When the exploit is constructed: ranking collapses, counting survives

W. Schulz

Jul 30, 2026

Abstract

At a matched exploit rate, Best-of-8192 accuracy is 0.862 with sampled exploits and 0.000 with exploits constructed against the verifier — while plurality voting among high scorers holds at 0.999. The consensus threshold missed a dominant-wrong-answer regime and was corrected; the two-regime framing is prior art (Dwork et al. 2015).

Every result in this project's verifier track rests on one modelling choice that nobody had examined. The track models a verifier's blind spots as sampled: each wrong candidate is, with some fixed probability, a thing the verifier happens to score highly. That probability is the quantity the whole track is built around — it sets the safe sampling budget, it decides whether more compute helps or hurts, and it is the number the released command-line tool asks you for.

On 30 July 2026 an outside observation made the choice look suspicious. A talk about GPU kernel competitions catalogued the reward hacks that occur in a *real* adversarial population — tens of thousands of submissions written by people and by language models trying to win a leaderboard. Not one of them was sampled. The fastest submitted kernel for computing a vector's mean returned zero, because the test inputs are drawn with mean zero. A ban on one Python function was defeated by reaching it through a different accessor, then by splitting its name across two source lines so the pattern-matching detector could not see it, then by an entirely different function that does the same job. And one submission counted how many times it was being checked for correctness, behaved correctly during those checks, and behaved differently afterwards.

These are not blind spots a search stumbled into. They are blind spots built to specification, from the checker's own definition. The question that follows is narrow and answerable: at the *same* rate, does it matter whether an exploit was sampled or constructed?

It matters completely, and in one direction

The experiment holds everything fixed — rate, discrimination, clustering — and changes only how the exploiting candidate gets its score. In the sampled arm it draws from a fixed distribution, as the track has always assumed. In the constructed arm it scores just above the best honest candidate in its own problem: defined in terms of the checker's realized output rather than drawn independently of it.

With the rate matched to three decimal places, selecting the top-scoring candidate out of eight thousand gives 0.862 accuracy under sampled exploits and 0.000 under constructed ones. The safe budget falls fourfold. The mechanism is not subtle: a sampled exploit only *probabilistically* outscores the field, so most of the time the correct answer still wins; a constructed one wins with certainty the moment it

exists. A blind spot occurring once in a thousand candidates is survivable when it is stumbled upon and total when it is built.

So the safe-budget tables are a sampled-exploit result. That is now a caveat on the released tool.

The defence that survives, and why

Two of the four predictions failed, and they failed for the same reason — which turned out to be worth more than the prediction would have been.

The track's recommended defence is not to take the top-scoring candidate but to take the *most common answer among the candidates that clear a score threshold*. Under constructed exploits this defence was expected to collapse, since a constructed exploit clears any threshold by construction. It did not collapse. It scored 0.999 — identical to its performance against sampled exploits.

> Construction attacks ranking. It does not attack counting.

An exploit built to outscore the field has, by that very design, done nothing about *how many* candidates give each answer. Sorting is vulnerable to a constructed adversary; tallying is not. This strengthens the track's guidance rather than weakening it: the improvement from ranking to counting is worth 0.19 → 0.999 against sampled exploits and 0.000 → 0.999 against constructed ones. The defence matters more, not less, against an adversary that can see your scorer.

The third failed prediction was a plain mistake. Modelling the arms race properly — an exploit is discovered once and then *copied* by everyone, so its prevalence climbs with search rather than staying fixed — the effective rate rose from one in a thousand to 86%, and accuracy still showed a clean peak before collapsing. Adoption relocates the safe budget; it does not abolish it. The prediction said abolish.

A correction to the project's own shipped advice

Chasing whether the counting defence transfers to a leaderboard — where correct answers are *scarce* and a discovered trick is universal — produced the run's most useful result, and it is a correction rather than a discovery.

The track ships a threshold: below a certain exploit rate, determined by how often problems are solvable at all, voting neutralizes the collapse entirely. Tested at low solve rates, that threshold predicted safety and the vote was lost anyway. Counting the surviving candidates by category explains it. At a 5% solve rate the winner of the vote is neither the correct answer nor the exploit. It is the dominant wrong answer — the plausible mistake that many independent candidates make for ordinary reasons, which the threshold formula does not model at all.

