MEASURE FIRST: THE NOISE FLOOR OF AN AUTONOMOUS RESEARCH LOOP DOMINATES ITS ACCEPT RULE

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous research agents change their code, retrain, evaluate, and keep the change if the score went up. Recent work proposes better statistical tests for this keep-or-discard decision. We argue that the first problem is that nobody measures how noisy the score is, and we measure it in four ways. First, we re-run one production LLM distillation-and-quantization pipeline, unchanged, thirty times in each of sixteen settings (480 runs): run-to-run spread varies more than 13-fold across settings, and the improvement the pipeline's own authors were chasing cannot be distinguished from this noise (its 95% upper bound is 0.54 of it). Second, we read the source code of 26 systems: sixteen of eighteen deployed agent loops keep a change after a single run, and none lets a noise estimate reach the decision. Third, we replay eleven decision rules on the measured runs. The anytime-valid e-process, written against a noisy baseline as is natural, is not valid (false-accept 0.038, 0.148 and 0.278 at 8, 32 and 128 runs); corrected, its best variant reaches 0.048 power at half a standard deviation, where a five-run Welch t-test reaches 0.170 and needs no external noise estimate. Taking that estimate from three runs breaks the corrected rule in all sixteen settings. Fourth, we evaluate the same frozen weights 3,627 times on seven NVIDIA GPU models. Scores move in whole questions; 13 of 39 model, task and GPU combinations return the identical score every time while others drift by up to thirty questions; two GPUs that are each perfectly repeatable still disagree by twenty; and running inputs one at a time instead of batched removed the drift in every case we tested. We recommend two things: before trusting any comparison, re-run the unchanged pipeline several times on the hardware the loop will use, to learn how much the score moves on its own; and decide with a test that estimates the noise from the runs themselves, such as a Welch t-test, rather than one handed a noise level from outside.

1 INTRODUCTION

An autonomous research loop changes its own code, retrains, evaluates, and keeps the change if the score went up. Suppose the score rises by half a point. Did the change do that, or would re-running the unchanged code have moved the score as much? The loop cannot tell, and it makes this decision hundreds of times, each one conditioning the next. Recent work has identified this keep-or-discard decision, the *accept rule*, as the weak point (Shawn, 2026; Sengupta, 2026; Huang et al., 2025) and proposed anytime-valid statistical tests for it.

We think the accept rule is the right place to look and the wrong thing to fix first. Any rule needs to know how much the score moves on its own, its *noise floor*, and none of the deployed loops we surveyed measures it. Our claim is that the missing measurement, not the missing test, is what binds. We take that measurement in four steps, each answering the question the previous one raises.

What we did.

1. **How noisy is the score?** We re-ran one production LLM pipeline (distillation, a hyperparameter search, GPTQ quantization, 8 benchmarks), unchanged, 30 times in each of 16

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

settings: 480 runs. The run-to-run spread varies 13.2-fold across settings, and the improvement the pipeline's own authors were chasing cannot be told from it (Sec. 3).

2. **Do deployed loops account for it?** We read the source code of 26 systems. Of 18 deployed loops, 16 keep a change after a single run, and none lets a noise estimate reach the decision; three compute one and then ignore it (Sec. 4).
3. **Given the measured noise, which rule should a loop use?** We replayed eleven accept rules on the real runs. The anytime-valid e-process, as naturally written, is not valid; corrected, a five-run Welch t-test has 3.5$\times$ its power for fewer runs. What reliably breaks a rule's guarantee is feeding it a noise estimate taken from a few runs (Sec. 5).
4. **Is the noise a property of the code?** We evaluated the same frozen weights 3,627 times on 7 GPU models. Scores move in whole questions; 13 of 39 model, task and GPU combinations never move at all, while others drift by up to 30 questions; and running inputs unbatched removed the drift in every case we tried (Sec. 6). The noise a loop must measure belongs to the machine it runs on, not only to its code.

Related work. That single-run comparisons in machine learning are unreliable is well established: in deep RL (Henderson et al., 2018; Agarwal et al., 2021), supervised benchmarking (Bouthillier et al., 2021; Dodge et al., 2019), NLP score reporting (Reimers & Gurevych, 2017), LLM evaluation (Madaan et al., 2024; Miller, 2024), and leaderboard selection (Sculley et al., 2018). What is new is the consumer: those papers ask a human to report an error bar, while a loop makes the comparison itself, hundreds of times. Ning et al. (2026) make the complementary measurement, repeating the *implementation* of an idea; we repeat the *run*. Among the accept-rule proposals, PACE (Shawn, 2026) opens on our premise almost verbatim and gates on paired *binary* outcomes, POPPER (Huang et al., 2025) is distribution-free by construction, and SEA (Sengupta, 2026) bounds rewards in $[0, 1]$. Each gets its noise scale by bounding the outcome, a trade unavailable to a loop tuning accuracy or validation loss at fine resolution. Where the metric stays continuous the scale has to be estimated from repeated runs, and what that estimate costs is the question this paper answers.

2 SETUP

The pipeline. The system under measurement is a production LLM pipeline that distils a large teacher into a small student under a LoRA adapter (Hu et al., 2021), tunes the distillation hyperparameters with Optuna (Akiba et al., 2019), quantizes the result with GPTQ (Frantar et al., 2023), and reports accuracy on 8 multiple-choice benchmarks. It is the kind of artifact an autonomous loop would be pointed at: a scalar metric, an expensive run, and a proposal-accept structure.

