceptance criterion. In the controller design, each attempt debits the budget before invoking the generator. The supervisor applies the diff in an isolated copy and freezes the resulting tree. A separate validator records its verdict and the identities of the task, profile, attempt, and tree. On the test path, the acceptance criterion requires a successful test command and a passing result for each required parameterized test node. Failures elsewhere in the verifier's suite can also prevent acceptance. On the formal path, the validator checks evidence for an explicit contract.

To prevent a hallucinated claim of test success from authorizing delivery, the supervisor takes the acceptance evidence from the separate validator. It checks that the evidence names the active attempt and frozen tree, then confirms delivery of that same tree within the authorized deadline. Section 4 proves this property for the abstract controller. Section 5 documents the patch application, verifier verdicts, and test evidence recorded in the Kaggle runs.

An attempt may end without a patch, after an application failure, with rejected or unavailable evidence, or without timely delivery confirmation. In these cases, the supervisor discards the candidate, revokes any active authorization, and rechecks the budget. If an attempt and sufficient reserve remain, the controller debits another attempt and returns to generation. Otherwise, it records ABSTENTION.

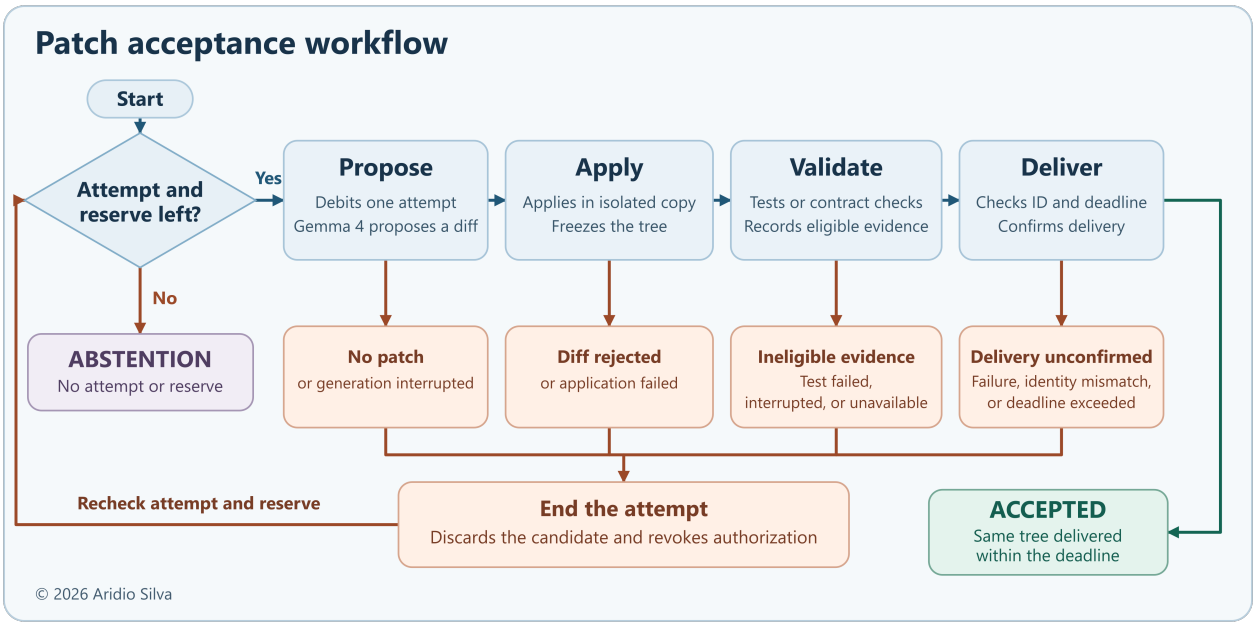


Figure 1. Abstract repair controller workflow. An attempt that ends without acceptance returns to the attempt and reserve check. ACCEPTED requires delivery of the same tree within the deadline. S54M used an operational budget of one session per task (Section 5).

Mapping to the S54M executor

In S54M, the executor evaluated patches against hash-checked snapshots, recorded the patch hashes, and separately checked the identities of the required JUnit test nodes. The measured outcome was the verifier's verdict. The same-tree delivery guarantee proved in Section 4 applies to the abstract controller.