The corrected boundary is a three-way count rather than a two-way one, and it reproduces the measured survivor counts to three significant figures. Its practical content is blunt: whether consensus is available to you is a fact about your solve rate, your scorer's discrimination and your filter threshold — a condition with no exploit rate in it whatsoever. Below a certain difficulty, neither selection rule is safe, and the shipped tool now says so with a live warning.

What the patch loop can and cannot buy

The talk also stated a hopeful claim: as more people compete, more hacks get found, and the evaluation gets more robust. That is a claim that exploitability tends to zero given enough rounds. The defence actually deployed was a pattern-matching detector, automatically synthesized from previously-caught

hacks — and the worst example in the talk defeats it by splitting a function name across two lines, which such a detector *cannot express*. That is not a missing patch. It is a limit of the language the patches are written in.

Simulating the loop gives the shape. When the detector's language can express everything the adversary can build, exploitability does go to zero. When it cannot, exploitability floors at exactly the fraction of exploit space the language cannot reach, and stays there. Doubling the number of rounds moved that floor by 0.007. Adding one level of expressive reach to the detector moved it by 0.124 — about eighteen times more. Upgrading from a pattern-matcher to a learned detector, which the talk names as the thing it would buy with more funding, converges roughly three times faster to *precisely the same floor*.

Sophistication within a language buys speed. Only reach buys ground.

The honest accounting

Two adversarial literature sweeps returned on this work the same day it was written, and both cut against it.

The framing is owned, and by eleven years. Dwork, Feldman, Hardt, Pitassi, Reingold and Roth's 2015 work on adaptive data analysis [1] is the canonical version of exactly this split, proved: the held-out set *is* the checker, queries chosen without looking at it are safe, queries chosen after looking at it destroy its validity — and the fix is explicitly not more or more-independent held-out sets. Garrabrant's 2017 taxonomy of Goodhart's law already forks the *remedies* by regime [2]. The coverage result the patch-loop experiment reproduces has a 2020 owner: Zhuang and Hadfield-Menell's analysis of proxies that cover only some of the attributes the principal cares about [3]. Nothing above is new, and the recommendation adopted was to stop: no write-up, no further experiments on that thread.

One published result points the opposite way from the prescription. The formal treatment of ungameable ("unhackable") proxies singles out *simplification* — a coarser proxy, not a richer one — as its important special case [4], and Garrabrant's tentative advice for the adversarial regime agrees: choose a proxy so simple that an adversary has minimal control (offered, in his words, as "not a good plan" but "all I have") [2]. The patch-loop conclusion was that the lever is expressive *reach*. A reconciliation suggests itself — a simpler reward shrinks the exploit space while a richer detector covers more of a fixed one, and both close the same gap — but it is a hypothesis held to the same standard as the objection, and it has not been checked. Anyone building on the coverage result has to resolve that first.

And two papers say the decorrelation half is wrong. Ensemble-based conservative optimization has been reported to practically eliminate over-optimization for best-of-N sampling [5], and model averaging has been presented as a remedy under strategic manipulation [6]. "Decorrelation fails against an adaptive adversary" is not settled.

And this retracts a claim made earlier in this collection. The chapter reporting the adaptive-surplus result closed by saying that the measured two-regime decomposition — a surplus floor and a coverage cliff — appeared unclaimed. It is not unclaimed; it was proved in 2015, and the coverage half has a 2020 owner. The measurements in that chapter stand exactly as run. Its novelty claim does not, and is withdrawn here rather than left to age quietly.

What survives the sweeps is narrow and it is the part that was never about novelty: the measured collapse from 0.862 to 0.000 at matched rate, and the correction to this project's own threshold. Both are statements about the project's shipped tool, and both stand.

The real population, which turns out to be measured already

The obvious weakness of everything above is that it is simulated. The obvious remedy is the kernel competition itself, and a third sweep went looking for whether its data is public.