The replication. We re-executed this pipeline in each of 16 conditions (8 tasks $\times$ two optimizers, Adam (Kingma & Ba, 2015) and Muon (Jordan et al., 2024)) to $n = 30$ replicates: 480 runs and 960 benchmark evaluations, all completing without failure. Every run carries a matched pre- and post-quantization evaluation; the 80-run sub-block on which the proxy analysis was first run (Appendix B) also has its 1,280 hyperparameter trials logged. Commands came verbatim from the original experiment manifests, with only replicate indices and output roots rewritten, and every evaluation used one pinned harness (`lm-eval` 0.4.9.1 (Gao et al., 2024; Biderman et al., 2024) under vLLM (Kwon et al., 2023), identical `model_args`). The pipeline exposed no seed flag, so replicates differ in the Optuna draw, in initialisation and in nondeterministic GPU kernels: precisely how a loop re-runs a proposal it has already written. The data-synthesis stage was not re-executed: one frozen synthetic dataset is shared by all 480 runs, so our σ is that of training, quantization and evaluation with the data held fixed. A further 78 archived runs from an earlier evaluation stack are excluded from every variance estimate, because mixing evaluation environments would confound re-execution noise with harness drift.

3 THE MEASURED FLOOR

The floor moves by more than an order of magnitude across conditions. Table 1 gives the mean and replicate SD of every condition at $n = 30$; Figure 1 in Appendix A plots all 16 on a log axis with bootstrap CIs. The tightest condition (HS/muon) sits at 0.082 accuracy points and the widest (MMLU/adam) at 8.64, a ratio of 105$\times$. That ratio is driven by one collapsed-basin condition,

and a paper about the hazards of reporting maxima should not lead with a max-driven statistic: dropping the single widest condition still leaves 13.2× between HS/muon and MMLU/muon. Both widest conditions are MMLU; dropping that task outright still leaves 6.9×, to OBQA/adam. Which condition is "tightest" depends on the unit: in items (Table 1) it is OBQA/muon at 0.87 items, while HS/muon is 8.3, because an item SD grows with the size of the evaluation set. Across tasks, points are the fair comparison; for reading whether one run's gain is one question or thirty, items are, and Sec. 6 uses them. Either way, a fixed threshold ("keep it if it gained 0.5 points") is a different test in every condition: that gain is 6.1σ in HS/muon and 0.06σ in MMLU/adam, and 3 of 16 conditions have a replicate SD that exceeds it outright. The median condition sits at 0.186 points.

The effect being chased is smaller than the noise. Write ρ for the ratio of the effect a proposal is expected to produce to the run-to-run SD of the quantity it is read from. When $\rho \gg 1$ the cheap rule is fine and statistics are a tax; when $\rho \lesssim 1$ a single run cannot separate an improvement from luck. The comparison this pipeline's authors were making, the optimizer's effect on the quantization gap, averages 0.141 accuracy points in absolute value over tasks, against a median replicate SD of 0.636 points on the same quantity (the gap differences two noisy accuracies, so it is noisier than accuracy's 0.186), so $\rho = 0.22$. That is what noise alone produces: with no true difference the same unsigned estimator averages 0.21 (permutation $p = 0.43$), the signed mean effect is -0.14 ± 0.10 in units of that SD, and the 95% upper bound on the true mean absolute effect is 0.54. At 30 runs per arm the experiment bounds its own headline effect rather than detecting it, and even at $\rho = 0.22$ a loop comparing one run per arm would get the sign right 56% of the time. Heineman et al. (2025) define a signal-to-noise ratio for evaluation and raise it by improving the benchmark; ρ is the same shape measured over re-execution of the whole pipeline.

"So seed your runs." The obvious objection, and it deserves a measurement rather than an argument. A replicate here differs from its neighbour in the hyperparameter draw, in training nondeterminism, and in evaluation. The first is seedable: seeded, repeated runs select byte-identical hyperparameters, and in the one condition where we tested it seeding more than halved the replicate spread. The third is not seedable at all, because by the time it is incurred the weights are frozen and there is no seed left to set, and it is exactly what Sec. 6 measures by evaluating fixed weights over and over. On the 39 cells there, the median wandering cell contributes 0.024 points of standard deviation with nothing varying but the kernels. We can do better than that cross-task comparison, because this pipeline's own student is a Llama-3.2-3B and it runs on B200s, which is a cell in the archive. Matched on the student's base model, the card and the three tasks the archive shares with the pipeline, the evaluation term is <0.01% to 11.6% of the pipeline's variance, largest on ARC-e/muon and negligible where the condition is itself wide. So seeding is worth doing and cannot reach zero. Unbatched evaluation can: in the four perturbing arms we re-ran one sequence at a time, this term vanished (Sec. 6).

Five runs is a draw, not a measurement. The same data answer the question a practitioner actually faces: how deep does a calibration have to be? Taking the first five replicates of each condition (the depth a careful study would report) and comparing against the $n = 30$ estimate and its bootstrap interval, 9 of 16 five-replicate SDs fall *outside* the 95% CI of the deep estimate, 4 of them understating the noise and 5 overstating it. It moves in both directions, so this is imprecision rather than a correctable bias. At $k = 3$ the median $\hat{\sigma}/\sigma$ ratio runs from 0.66 to 0.99 across conditions and the probability of being off by more than 25% averages 0.65; at $k = 20$ that probability is 0.01. The $k = 3$ figure is textbook, matching the 0.64 that the χ^2 sampling distribution of $\hat{\sigma}$ predicts for normal data, so a three-run calibration is sizeable from theory and no surveyed system sizes it at all. The $k = 20$ figure beats theory's 0.12, and we do not claim the credit: those subsets come from the same 30 runs that define the reference, so the estimate sits inside its own target. At $k = 15$, the deepest split that leaves a disjoint reference, in-sample gives 0.06 and a disjoint split 0.34, above theory's 0.19 because a 15-run reference is noisy too. An independent twenty-run $\hat{\sigma}$ should be off by more than 25% about as often as theory says, 0.12 of the time, and the plug-in costs in Sec. 5.1, whose subsets overlap the reference the same way, are lower bounds.