4 Termination and tree-bound acceptance

The control state is (r, b, q), with r attempts remaining, abstract reserve b, and phase q. The nonterminal phases are ready, propose, verify, and publish. The initial state is ready, with r = M. In the checked instance, starting requires b ≥ 4 to cover the four debits of a full attempt. An attempt consumes one unit of r before calling the generator. The supervisor regains control when each operation completes or is interrupted and takes a permitted next step in finite time.

The ranking function measures the remaining transition budget. Each unstarted attempt contributes four transitions. The phase weight counts how many transitions may remain in the current attempt. Starting consumes one transition and leaves up to three more, for the patch outcome, the validation outcome, and the delivery outcome. An early failure returns control to ready sooner.

Define R(r, q) = 4r + w(q), where w is 0 in ready, 3 in propose, 2 in verify, and 1 in publish. Starting an attempt spends one unit of r and increases the phase weight from 0 to 3, so R changes by 4(−1) + 3 = −1. Advancing to the next nonterminal phase decreases w by one. Returning to ready after a failure sets w to zero without restoring r. Thus, every transition between nonterminal states decreases R by at least one.

R starts at 4M and is nonnegative, so there can be at most 4M transitions between nonterminal states and one final transition. With initial reserve of at least 4M, the worst-case path uses all M attempts. Each attempt takes four transitions, fails at delivery, and returns to ready. The final transition records ABSTENTION, attaining the 4M + 1 bound. A wall-clock bound also requires bounds on operation durations, including supervision, cancellation, and cleanup.

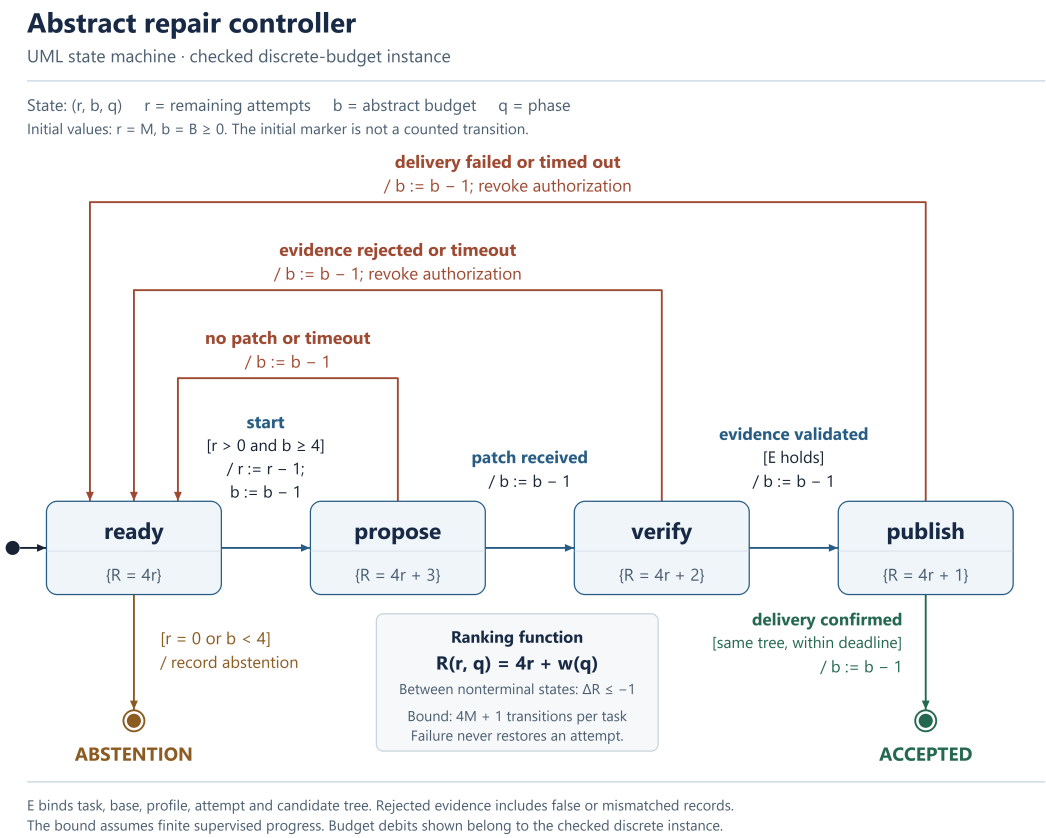


Figure 2. Discrete controller model. Verify covers application and validation. E denotes eligible evidence bound to the candidate tree.

