# Regeneration Re-Prices Contamination

A free replay oracle makes tool-agent benchmarks evergreen; whether that resists contamination depends on what you score.

June Kim
Independent Researcher
june@june.kim
ORCID 0009-0005-3153-9396

July 6, 2026

## Abstract

Every static benchmark decays: once its instances reach the public web, a model can score high by having seen them, and the number stops measuring capability. Hiding the test set leaks anyway; mining fresh instances pays a human annotation for each one. Here is a third option for stateful tool-agent tasks, where an agent drives tools against a mutable world and is graded on the final state. For these, regenerating a fresh instance and its grader is free: re-executing the task's reference program against a reseeded world computes the new goal state as a byproduct. That replay-derived oracle makes the benchmark evergreen, difficulty-controlled, and paired at zero marginal annotation cost, and we give the criterion, scored-target equivariance, for when it resists contamination. Whether it does is a property of what you score. Holding the world and the regenerator fixed, a memorized answer passes 0% of regenerated instances while a leaked, state-general SQL query passes 100%: regeneration is cosmetic against the second. The generator passes every audit check over 8,025 instances (invalid rate zero), and the boundary reproduces on a second, independently built benchmark, AppWorld, on held-out state drawn from its own generator: a memorized answer dies (0 of 13), a state-general procedure survives (32 of 36). The honest headline is that regeneration does not defeat contamination; it re-prices it, from a per-instance leak to a per-class one, and we say which adversary every claim holds against.

## 1 Introduction

Why does a benchmark stop working the day it succeeds? Because success means adoption, adoption means the instances end up in the crawl, and the next model trains on them. From then on a high score is ambiguous between two explanations that the number cannot separate: the model got better at the task, or it got better at remembering the answer. This is not a hypothetical. The measured extent of benchmark leakage across the field is large (Balloccu et al. 2024; Xu et al. 2024), and a benchmark's own authors mostly cannot audit what a lab trained on.

Two families of repair exist and both pay a recurring tax. You can hide: keep the test set private, gate it, encrypt it. Leakage routes around the barrier through forks, quoted logs, and re-uploads, and every disclosure spends down the secret. Or you can mine: continuously scrape fresh problems the models cannot have seen yet (LiveBench; LiveCodeBench; SWE-rebench). Mining works, but it pays a human to collect and verify each new instance, forever.

There is a third route, and it is only available in a specific corner. Consider a stateful tool-agent benchmark in the style of -bench: a simulated user, a database-backed world, a set of tools, and a grade computed by comparing the final database state to an annotated goal. -bench ships a reference program for each task, a canonical sequence of tool calls, and grades by replaying it to produce the goal state and hashing. That reference program is the lever. If you reseed the world (re-key the entities, resample which branch

of a conditional instruction fires) and re-run the reference program against the reseeded world, you get a fresh instance whose correct final state is computed by the same replay. The grader is not annotated. It is regenerated alongside the instance, for free.

The mechanism buys four properties unconditionally: the benchmark becomes evergreen (a fresh draw per evaluation), reproducible (the draw is seeded), difficulty-controlled (re-keying is a pure nuisance swap, and branch-selection swaps one authored bit within the same task, not a new problem), and paired (the same seed can be handed to two systems for a common-random-number comparison). The property everyone actually wants, contamination resistance, is the one we refuse to assert flatly, because it is not a property of the mechanism. It is a property of what you score, indexed to which adversary.

The contribution is three things: the observation that the replay oracle makes freshness free in this corner (§2); the equivariance criterion that says when regeneration resists contamination and when it is cosmetic (§4); and the honest accounting of what regeneration does to a determined adversary, which is re-price, not defeat (§5). None of the primitives is new. The assembly, and the criterion that tells a benchmark builder when to reach for it, is.

## 2 The oracle is the byproduct

Four research communities independently beat memorization the same way: stop hiding the test, and generate a fresh one. They differ in one thing, and that difference is the whole story: how they attach a label to a generated instance.

- Constructive (generation is grading). Where the correct answer is a byproduct of the generation procedure, you get the label for free and unlimited. Vision's domain randomization gets the class from the render; our replay oracle gets the goal state from re-executing the reference program.
- Human re-annotation. Recht-style rebuilt test sets get ground truth, expensively, once, not renewably.
- Metamorphic relation. A consistency constraint, not a label. It bounds how the answer may change under a transform without ever saying what the answer is.
- Cross-reference vote. A disagreement flag. It cannot say who is right, and it fails on shared blind spots.