The optimized proxy runs backwards. The pipeline's search minimises a validation loss while the paper reports benchmark accuracy, so the search is only as good as the agreement between them. Pooled over the 480 runs after standardising both quantities within each condition, the correlation

Table 1: Measured noise floor at $n = 30$ per condition. Mean and replicate standard deviation of full-precision accuracy for each task under each optimizer, in accuracy points and, beside it, in items (the SD times that task's N). $\rho_{0.5}$ is the signal-to-noise ratio a fixed 0.5-point gain would achieve in the noisier of the two optimizers for that task. Per-condition bootstrap intervals are in Appendix A.

	Adam			Muon			
Task	Mean	SD	items	Mean	SD	items	$\rho_{0.5}$
ARC-e	70.32	0.45	10.7	70.97	0.11	2.6	1.11
CQA	73.77	0.22	2.7	73.99	0.13	1.5	2.23
HS	71.55	0.15	15.5	71.76	0.08	8.3	3.25
MMLU	49.60	8.64	1213.8	61.08	1.09	152.4	0.06
OBQA	38.70	0.57	2.8	39.40	0.17	0.9	0.88
PIQA	77.29	0.13	2.3	77.14	0.16	2.9	3.20
SIQA	47.07	0.30	5.8	47.23	0.11	2.1	1.67
WG	68.57	0.25	3.2	68.71	0.20	2.5	1.98

is $r = +0.276$ (cluster-permutation $p < 10^{-4}$): replicates whose search reached a lower validation loss tend to score lower. The sign differs by optimizer: positive in all eight Adam conditions (six significantly), while Muon holds all three significant negative correlations. With MMLU set aside, a cluster bootstrap over conditions spans zero (Appendix B).

4 How deployed loops actually decide

Section 3 shows the noise is large and uneven. Here we ask whether the loops people actually run take it into account. We surveyed 26 systems (18 deployed autonomous-research loops, 3 benchmarks that define the protocol those loops are scored under, and 5 proposed accept rules) in four strata, each drawn to exhaustion from a stated frame: three community indexes of autoresearch systems for the loops, the agentic-research literature for the benchmarks, and forward citation and arXiv search for the proposed rules. The frames are incomplete, so the sample is not exhaustive of the field. Every entry was read from primary source: a repository cloned and inspected with line numbers at a named commit, or a method section read in full text. Nothing is inferred from a README summary or a search snippet. The sample spans the community anchor and its ports (Karpathy, 2026), published research agents (AI-Scientist (Lu et al., 2024), AIDE (Jiang et al., 2025), MLAgentBench (Huang et al., 2024), AlphaEvolve (Novikov et al., 2025) and others), one deployed in-production loop (Wu & Hsieh, 2026), the benchmarks that define the protocol (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025), and the proposed accept rules (Jain & Linares, 2026; Roth et al., 2026; Shawn, 2026; Sengupta, 2026; Huang et al., 2025). Every entry, with the commit read and the file-and-line evidence behind its verdict, is in Appendix C. Ding et al. (2026) independently survey 24 runnable systems on a coarser *verification* axis and Ren et al. (2026) systematize 56; both report compatible headlines. We ask specifically what the accept rule is, and read it from source.

Four of the 18 are the community anchor and three ports of it, but twelve of the remaining fourteen still commit on a single run, so the headline does not rest on that lineage. Over all 18: 16 commit on a single run of the candidate; 0 compute any variance estimate in the accept path; and 0 apply any multiplicity correction across the trial sequence. A uniform grep for every correction procedure we could name returned zero hits in every repository.

The interesting finding is not that everyone uses one run. It is that the practitioner community has independently rediscovered the problem, written the fix down, and not wired it in. 3 of the 18 loops state a noise-aware accept rule and 0 of those 3 enforce it in the shipped loop: one implements a complete bootstrap gate that the agent-facing instruction file never invokes; one stores and type-validates a `noiseFloor` that is never compared to a metric anywhere in the repository; one computes a median-absolute-deviation confidence score that its own README calls advisory and says never auto-discards. Where a noise scale does reach a decision it is a constant: one system subtracts a hard-coded 0.1 (ten accuracy points) from the incumbent's score, with a default of one run per candidate and the gate off by default.

Exactly one surveyed accept rule estimates its noise scale from replicates. Of the 18 deployed loops, zero estimate a variance proxy. Of the 5 proposals exactly one estimates σ from runs (the sequential gates bound the outcome instead, Sec. 1): a credibility layer that estimates σ from $K = 10$ baseline runs and verifies each accepted configuration on 10 further runs, 20 runs beyond the search's own. It is therefore the only surveyed system operating in the regime our measurement is about: a continuous metric, a proxy estimated from replicates, and a gate whose validity rests on that estimate. It pays exactly the cost we quantify, and on normal data a 10-run $\hat{\sigma}$ is off by more than 25% 0.29 of the time (χ^2, as in Sec. 3); by analogy with the $\eta = 2\sigma$ Ladder (Sec. 5.1), a margin as wide as its 2σ should absorb that error by giving up power rather than level. The claim is about the cost of σ, not about anyone's diligence. (That system is a spacecraft guidance-and-control loop, a domain where a run is cheap enough to afford 20 of them. For an LLM pipeline it is not, which sharpens rather than weakens the point.)