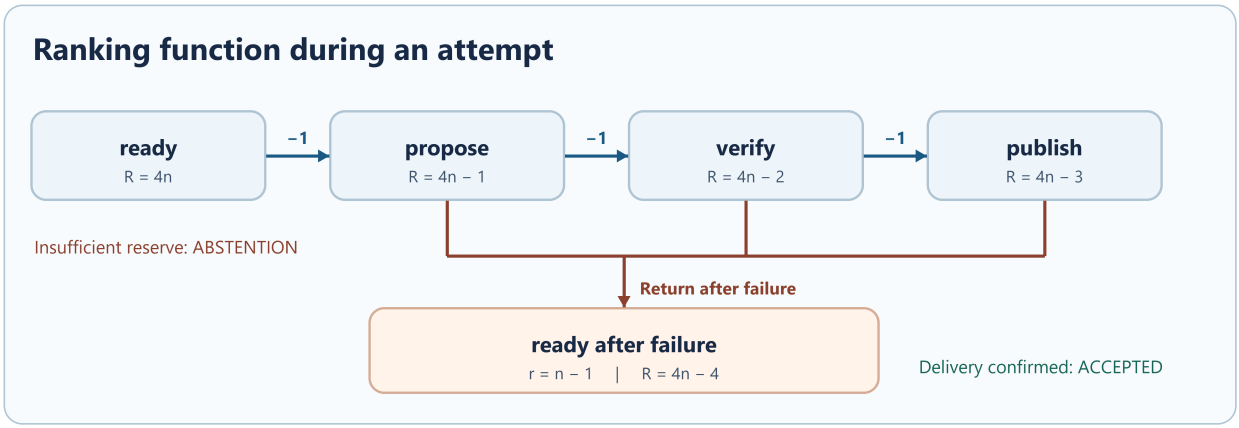


Figure 3. The ranking function decreases as the controller advances through phases and when it returns after failure. Here n is the value of r when the attempt starts.

The tree-bound acceptance invariant states that every transition into ACCEPTED confirms timely delivery of a frozen tree backed by eligible evidence from the authorized validator for the active attempt. The transition from verify to publish checks that evidence and fixes the tree identity. The transition from publish to ACCEPTED checks timely delivery of that same tree. A failure revokes authorization before another attempt can begin. These rules preserve the invariant by induction over the transitions.

On the contract path, the guarantee depends on verifier soundness. On the test path, acceptance attests to the suite outcome under the recorded profile. If the required evidence is unobtainable in that profile, the only reachable terminal state is ABSTENTION.

5 Evaluation

Model checking and pilot

A breadth-first search in Python 3.12 started from 143 initial combinations of r from 0 to 10 and b from 0 to 12. It explored 556 monitored states and 953 transitions, checking the budget, successor states, ranking decrease, and the evidence condition in ACCEPTED. When we allowed a false-evidence event to enter publish, the search found a path to ACCEPTED without eligible evidence. The model checker covers this finite instance [13]. The ranking proof covers arbitrary finite M under the stated supervision assumptions.

Policy P instructs the agent to map requirements to validation obligations, prefer a strict subset of obligations among otherwise comparable plans, and invalidate evidence affected by edits. In the synthetic pagination pilot, the workflow produced and applied a patch that passed both tests [8]. The Kaggle evaluation used P throughout R1–R6 and S54M [10].

Task selection and round design

We classified the 129 public tasks [7] by the predominant work described in each task. The study catalog comprises 87 bugs, 21 features, 10 refactoring or maintenance tasks, five mixed tasks, three documentation tasks, two testing or automation tasks, and one performance task. We selected bugs because they formed the largest group, at 67.4% of the catalog. Sequential batches let us investigate interruptions and adjust the harness within the available quota. The initial six rounds, R1–R6, covered 33 cataloged bugs, of which 32 were executed. S54M contained the other 54 task IDs, with no overlap [10].

In R1–R6, we reran 16 unresolved tasks while adjusting the executor. The rounds used limits of 24, 30, or 35 requests. The common S54M configuration used the gemma-4-31b-it-qat-w4a16-ct checkpoint (Kaggle revision 2), vLLM 0.19.1, temperature 0.2, and a 32,768-token context on four

NVIDIA L4 GPUs. It allowed up to 30 requests in one agent session per task, with no automatic retry [10]. The retry transitions are exercised in the discrete controller model [13].

In S54M, task-worker time through cleanup had a median of 317.5 s and a maximum of 869.9 s. The full cell ran for 5 h 34 min 42 s, averaging 6 min 12 s per task across 54 tasks [10]. These development timings include patch validation. The prediction track excludes patch validation from its 12-hour agent budget [14].