The constructive rung is the strongest and the rarest. It is reachable only where the answer falls out of the generation. Vision reaches for metamorphic or vote oracles precisely when it cannot render the label; text-QA refresh either pays for the label or derives it symbolically (DyVal, GSM-Symbolic); mining sidesteps generation entirely. Stateful tool-agent tasks are the corner of language-model evaluation where the top rung is reachable, because replaying the reference program emits the instance and its grade together. That is the port, and why it is new: not a new trick, the strongest oracle trick applied where it had newly become available.

The nearest prior system, $^2$-bench, has both halves and never joins them. It has replay-derived grading (in its retail and airline domains) and it has a compositional task generator (in its telecom domain), but the generator is a one-time authoring convenience that dumps static JSON, and the two halves live in different domains and never co-occur (read from $^2$'s source, `src/tau2/evaluator/` and its telecom task generator, not inferred from its paper). The delta claimed here is exactly the join $^2$ leaves unbuilt: wire the replay oracle to seeded, reveal-at-evaluation-time regeneration, in the same domain, as a freshness protocol rather than an authoring step. "Generation is grading" is -bench's, and we cite it as such; the freshness assembly is ours.

## 3 The mechanism, and that it is sound

Concretely, a generator takes a shipped task and a seed and returns a fresh, self-contained instance: a reseeded initial database, a re-derived reference program, a re-derived instruction, and the replay-computed goal state. Two regeneration operators matter. Re-keying renames entities format-preservingly (a new order id, a new credit-card token) and holds difficulty exactly fixed; it is a pure nuisance swap. Branch-selection is the interesting one: the seed resamples the catalog content that decides which authored branch of a conditional instruction fires ("exchange the keyboard for a clicky one; if none is available, take the RGB one"), and the reference program re-derives by re-evaluating the instruction's own predicate over the reseeded data. Critically, the golden is re-derived by running the predicate, not by remembering which branch the seed chose, so a stale generator (the resampler and the predicate disagreeing) surfaces as a failed check rather than a silently wrong grade.

A generative oracle is only worth anything if it is correct, and correctness here is closer to a proof obligation than to a statistic, so the right axis is not large-n significance but large-n coverage of the generator's whole output. We audit every generated instance for injectivity of the id map, coverage (no original id leaks into the fresh database, instruction, or outputs, checked at the substring level), determinism (regenerating twice at a seed yields the same mapping and the same oracle hash), faithfulness (the re-keyed reference reproduces the shipped one's tool-error pattern position for position), and solvability. Over 8,025 generated instances across two domains, the invalid rate is zero, after the audit itself caught and we fixed a real id-leak that a five-seed unit test had certified clean: a format-preserving mint appended a suffix to digit-less airline reservation codes (`XEWRD9` → `XEWRD9_3602`), embedding the original as a substring. That leak was systematic (it hit every digit-less reservation code, not a rare draw), and the audit tests every id in every instance rather than sampling, so the zero is a coverage result and not a lucky sample. The audit paid for itself by finding a leak the small test missed, and that is the argument for running it at scale. The point of a free oracle is that this coverage costs no model calls.

## 4 When it works: score a target that moves with the state

Now the criterion. Regenerating the world is necessary for contamination resistance and nowhere near sufficient. What matters is whether the thing you grade moves when the world moves.

Call it scored-target equivariance: regeneration invalidates a memorized answer only if the scored target co-varies with the regenerated state, and recovering that co-variation requires the capability under test. The failure mode this rules out is invariance. If the scored artifact is invariant under regeneration, because correctness is defined by executing it against the data, then the correct answer is a fixed point of the whole grader family and memorizing it survives. A text-to-SQL benchmark scores the query; regenerate the rows and the gold query still executes correctly, so a leaked query passes every fresh instance. Regeneration there hardens the grader against wrong answers but does nothing for contamination. Conflating those two is the common and costly mistake.

The cleanest demonstration holds everything fixed except the scored target. On one relational substrate, with one regeneration operator, we score a perfectly-leaked adversary three ways. Score the memorized answer value (the  -bench shape) and it dies as the answer moves: on questions whose answer the resampler shifts, a leaked answer passes 0% of regenerated instances. Score the memorized SQL query (the text-to-SQL shape) and it survives untouched, 100%, because the query is state-general and the grader executes it against the fresh rows. Same database, same regenerator; the only difference is what gets graded. That is the boundary, and it is the boundary a benchmark builder actually has to reason about.

The construction is decisive but bloodless, so we ran it on a live learner. Prime a model in-context with worked (question → answer) pairs, then test on held-out questions over regenerated data with no access to the fresh rows: the memorized answer goes stale (0.62 on the shipped world, 0.12 on regenerated). Prime it instead with (question → query) pairs and it generalizes the method to held-out questions and passes every regenerated instance, including ones it never saw. A real model reproduces the boundary the construction predicts. (One methods note that cost a run: the coding-agent CLI we used as the "model" has filesystem access and quietly read the real data off disk until we sandboxed it. A control condition that should have scored zero scored 1.00, which is how we caught it.)