5 ACCEPT RULES, REPLAYED ON THE REAL RUNS

The loops do not measure the noise (Sec. 4). Suppose one did: which accept rule should it then use? We replay eleven accept rules over the measured pools: what loops ship (best-of-1, and peeking at a growing pool without correction); the σ-free classical options (a fixed-n Welch t (Welch, 1947), Bonferroni-corrected peeking, and AdaStop (Mathieu et al., 2024), a group-sequential permutation test); the margin rules descended from the Ladder (Blum & Hardt, 2015) and its adaptive-data-analysis setting (Dwork et al., 2015), a loop reading one benchmark to choose its next move being a reused holdout; and a Robbins normal-mixture e-process (Robbins, 1970; Howard et al., 2021; Ramdas et al., 2023), the textbook anytime-valid gate for a continuous metric with known scale, both as it is naturally written and corrected. It is not any surveyed proposal's gate: those bound the outcome instead (Sec. 1). Classical racing and best-arm identification (Maron & Moore, 1993; Even-Dar et al., 2006) eliminate arms under a fixed confidence budget; a loop instead presents one candidate against one incumbent, which is what we replay. AdaStop is enumerated exactly (N=2 scores per agent per interim, K=4 interims), in its published two-sided form and a labelled one-sided variant. A decision is a comparison of a candidate against an incumbent; the incumbent's runs are drawn from a condition's real $n = 30$ pool and the candidate's from the same pool shifted by a known Δ in units of that condition's own SD, so $\Delta = 0$ is an exactly true null. Within a column every rule sees the same draws, $\alpha = 0.05$, candidate horizon $k_{\max} = 8$, 5 incumbent runs, and 60,000 trials per cell, each drawing its condition at random.

The replay first reproduces an earlier reference implementation of itself: best-of-1, which consumes a known prefix of the random stream, gives 0.4983 against the reference's 0.4990 over its 3,000 trials, and its e-process, re-run over 8 seeds, gives 0.0483 (SD 0.0026) against a single-seed 0.053.

The e-process as written is not anytime-valid. The natural candidate-versus-incumbent gate sums $(x_i - \bar{y}_m)/\sigma$ over candidate runs against the incumbent's $m = 5$-run mean, as if that mean were known. It is not: its error is shared by every term, so the sum's variance grows as $t + t^2/m$ rather than t, and the mixture under-covers more the longer the gate runs. Its false-accept rate is 0.038 at our horizon of 8 runs, 0.148 at 32 and 0.278 at 128, and tends to $1/2$: whenever the incumbent's mean came out low, S_t drifts upward without bound. The fix runs the same mixture on the information clock $I_t = tm/(m + t)$ with $B_t = S_t\, m/(m + t)$, under which the score is Brownian motion and Ville's inequality holds at every horizon: 0.002, 0.003 and 0.004. This is the classical sequential test against a fixed control, and since $I_t < m$ its evidence is capped by the incumbent's own runs, a cap any test against a fixed five-run incumbent shares and one that re-drawing the incumbent, as AdaStop does, lifts.

The sophisticated machinery does not pay for itself. Corrected, the gate holds α with room to spare and has little power left: 0.018 at $\Delta = 0.5\sigma$ for 7.95 candidate runs. The most favourable valid variant we tried, a one-sided mixture at $\tau = 2$ (of $\tau \in \{0.25, 0.5, 1, 2\}$), reaches 0.048 at false-accept 0.007. A fixed-n Welch t on five runs reaches 0.170: 3.5× that, for fewer runs and with no external σ. This is the known price of time-uniform validity at a short horizon, and a loop that knows its horizon should not pay it. Even the gate as written, which buys power with a level it does not have, is no bargain: Welch has 93% of its power for 67% of its runs. The most powerful valid rule in the table is AdaStop's one-sided variant (0.227), at 7.68 candidate runs plus as many

Table 2: Eleven accept rules replayed over the real $n = 30$ pools, $\alpha = 0.05$, pooled over 16 conditions. "σ" says whether the rule consumes an *externally supplied* noise scale. Power is at Δ in units of the condition's own SD; "cand" is mean candidate runs at $\Delta = 0$, the cost a loop pays almost all the time, and "tot" adds the incumbent runs, which AdaStop must re-draw at each interim rather than reuse. The last false-accept column estimates the noise scale from three runs instead of knowing it. Blocks: shipped; σ-free; plug-in σ. Every rate has Monte-Carlo SE at most 0.002.

		oracle σ			$\hat{\sigma}(k{=}3)$	runs	
Rule	σ	false-acc.	pw@0.5σ	pw@1σ	false-acc.	cand	tot
best-of-1 *(what loops ship)*	none	0.514	0.685	0.817	0.513	1.00	6.00
naive peeking	none	0.120	0.325	0.606	0.122	3.21	8.21
Ladder, parameter-free	none	0.265	0.551	0.812	0.266	5.00	10.00
Welch t**,** $k = 5$	none	0.045	0.170	0.411	0.044	5.00	10.00
Bonferroni-corrected peeking	none	0.020	0.085	0.245	0.021	7.90	12.90
AdaStop $N{=}2, K{=}4$	none	0.023	0.139	0.399	0.022	7.94	15.88
AdaStop, one-sided variant	none	0.047	0.227	0.548	0.047	7.94	15.88
Ladder, $\eta = \sigma$	plug-in	0.054	0.209	0.487	0.118	5.00	10.00
Ladder, $\eta = 2\sigma$	plug-in	0.001	0.008	0.053	0.039	5.00	10.00
Robbins e-process, as written	plug-in	0.038	0.182	0.484	0.138	7.91	12.91
Robbins e-process, corrected	plug-in	0.002	0.018	0.108	0.075	8.00	13.00

re-drawn incumbent runs. Without a futility boundary, which we did not add, the e-process cannot stop early near $\Delta = 0$, where a loop lives: corrected, it spends at least 7.95 of its 8 candidate runs at every $\Delta \leq 0.5\sigma$. At matched power (Table 10, Appendix D) there is nothing to match up to $\Delta = 0.5\sigma$, since a valid Welch test at any n has more power, and at 1σ Welch needs 3 runs per arm against the gate's 7.67.

On real, unequal-variance comparisons the plug-in is what hurts. The replay above shifts one pool by a constant, so both arms carry the same variance and Welch's reason to exist is never exercised. We therefore re-ran every rule on the sixteen real comparisons available to us, Adam against Muon on each task in both directions, both arms at their own $n = 30$, arm-to-arm SD ratios from 0.13 to 7.96, and the truth read off the full pools rather than imposed. Welch reaches 0.595 power at 5.00 runs. The corrected e-process, given the incumbent's σ (the only one a loop has before running the candidate), reaches 0.106; handed the candidate's own, which no loop has, it would reach 0.619. AdaStop, which re-draws the incumbent, is the most powerful here: its one-sided variant reaches 0.709, the two-sided form 0.621 at 6.96 runs. Best-of-1, what loops ship, keeps a genuinely *worse* candidate 14.0% of the time, against 0.2% for Welch.