Each task counted once, using its last recorded outcome in R1–R6 or its sole execution in S54M. The 109 executions include 23 reruns. The executor used the verifier in swegemma, the software package that runs the benchmark's patch evaluation, with the public development profile. For the 37 resolutions, the frozen extracts record an applied patch, agreement across recorded verifier outcomes, and successful test execution [10]. The underlying run receipts also record confirmation of required JUnit identities by a separate executor check. For the failure analysis, we matched required IDs from the test diff against recorded pytest failures and errors using the collection root [10].

Resolution and observed failures

Scope	Selected	Executed	Resolved	% of executed
Total by task	**87**	**86**	**37**	**43.0%**
R1–R6, last execution	33	32	17	53.1%
S54M, common configuration	54	54	20	37.0%

Across FastAPI, Rich, and Requests, the verifier produced usable verdicts for 81 of the 86 executed tasks. The 37 resolutions represent 43.0% of executed tasks and 42.5% of selected tasks, including the deferred task in the latter denominator. The nominal 95% Wilson interval for 37/86, 33.1% to 53.6%, provides a descriptive reference for the observed set. FastAPI had 19 resolutions in 36 executed tasks, Rich 12 in 39, and Requests 6 in 11 [10].

The 86 tasks accounted for 109 executions: 55 records for 32 tasks in R1–R6 and 54 records for 54 tasks in S54M. In order, R1 through R6 recorded 0/2, 0/1, 1/6, 13/26, 3/15, and 0/5 resolutions per execution. Of the 17 resolutions in R1–R6, 14 occurred on the first recorded execution and three in R5, when the tasks were rerun after changes to the executor. Those three are included in the study's 37 resolutions. S54M yielded 20 resolutions under a common configuration. Task selection by batch, configuration changes, and selective reruns of 16 tasks limit comparisons of performance across rounds [10].

Of the 49 unresolved tasks, 44 produced applied patches that were rejected, 14 in R1–R6 and 30 in S54M. A required test failed in 36 cases, 13 and 23, respectively. Seven S54M cases had identified failures outside the required IDs. One historical rejection had insufficient test identity information for classification. Of the five tasks without a usable verdict, one ended without a patch in R1–R6. In S54M, the memory guard interrupted two tests and the batch continued. After patch application in another case, a syntax error blocked import and the verifier returned evaluation error code −1. Truncated output left the patch empty in the remaining case [10].

Benchmark validity

Test identity. In nine synthetic JUnit cases, the swegemma 0.2.11 comparator falsely accepted three mismatches involving the module, function name, or parameter. A strict comparator rejected those three in the same local experiment. The frozen JUnit receipt records each required identity and both comparator outcomes [12].

Profile satisfiability. Task `fastapi_14186` remained in the inventory as deferred. The fixed public profile requires `test_model_description_escaped_with_formfeed`, enabled only with Pydantic v1, and `test_model_field_default_required`, enabled only with v2. Both cannot pass in the same fixed environment. A local diagnostic with the reference patch recorded 52 cases and three skips under

Pydantic 2.13.4. Pytest exited with code zero, and the verifier rejected the result because a mandatory test was skipped. The frozen diagnostic and verdict document the test conditions and outcomes [7, 9].

Operational robustness

In a local reproduction, three malformed arrays exceeded the five-second limit in the vLLM 0.19.1 parser. The compatibility layer returned an explicit error for each, and a correctly delimited control returned under both versions. The frozen parser receipt records those cases [11]. In S54M, the memory guard stopped two test process trees and the batch completed, as recorded in the frozen operational receipts [10]. Appendix A links selected platform and evaluation risks to their frozen evidence.

6 Discussion and conclusion

The controller and benchmark audit yield two design rules for agentic software repair. First, bind the release decision to evidence from an authorized validator for the exact tree being delivered. This keeps a hallucinated claim of test success from authorizing release. Second, check that all mandatory tests can pass under the configured profile and that the verifier matches their full identities. The deferred task and the JUnit experiment show how each check changes what a benchmark verdict can establish.

The Gemma 4 runs recorded 37 resolutions in 86 executed bug tasks, including 20 in the 54-task S54M batch under a common configuration. The controller's termination and acceptance results apply to repair workflows that can enforce the same attempt budget, supervision, and evidence checks.