That leaves the criterion's second clause: does recovering the co-variation require the capability, or could a shortcut recover it? Cut the observation channel and find out. On the shipped world, where a contaminated model's memorized answer is exactly correct, blocking the one tool that reads the regenerated catalog collapses a capable model from 0.67 to 0: it retries the dead tool and refuses to fall back on the item id it recites digit-perfect from memory. The fresh answer is recoverable only by reading the regenerated world and acting on it, which is the capability the benchmark measures, not by recall and not by a cheap read of one field. That is the second clause with teeth: the co-variation is recoverable only through the measured skill, and a model carrying the right answer in its weights still cannot spend it.

The decision rule, for a benchmark builder: score the state, and derive the golden by replay. If your scored artifact is a state-general program graded by execution, regenerating the inputs is theater against a leak, no matter how much you regenerate.

Even on the good side of the boundary, "contamination-resistant" is a claim that has to be indexed, because there is no hardness assumption to lean on. Evaluating one predicate is hard for no interesting adversary. So resistance is only ever relative to a declared adversary class.

- A0, the verbatim replayer. Emits the memorized shipped answer. Defeated by any equivariant regeneration; this is what the boundary gap measures.
- A1, memorize plus cheap transport. Memorizes the re-key maps, the closed-form plug-ins, and the finite authored branch pool plus a dispatch selector. This adversary closes the branch-selection gap, because the template pool is finite, authored, and public.
- A2, fine-tune on the public generator's outputs. The strongest, and the one a memorization meter most needs to face.

The honest headline follows. Regeneration does not defeat memorization; it re-prices it, from per-instance (a leak that expires on the training-cutoff clock) to per-class (something a model must learn across the whole orbit, on a train-on-the-quotient clock), at a per-class cost bounded by the authoring budget already spent. Stated against A0 the claim is narrow and true: regeneration opens a measurable gap where cosmetic re-keying opens none. Stated against A1 the measured construct shrinks from "follow a natural-language policy over dialogue" to "dispatch among authored templates," and the gap can close.

Whether A1 actually wins is an empirical question about orbit size, and the literature answers it in a way that is sobering for a hand-authored benchmark. Models do absorb methods, not just answers: Ruis et al. (2024) find that for reasoning tasks the influential pretraining documents demonstrate how to solve (formulae, code) rather than containing the answer, and the same document is influential across many questions in a task. So a method is a learnable, memorizable unit. The question is when memorizing it is contamination and when it is skill, and that is set by orbit breadth. Cobbe et al. quantify it in reinforcement learning: agents overfit an orbit-specific policy below 4,000 procedurally generated training levels, still overfit noticeably at 16,000, and close the generalization gap only with an unbounded set of levels. GSM-Symbolic shows the memorized object is regeneration-fragile in language too: accuracy drops when only the numbers change and drops further as clause structure grows, up to 65% from a single irrelevant clause, consistent with replayed reasoning templates rather than reasoning.

The implication is uncomfortable. The seeds within a template are high-entropy, so regeneration genuinely defeats A0. But the pool of distinct authored templates is small and public. A benchmark with a few dozen hand-authored branch templates sits far below Cobbe's 16,000-still-overfits mark, so A1, memorizing the template pool, is the expected outcome, not a hypothetical. The units differ (RL levels versus authored dialogue templates), so this is an order-of-magnitude analogy rather than a like-for-like threshold; the direction is what carries. Genuine resistance against A1 therefore requires the template orbit itself to be large, which means structural class generation, procedurally generating the templates rather than hand-authoring them, not more hand-authored branches. That is the essential next step, and we do not take it here. What we can say is exactly A0, and we say it.

## 6 Why regeneration is worth the complexity

The mechanism justifies its complexity on a second axis that has nothing to do with adversaries: statistical power. A fixed static test set caps the number of instances, and that cap bites. On a 115-task retail suite, the minimum detectable effect for an unpaired comparison of two harnesses is roughly 15 to 18 points (two-proportion z-test, $= .05$, 80% power); a realistic 5-point gap needs on the order of 1,500 instances per arm. The derivation is in the repo's power receipt. Worse, the common per-task `pass^k` estimator attenuates a per-trial gap sharply as k grows. The static benchmark simply cannot resolve the comparisons people run it for.