But σ-free is not by itself a virtue. The parameter-free Ladder margin needs no noise scale and is the worst rule after best-of-1 (0.265 false-accept): its guarantee is on leaderboard error, not on accepting a null candidate, so it has no type-I guarantee at any margin.

5.1 What does break a guarantee is a plug-in σ

The rules in Table 2 divide into two kinds, and we derive the division from the data rather than declaring it: recomputing every rule with σ estimated from three runs instead of known, exactly four rules move by more than two percentage points (both e-processes and the two Ladder margin rules), and every other row stays within 0.002, at most 1.1 standard errors of the independent draws the two columns use. Those four are precisely the rules that take a noise scale as an argument. The Welch, Bonferroni and AdaStop rows do not move because they never had a σ to get wrong: they standardise by the runs' own sample SD and their reference distribution pays for that estimate. The Ladder, a 2015 mechanism nobody thinks of as noise-sensitive, already exceeds α at $\eta = \sigma$ with σ known (0.054) and doubles with a three-run $\hat{\sigma}$ (0.118); at $\eta = 2\sigma$ it moves too but stays under α, so a wide margin trades power rather than level.

This is not an argument against measuring σ. Using $\hat{\sigma}$ to *size* an experiment fails visibly, in a run count the reader can see; using it *inside* the accept rule fails silently, in a level no output reveals.

Table 3: False-accept under a true null ($\Delta = 0$) by where the noise scale comes from, $\alpha = 0.05$, mean over 16 conditions at 20,000 trials per condition and source. "Infl." counts conditions above α by a one-sided z test, Bonferroni-corrected over the 16. The two gates are those of Table 2; calibrated t is a fixed-n test with a t_{k-1} critical value. Calibration runs are held out from the runs the test draws, unlike Table 2's $\hat{\sigma}$ column, so the two differ by a few thousandths; both come from the 30 runs that define the oracle, so every row is optimistic (Sec. 3).

		e-process, corrected			as written		calibrated t	
Source of σ	Extra	f.-acc.	Infl.	pw@0.5σ	f.-acc.	Infl.	f.-acc.	Infl.
oracle σ (all $n = 30$ runs)	–	0.002	0	0.018	0.037	0	0.049	0
incumbent's own 5 runs	–	0.027	1	0.088	0.077	11	0.040	2
calibration, 3 runs	3	0.079	16	0.180	0.142	16	0.049	6
calibration, 5 runs	5	0.032	0	0.099	0.089	16	0.047	5
calibration, 10 runs	10	0.012	0	0.052	0.059	11	0.046	1
calibration, 20 runs	20	0.006	0	0.031	0.044	1	0.043	0

We recommend the first and not the second. Table 3 prices the second. Under an oracle σ the corrected gate sits at 0.002. Reusing the incumbent's own five runs, the free option a loop would reach for, gives 0.027; a dedicated three-run calibration gives 0.079 and inflates all 16 conditions; five runs give 0.032. The corrected gate is conservative enough that a five-run $\hat{\sigma}$ spends its slack rather than the level; the gate as written has no slack and needs 20 runs to reach 0.044. The textbook fix is to carry $\hat{\sigma}$'s own uncertainty in the threshold, as the first stage of Stein's two-stage procedure does (Stein, 1945): a t test on an independent k-run $\hat{\sigma}$ with a t_{k-1} critical value. It averages 0.049 at $k = 3$, but six of 16 conditions still exceed α (worst 0.070) because the pools are not normal, and its price is power: 0.146 at 0.5σ against 0.217 with σ known. The direction is the trap: a short $\hat{\sigma}$ is usually low (its median undershoots σ), so a plug-in gate usually becomes *more eager*. At $k = 3$ the corrected gate's power rises from 0.018 to 0.180 along with its false-accept, and a reader comparing only power columns would draw the opposite conclusion from the truth.

6 THE FLOOR IS NOT A PROPERTY OF THE CODE

Everything above holds the hardware fixed. A loop that concludes "below threshold" has made a claim about a machine, and we can now say how much of one.

We evaluated fixed weights (no training, no quantization, identical model files) in a byte-identical container, repeatedly, on 7 NVIDIA GPU models spanning 5 compute capabilities, at 5 sites, over 3 models from two families and 3 tasks: 3,627 evaluations in 39 (model, task, GPU) cells (Tables 11 and 12, Appendix E). That breadth belongs to one model: only Llama-3.2-1B ran on all 7 cards and 5 sites, while the other two reach three cards at one site. The result is stronger than "scores differ across hardware", which is known (Hochlehnert et al., 2025; Yuan et al., 2025).

Nondeterminism is quantized by item. Every accuracy value in the archive is an exact integer multiple of $1/N$, where N is that task's eval-set size, with no exceptions out of 3,627 and a worst residual of 9.1×10^{-13} from the nearest whole count. Taken one task at a time that is nearly tautological: accuracy is a count over a fixed denominator. MMLU is the real test, because its single number spans 57 subject splits. It lands on the $1/14042$ lattice because the harness micro-averages the pooled questions; averaging 57 per-split accuracies would not land there. The mechanism is the same in both cases: the answer is an argmax, so it can change only where the kernels move the top two loglikelihoods by more than the gap between them. A score therefore cannot move continuously. Nor is that peculiar to these cross-hardware runs: recovering N for each pipeline task from its own 60 full-precision accuracies, as the smallest integer making every value in that task integral, every one of them is an exact item count too, to within 9.1×10^{-13}. Sec. 3 and this section move the same quantity. It moves in quanta of one question: 0.042 points on ARC-Easy ($N = 2376$), 0.0071 on MMLU ($N = 14042$). An accuracy of `0.6359427609` implies ten decimals the measurement does not have. The honest unit is an item count, 1511 of 2376 correct, which makes "is this one question or thirty?" immediately visible.