A matched baseline on the 54 S54M tasks, using the same model, budget, and verifier, would allow a paired estimate of the effect of policy P on task resolution. We are extending the evaluation to the remaining task types in the 129-task public development set. As these runs are completed and verified, we will report the results by task type in a subsequent revision.

Traceability. The frozen supplement contains the inventory, 109 execution records, two memory-guard events, and fingerprints of the seven source ZIPs. The verification script checks the extracts and recomputes the task totals, resolution rates, and required-test failure counts. Archive and member hashes link the extracts to the retained original records [10].

References

[1] Silva, Aridio. SGAEIA, a research program and open reference architecture. Project website.

[2] Gemma Team (2026). Gemma 4 Technical Report.

[3] Jimenez, C. E. et al. (2024). SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. ICLR.

[4] Qi, Z. et al. (2015). An Analysis of Patch Plausibility and Correctness for Generate-And-Validate Patch Generation Systems. ISSTA.

[5] McConnell, R. M. et al. (2011). Certifying Algorithms. Computer Science Review.

[6] Ma, G. et al. (2026). SWE-Proof: Can Language Models Resolve Real-World Issues with Machine-Checked Proofs?.

[7] Kaggle (2026). Gemma 4 Developer Agent Competition data. SHA-256 of tasks.jsonl e4b3fd60f69dbc2b9213e54eeb9636db78aefe92c1d06269d73d9f5f8f3c8ad6.

[8] Silva, Aridio (2026). Synthetic pagination pilot, receipt extracts. Frozen extract, November 7.

[9] Silva, Aridio (2026). fastapi_14186, public profile and verdict. Test and evaluation extracts, November 7.

[10] Silva, Aridio (2026). Frozen supplement for Kaggle executions through November 11. GitHub, paper-evidence-v1.2.0. S54M ZIP SHA-256 d9b1145ba1c78582797d29a0d1bd16466c8e9e1f0b2170d635374693d0aa3d60.

[11] Silva, Aridio (2026). vLLM 0.19.1 malformed-array and compatibility-layer test receipt. November 4.

[12] Silva, Aridio (2026). Synthetic JUnit identity test receipt. November 2.

[13] Silva, Aridio (2026). Finite-state repair controller checker. Frozen evidence release.

[14] Kaggle (2026). Gemma 4 Developer Agent Competition evaluation and timing rules.

[15] Le Goues, C., Nguyen, T., Forrest, S., and Weimer, W. (2012). GenProg: A Generic Method for Automated Software Repair. IEEE Transactions on Software Engineering.

[16] Yang, J. et al. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering.

[17] Xia, C. S., Deng, Y., Dunn, S., and Zhang, L. (2024). Agentless: Demystifying LLM-based Software Engineering Agents.

[18] Zhang, Y. et al. (2024). AutoCodeRover: Autonomous Program Improvement.

[19] OpenAI (2024). Introducing SWE-bench Verified.

[20] Torres-Arias, S. et al. (2019). in-toto: Providing farm-to-table guarantees for bits and bytes. USENIX Security.

[21] Silva, Aridio (2026). R01–R72 catalog of platform and environment risks in the Gemma 4 challenge. English edition. Frozen release v1.2.2.

Appendix A Platform and evaluation risks

The R01–R72 report [21] catalogs risks involving platform operations, dependencies, evaluation, and project decisions. The cases below link to their frozen evidence.

R65, parser non-progress. Three malformed arrays exceeded a five-second limit in vLLM 0.19.1. The compatibility layer returned explicit errors. Parser receipt

R69, network-sensitive tests. Two tasks passed local reference-patch rehearsals with network access and were later resolved on Kaggle [10]. Strict network isolation remains untested. Local checks

R70, incompatible mandatory tests. The fixed public profile for `fastapi_14186` requires tests gated on opposing Pydantic versions. Task diagnostic

Related checks. The JUnit receipt records three synthetic false acceptances. The S54M operational receipts record two memory-guard interruptions.

Project Links

Public evidence repository

Frozen evidence for reference [10], with records from all seven Kaggle evaluation rounds and scripts to reproduce the reported counts.

https://github.com/aridiosilva/Gemma4_Google_Deepmind_S1/blob/433e7508f713d947b417d8522bc84b96eae1da45/evidence/kaggle-bugs-2026-11-11/README.md

License

This Writeup has been released under the Attribution 4.0 International (CC BY 4.0) license.

https://creativecommons.org/licenses/by/4.0/

Citation

DOI: https://doi.org/10.34740/KAGGLE/W/125650

Silva, Aridio (2026). *Budget-Bounded Software Repair with AI Agents.* Kaggle.