Pairing helps, but pairing is not the generator's gift. You can already pair on the static set (hand the same 115 tasks to both systems and run an exact McNemar test), and it does not rescue you, because the n is still 115. The generator's gift is unbounded n while keeping the pairing. It lifts the instance cap and preserves the common-random-number structure, which is the combination the static set cannot offer. And it does so at zero marginal annotation, because the oracle re-derives by replay: the fixed cost of authoring the generator is paid once, and every fresh graded instance after that is free, where mining pays per instance and hand-authoring pays per task. That is the answer to "why the extra machinery": not contamination alone, but powered, paired, evergreen measurement in a regime where mining and templating cannot reach.

A caution travels with this. A gap that appears on fresh instances is necessary but not sufficient evidence of memorization; it can be reconstruction-hardness, as Recht et al. found for a rebuilt ImageNet test set where the ranking was preserved and the gains transferred. A contamination claim needs a rank-preservation or gap-decomposition analysis, not just a drop. The meter reads reliance on the fresh world; it does not, by itself, pronounce on why a number moved.

## 7 An independent second domain

A method shown only on the benchmark it was built for is a trick. To make it a method we ran the recipe on a second, independently built benchmark: AppWorld, a local tool-agent world of nine apps graded by a deterministic state-diff over the app databases, by different authors, on a different app suite. AppWorld ships a seeded task generator and a runnable reference solution per task, which is exactly the shape the recipe needs.

We did not merely observe that AppWorld ships instance variants; we ran its generator at a held-out seed across a dozen scenarios, producing fresh instances with new users and new answers that never shipped. The oracle was re-derived by replaying the freshly generated reference solution against the fresh state. On that held-out state, both sides of the equivariance boundary reproduce. A memorized answer dies: on the QA subset whose gold answer the regeneration actually moves (13 instances), a leaked shipped answer passes 0. A state-general procedure survives: across all 36 held-out instances, the shipped reference procedure, which reads the world dynamically, passes 32. Answer memorization goes stale on fresh state; procedure memorization does not, exactly as the criterion predicts, now on a benchmark we did not build. Honest scope: AppWorld and  -bench are the same genus, database-backed tool-agent benchmarks with deterministic state checks, so this is a second benchmark within the method's declared domain, not a modality-distant one. We state that and take it.

## 8 Limits

The claims here are deliberately narrow, and the narrowness does the work.

The recipe is genus-bound. It applies where state is enumerable typed entities, the interaction is a program over them, the oracle is a deterministic function of the final state (not a lookup and not a judgment), a constraint-preserving resampler exists, and the scored target is equivariant. That is a real and growing corner, agentic benchmarks, not a universal recipe.

A0 is the floor we cleared, not the adversary we most need. The measured gaps are against verbatim answer-replay. We have not run the A2 experiment that would separate genuine skill from procedure-memorization: fine-tune on the generator's outputs and test on held-out families in the CoinRun style. It is the honest hole the whole equivariance story points at, and Cobbe's numbers say the current hand-authored orbit is far too small to pass it. The fix is the same one A1 demands, structural class generation (§5): grow the template orbit past the memorization threshold. It is future work, and it closes both gaps at once.

The impact demo is piloted, not run. We built the clean instrument (one model at two reasoning-effort tiers, same weights, only compute differs) and piloted it. The pilot did its job by showing the confirmatory is over budget: the per-instance outcome is stochastic across runs, so common-random-number pairing fixes the instance but not the agent's own noise, and pinning a modest gap would need many templates and many trials per instance. So the "changes a conclusion" demonstration is pre-registered, with its feasibility numbers on file, rather than asserted.

What survives all three caveats is the part worth keeping: a free, sound, replay-derived oracle for a corner of evaluation where freshness had been expensive; a criterion, scored-target equivariance, that tells a benchmark builder when regenerating state resists contamination and when it is cosmetic; and an honest ledger of what regeneration does to a determined adversary. Score the state, derive the golden by replay, and say which adversary you mean. The rest is authoring budget.

## LLM collaboration disclosure

LLMs enter this work in three roles. Subject of study: the benchmark drives shipped LLMs as the agents under test (Claude Opus 4.8 at low and high reasoning effort, Cursor's Composer 2.5, and an in-context learner), and the recall probes query Claude Opus 4.8 and OpenAI's GPT-5.5; the model versions and exact commands are recorded in the archived receipts. Instrument: an independent model family adversarially reviewed the design and the results (Anthropic's Fable), a separate model verified every citation against source (Claude Sonnet), and the verdicts that matter rest on a mechanical layer, the replay oracle and

the audit harness and McNemar's test, that no model overrides. Writing aid: the prose was drafted and revised with Anthropic's Claude (Opus 4.8) from the author's repository documents, experiment receipts, and direction; the method, the experiments, the numbers, and the argument are the author's. No LLM decided what to publish.

## Funding

This work was conducted independently, with no external, institutional, or commercial funding. All compute and model-API costs were borne by the author.