Some triples are pinned and others wander. 13 of 39 cells are *pinned*: every repetition returns one identical integer, over as many as 117 repetitions. The rest *wander* over a contiguous band, spanning at most 30 items between its extremes, and which of the two happens is not a property of the card: every pinned cell belongs to one of the two Blackwell parts (B200 in 6 of 9, RTX PRO 6000 in 7 of 9 of their cells), and the other five cards we ran never pin at all. That is not a site or a driver artifact: the B200, the RTX PRO 6000 and the L4 all run at Site B, on one driver, out of the same container, and two of them pin while the third never does. The card is close to necessary and not sufficient: the RTX PRO 6000 pinned at Llama-3.2-3B/ARC-Easy over 77 runs wanders at Llama-3.2-3B/HellaSwag.

Localising it: a card that does not perturb cannot move. That batch-dependent kernel reductions make LLM inference nondeterministic is established (He & Thinking Machines Lab, 2025; Badash et al., 2026; Yuan et al., 2025); what is open is where that enters an accuracy number and why it reaches some (model, task, card) triples and not others. The harness scores ARC-Easy and HellaSwag by an argmax over length-normalised loglikelihoods and MMLU over raw ones, so an item can change answer only where the kernels move those scores by more than the gap between its top two choices. We instrumented twenty arms (of the 27 combinations of three models, three tasks and three cards) at eight repetitions each, logging every per-choice loglikelihood at every repetition (559,357 per repetition), and rebuilt every item's correctness from them with no disagreement against the harness's own metric. We expected pinned arms to have wide margins. Instead the arms split on something prior to margins. Nine are bit-exact: across 254,844 loglikelihoods none differs between any two repetitions, and each returns exactly the integer its cell pins at in the archive, so logging did not change the computation. In the other eleven, 1.9% to 58.5% move, by up to 2.28 nats. That a bit-exact arm cannot flip is forced, not observed; the evidence is on the perturbing side.

Batching is necessary for it. These arms let vLLM schedule up to 16 sequences together, so what shares a kernel launch with an item can change from one repetition to the next. We re-ran four perturbing arms (Llama-3.2-1B on the L4 and Llama-3.2-3B on the B200, ARC-Easy and HellaSwag) changing one engine argument, `max_num_seqs=1`. All four became bit-exact: where 13.4% to 38.7% of loglikelihoods had moved, none of 99,338 now does. Bit-exact is not the same as agreeing with the batched number: the L4 on Llama-3.2-1B/ARC-Easy now pins at 1520 items, outside the 1512–1518 band of all 87 batched archive runs. The price is 1.9 to 3.1$\times$ the wall-clock. We did not log batch membership, so this does not separate varying composition from batch-size-dependent kernel selection; it shows batched execution is necessary.

The same card, bit-exact on one model and not on another. Holding the silicon fixed, both Blackwell parts split by model. On the B200, five arms on Llama-3.2-1B and gemma-2-2b are bit-exact across 155,506 loglikelihoods while two on Llama-3.2-3B move 14.5% and 25.3%; on the RTX PRO 6000 the same model moves 1.9% and 2.4% and the others nothing. Same part, site, driver, container and harness; what changes is the model, and with it the shape of every matrix the kernels see. Determinism here is a property of the computation, not of the part number.

The margin then decides *which* items move. Taking each arm's largest measured shift as the reach of the perturbation, every flip falls inside the at-risk set it defines; we take the threshold from the shift rather than the flips, since the largest flipped margin would contain them by construction. Containment is the weak half: pooled, 237 of 14,684 at-risk items flipped, a precision of 1.6%. Of those flips, 169 (71%) cross the correctness boundary, and ten of the eleven perturbing arms moved their score. The exception is the RTX PRO 6000 on Llama-3.2-3B/ARC-Easy: 2.4% of its loglikelihoods move and no normalised argmax changes, matching the 1694 it returns on all 77 archive runs, so bit-exactness is sufficient for a pinned score but not necessary. MMLU adds a case margins alone hide: each choice is one token, and in bf16 5.8% of items tie exactly between two choices; 96 of the 111 flips of Llama-3.2-1B on the L4 are such ties broken the other way.

So a discrete argmax is why a score moves in whole items, and three things decide whether it moves: whether batching perturbs the scores, whether an item's margin lies inside that perturbation, and whether the flip crosses the correctness boundary. Only the first is a lever a practitioner holds, at a price in throughput. This rests on twenty arms and four interventions: enough to establish the chain where we measured it, not to predict which triples pin elsewhere.

The vendor's condition is met, and it is not the one that matters. NVIDIA guarantees bit-wise cuBLAS reproducibility only across GPUs with "the same architecture *and* the same number of SMs" (NVIDIA, 2026), and AMD states a narrower version for rocBLAS (AMD, 2026). Provenance records the compute capability and `multi_processor_count` of every card, so both halves are checkable. The A100-PCIe and A100-SXM4 both report 108 SMs: same architecture, same SM count, one container (the drivers differ). Yet neither reproduces its own number, so no pair of runs is guaranteed to match: the guarantee covers repeated identical calls, and batched serving does not issue them. The L4 and L40S report 58 and 142 and never met the condition. The condition a practitioner can check does not govern this number.

Two cards that never vary can still disagree. Of the six cross-architecture pairs where both arms pin, the widest in items is gemma-2-2b/HellaSwag: the `sm_100` B200 pinned at 7271 items and the `sm_120` RTX PRO 6000 at 7291, 20 items apart, 0.199 accuracy points, with *zero* within-arm variance on either side, over 93 and 101 repetitions. That offset is 2.4× the tightest replicate SD we measure anywhere in Sec. 3: two loops running the same code on two cards would disagree about the same candidate, reproducibly.