It is. The submission corpus is downloadable [8], and roughly eleven hundred submissions across its configurations carry a machine-assigned status of *hacked*, with source code, author and timestamp retained. [verification pending] A subset carries the detector's verdict verbatim, naming the rule each one tripped — output-replay caching, pointer replay, last-call replay. There is even a rule that fires when a submission branches on which evaluation mode it is running in: a literal defeat-device detector, already shipped.

More than that, the exploit rate has already been measured on live traffic. A workshop paper by the competition's own team reports the hacked-submission rate falling from 3.45% to 0.37% over a 32-day, 182,798-submission window after their detector was integrated [7]. That is the closest thing in the literature to the number this project has only ever obtained by inverting somebody else's over-optimization curves.

Two things about it deserve stating plainly. It is an observational before-and-after on live traffic, not a controlled comparison. And the exploit count is the detector's own verdict, so the metric is partly coupled by definition to the thing it evaluates — nobody has audited the auditor, and there is no independent characterization of what it misses. A residual 0.37% is exactly what a floor set by expressive reach would look like, and it is also exactly what a detector scoring its own homework would look like. The simulation cannot tell those apart, and neither can the published number.

That last gap — an independent false-negative characterization of a deployed exploit detector — is the one genuinely open slot the sweeps left. It is also, unlike everything else in this chapter, a question about a real system with a public corpus attached.

A correction to how this chapter's own premise was described

One example above was originally characterized as a defeat device in the purest sense: a submission that detects it is being tested and swaps in incorrect results. The post-mortem of that specific case says explicitly that it is not. The outputs are correct in both phases; what the submission games is *timing amortization*, charging one batched launch against fifteen measured ones. The counting-the-checks behaviour is real, and a separate detector rule exists for it. The correctness swap was a gloss, and it was not in the record.

It is the kind of error this project's methodology is built to catch, arriving in the one place the methodology had not been pointed — at a secondhand account of somebody else's system, accepted because it was vivid.

Sources

1. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., & Roth, A. (2015). "Preserving Statistical Validity in Adaptive Data Analysis." *STOC 2015.* arXiv:1411.2664
2. Garrabrant, S. (2017). "Goodhart Taxonomy." *LessWrong.* lesswrong.com — four regimes (regressional, causal, extremal, adversarial), each with its own mitigation section; see also Manheim, D. & Garrabrant, S. (2018), "Categorizing Variants of Goodhart's Law," arXiv:1803.04585.

3. Zhuang, S. & Hadfield-Menell, D. (2020). "Consequences of Misaligned AI." *NeurIPS 2020*. arXiv:2102.03896
4. Skalse, J., Howe, N. H. R., Krasheninnikov, D., & Krueger, D. (2022). "Defining and Characterizing Reward Hacking." *NeurIPS 35*. arXiv:2209.13085
5. Coste, T., Anwar, U., Kirk, R., & Krueger, D. (2024). "Reward Model Ensembles Help Mitigate Overoptimization." *ICLR 2024*. arXiv:2310.02743 — "conservative optimization practically eliminates overoptimization" for best-of-N; for PPO it reduces it.
6. Hennessy, C. A. & Goodhart, C. A. E. (2023). "Goodhart's Law and Machine Learning: A Structural Perspective." *International Economic Review* 64(3): 1075–1086. doi:10.1111/iere.12633
7. Baslak, M. E., Schultheis, E., Sirovatka, M., Zhang, A., & Saroufim, M. (2026). "We Let Agents Compete and They Tried to Cheat. KernelGuard: Defending GPU Competitions from Adversarial Agentic Systems." Second Workshop on Agents in the Wild, *ICML 2026*. icml.cc
8. GPU MODE, *kernelbot-data* (submission corpus). Hugging Face. huggingface.co/datasets/GPUMODE/kernelbot-data

The experiments (sampled vs constructed exploits, the counting defence, the threshold correction and the patch-loop simulation) are this project's own run-once results, recorded in `verifier-bottleneck/FINDINGS.md` *in the Quantum-Intelligence repository.*

Published by NeuroGenesys (AW3 Technology, Inc.). Canonical version: https://neurogenesys.si/research/constructed-exploits