BibTeX (Kaggle record)

```
@article{aridio_silva_2026,
  title={Budget-Bounded Software Repair with AI Agents},
  url={https://www.kaggle.com/w/125650},
  DOI={10.34740/KAGGLE/W/125650},
  journal={Kaggle},
  publisher={Kaggle},
  author={Aridio Silva},
  year={2026}
}
```

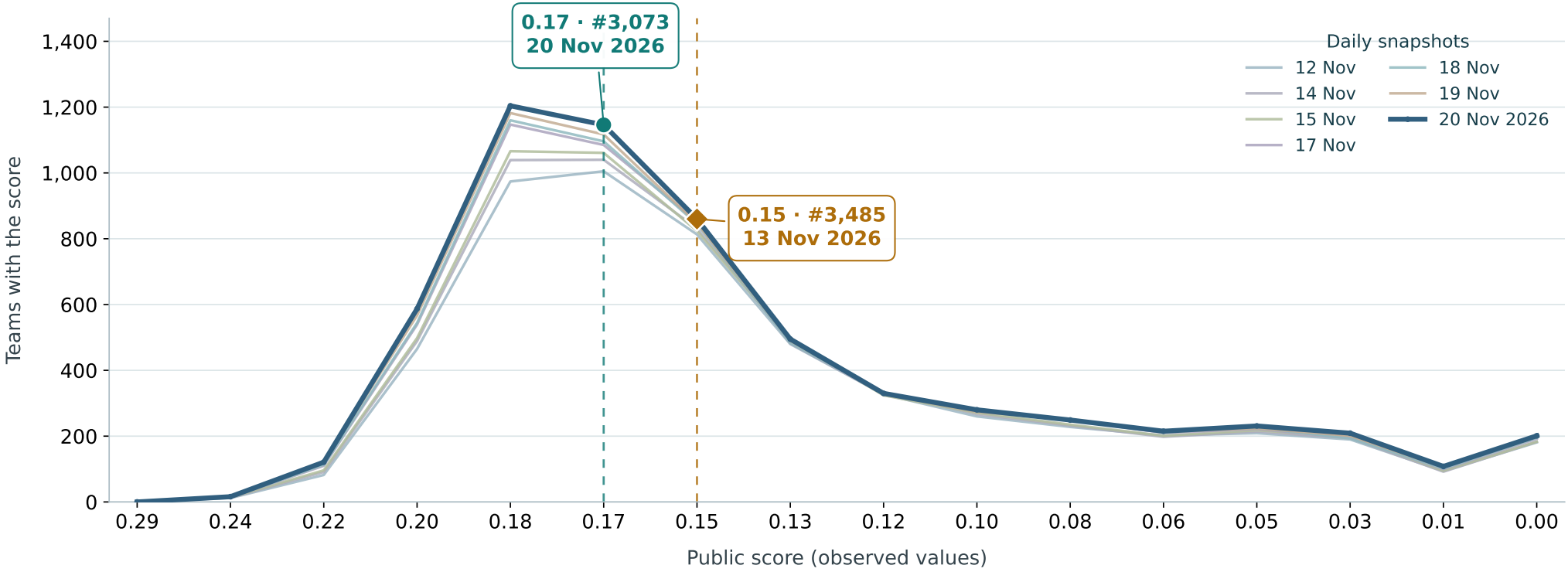
Published paper on Kaggle: https://www.kaggle.com/competitions/gemma-4-developer-agent-paper/writeups/budget-bounded-software-repair-with-ai-agents

Gemma 4 Developer Agent Competition 2026

First Kaggle competition: public leaderboard progress

Score 0.15 → 0.17 (+13.3%) Rank 3,485 → 3,073 (412 places)

17,814 entrants · 7,060 participants · 6,269 teams — viewed 20 Nov 2026, 17:09 BRT



Milestone	Score	Rank	Teams in record
13 Nov 2026	0.15	#3,485	5,523
20 Nov 2026	0.17	#3,073	6,252

Distributions: 20 Nov 2026, 11:49 BRT. Markers use the 20 Nov curve; ranks retain their stated dates.
Rank compares teams. The snapshot contains 6,252 teams; the totals above were viewed later.
Source: Kaggle and the author’s recorded history. © 2026 Aridio Silva