7 What to do instead

1. **Measure the floor before trusting any comparison.** Five runs is a draw, not a measurement: 9 of our 16 five-replicate SDs fall outside the bootstrap interval of the $n = 30$ estimate. Buy compute for the floor first, not last.
2. **Use a test that carries its own variance estimate.** A Welch t on $k = 5$ had 3.5× the power of the best valid e-process we tried (0.170 against 0.048 at $\Delta = 0.5\sigma$), for fewer runs and no external σ. An e-process against a sampled incumbent must run on the information clock; as naturally written it is not anytime-valid.
3. **If you must plug in a σ, pay for its error in the threshold.** Three runs take the corrected gate from 0.002 to 0.079; a t_{k-1} critical value keeps the average near α at a cost in power.
4. **State the hardware, and report items.** "Below threshold" is a claim about a machine, and ten decimals of accuracy is a claim the measurement cannot support; 1511 of 2376 can be checked.

8 Limitations

The floor is measured on *one* pipeline and one model family at depth, so what transfers is the method rather than the number. Neither a multi-seed noise floor nor a loss-to-downstream decoupling is new (Zhao et al., 2026); the floor of this kind of pipeline, and its ratio to the effect chased, is. Freezing data synthesis makes our σ an underestimate. The survey establishes what is *coded*; both replays resample real pools and score single decisions, not a loop's sequence of them; we replay one known-scale e-process and no scale-invariant one; and the localisation does not predict which triples pin elsewhere, while the batch-size intervention covers four arms on two cards. The replay treats σ as fixed for a campaign, but it moved up to 7.96-fold within one task when only the optimizer changed, so a loop whose edits change the pipeline must re-measure it. Reporting items applies to count-based metrics; a continuous or sampled metric has no lattice.

9 Conclusion

An autonomous research loop is a measuring instrument that acts on its own readings, and no one would trust an instrument whose noise had never been measured. For the pipeline we re-ran, the noise was larger than the effect being chased, varied more than 13-fold between settings, and changed with the GPU the evaluation ran on. Against that measured noise the choice of test mattered less than the literature suggests: a five-run Welch test did better than the anytime-valid gates, and the reliable way to break a guarantee was to hand a rule a noise scale estimated from too few runs. Before tuning the accept rule, re-run the unchanged pipeline on the machine the loop will use, and let the test carry its own estimate of the noise. The same holds wherever a single number gates a decision: safety evaluations, leaderboard submissions, and the regression checks that decide whether a model ships.

AI USE STATEMENT

The authors used an AI coding assistant for software engineering and analysis support: writing and debugging the replication, harvesting and analysis scripts, porting estimators between them, drafting and revising text, and operating the build that generates this manuscript. Every experimental result reported here comes from executing the pipeline and the evaluation harness on real hardware; no result, table entry, figure or citation was produced by a language model. The authors read every primary source cited in the survey of Sec. 4 themselves, at the commit or in the full text, and take full responsibility for the entire contents of this paper, including any part drafted with AI assistance.

REPRODUCIBILITY STATEMENT

The manuscript is generated: a single script reads the committed run-level CSVs, the result JSON archive and the analysis outputs, computes every number in the text, the tables and the figure, and emits the LaTeX source. The replication CSVs (480 runs), the recovered validation losses (480 values), the cross-hardware result archive (3,799 JSON files: the 3,627 counted here and 172 from the instrumented and ROCm arms, each arm with a provenance file recording GPU model, driver and container digest), the per-arm outputs of the loglikelihood analysis, the accept-rule replays with their self-check against a reference implementation, the scripts behind every analysis output, and the survey's per-system evidence at full commit hashes are included in the supplementary material. The per-sample loglikelihood logs behind Sec. 6 (559,357 values per repetition, several gigabytes) will be released with the camera-ready. The generator asserts, at build time, that the paired harvest agrees with the replication CSV on every run, that the survey tables agree with the survey document's own counts, and that every accuracy in the cross-hardware archive is an exact multiple of $1/N$; the build fails rather than emitting a number if any of these drift.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL https://arxiv.org/abs/1907.10902.

AMD. rocBLAS documentation: Bitwise reproducibility, 2026. URL https://rocm.docs.amd.com/projects/rocBLAS/en/latest/how-to/what-is-rocblas.html. Bitwise reproducible results require an identical GFX target ISA, a single HIP stream active per rocBLAS handle, identical ROCm versions, and atomic operations not allowed; no guarantee is given across different GPU architectures or different compute-unit counts.

Badash, Boneh, Komargodski, and Srivastava. Hawkeye: Reproducing GPU-level non-determinism. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026. URL https://arxiv.org/abs/2603.20421.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2024. URL https://arxiv.org/abs/2405.14782.

Avrim Blum and Moritz Hardt. The ladder: A reliable leaderboard for machine learning competitions. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1006–1014. PMLR, 2015. URL https://proceedings.mlr.press/v37/blum15.html.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems 3 (MLSys)*, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/hash/0184b0cd3cfb185989f858a1d9f5c1eb-Abstract.html.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

Tianyu Ding, Aditya Nannapaneni, Bingfan Liu, and Ling Zhang. Autonomous research agents: A survey of AI scientists and the verification gap, 2026. URL https://arxiv.org/abs/2608.05179.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, pp. 2185–2194, 2019. doi: 10.18653/v1/D19-1224.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015. doi: 10.1126/science.aaa9375.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7(39):1079–1105, 2006.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024. URL https://zenodo.org/records/12608602.

Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference, September 2025. URL https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/. Blog post.

David Heineman, Valentin Hofmann, Ian Magnusson, Yuling Gu, Noah A. Smith, Hannaneh Hajishirzi, Kyle Lo, and Jesse Dodge. Signal and noise: A framework for reducing uncertainty in language model evaluation, 2025. URL https://arxiv.org/abs/2508.13144.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3207–3214, 2018. doi: 10.1609/aaai.v32i1.11694.

Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.07086.

Steven R. Howard, Aaditya Ramdas, Jon McAuliffe, and Jasjeet Sekhon. Time-uniform, nonparametric, nonasymptotic confidence sequences. *The Annals of Statistics*, 49(2):1055–1080, 2021. doi: 10.1214/20-AOS1991.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel Candès, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, 2025. URL https://proceedings.mlr.press/v267/huang25n.html.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

Amit Jain and Richard Linares. Agentic AutoResearch for space autonomy: An auditable, LLM-driven research agent for aerospace control problems, 2026. URL https://arxiv.org/abs/2606.20394.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch. Created 6 March 2026; accessed August 2026.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2015. URL https://arxiv.org/abs/1412.6980.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023. doi: 10.1145/3600006.3613165.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Lovish Madaan, Aaditya K. Singh, Rylan Schaeffer, Andrew Poulton, Sanmi Koyejo, Pontus Stenetorp, Sharan Narang, and Dieuwke Hupkes. Quantifying variance in evaluation benchmarks, 2024. URL https://arxiv.org/abs/2406.10229.

Oded Maron and Andrew W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems 6 (NIPS)*, 1993.

Timothée Mathieu, Riccardo Della Vecchia, Alena Shilova, Matheus Medeiros Centa, Hector Kohler, Odalric-Ambrym Maillard, and Philippe Preux. AdaStop: Adaptive statistical testing for sound comparisons of deep RL agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lXyZr9TLEU.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations, 2024. URL https://arxiv.org/abs/2411.00640.

Jingjie Ning, Shanshan Zhong, Xiaochuan Li, Ji Zeng, and Chenyan Xiong. One run is not an idea: The implementation lottery in automated research, 2026. URL https://arxiv.org/abs/2607.26587.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA. cuBLAS documentation: Results reproducibility, 2026. URL https://docs.nvidia.com/cuda/cublas/index.html. "all cuBLAS API routines from a given toolkit version generate the same bit-wise results at every run when executed on GPUs with the same architecture and the same number of SMs".

Aaditya Ramdas, Peter Grünwald, Vladimir Vovk, and Glenn Shafer. Game-theoretic statistics and safe anytime-valid inference. *Statistical Science*, 38(4):576–601, 2023. doi: 10.1214/23-STS894.

Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 338–348, 2017. doi: 10.18653/v1/D17-1035.

Xingyu Ren, Youran Sun, Chugang Yi, Kejia Zhang, Jiaxuan Guo, Jianda Du, and Haizhao Yang. What's missing in autonomous research? a systematization of systems, benchmarks, and verification, 2026. URL https://haizhaoyang.github.io/research/autoresearch-survey.html.

Herbert Robbins. Statistical methods related to the law of the iterated logarithm. *The Annals of Mathematical Statistics*, 41(5):1397–1409, 1970. doi: 10.1214/aoms/1177696786.

Holger R. Roth, Ziyue Xu, Chester Chen, Daguang Xu, Peter Cnudde, and Andrew Feng. Auto-FL-Research: Agentic search for federated learning algorithms, 2026. URL https://arxiv.org/abs/2607.01366.

D. Sculley, Jasper Snoek, Alexander B. Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2018. URL https://openreview.net/forum?id=rJWF0Fywf.

Biswa Sengupta. Self-evolving agents with anytime-valid certificates, 2026. URL https://arxiv.org/abs/2607.00871.

Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents, June 2026. URL https://arxiv.org/abs/2606.08106. arXiv preprint; no institutional affiliation or venue stated.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, 2025. URL https://arxiv.org/abs/2504.01848.

Charles Stein. A two-sample test for a linear hypothesis whose power is independent of the variance. *The Annals of Mathematical Statistics*, 16(3):243–258, 1945. doi: 10.1214/aoms/1177731088.

B. L. Welch. The generalization of 'Student's' problem when several different population variances are involved. *Biometrika*, 34(1–2):28–35, 1947. doi: 10.1093/biomet/34.1-2.28.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI r&D capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, 2025. URL https://arxiv.org/abs/2411.15114.

Liwei Wu and Cho-Jui Hsieh. Closing the auto-research loop: An AI co-scientist for production search ranking, 2026. URL https://arxiv.org/abs/2603.22376.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference, 2025. URL https://arxiv.org/abs/2506.09501.

Yang Zhao, Jiahao Lu, Bin Huang, Guhua Zhang, and Jie Zhou. Most transformer modifications still do not transfer at 1-3B: A 2020-2026 update to narang et al. (2021) with downstream evaluation and a noise floor, 2026. URL https://arxiv.org/abs/2605.20798.

A PER-CONDITION NOISE FLOOR AT $n = 30$

Table 4: Every condition at $n = 30$, tightest to noisiest, in accuracy points, with 10,000-sample bootstrap 95% CIs on the SD (same estimator, B and seed as Figure 1).

Condition	n	Mean	SD	95% CI on SD
HS/muon	30	71.76	0.082	[0.060, 0.102]
SIQA/muon	30	47.23	0.106	[0.082, 0.126]
ARC-e/muon	30	70.97	0.110	[0.077, 0.137]
CQA/muon	30	73.99	0.127	[0.096, 0.152]
PIQA/adam	30	77.29	0.127	[0.096, 0.151]
HS/adam	30	71.55	0.154	[0.110, 0.186]
PIQA/muon	30	77.14	0.156	[0.109, 0.195]
OBQA/muon	30	39.40	0.174	[0.129, 0.213]
WG/muon	30	68.71	0.198	[0.157, 0.230]
CQA/adam	30	73.77	0.225	[0.176, 0.263]
WG/adam	30	68.57	0.252	[0.182, 0.310]
SIQA/adam	30	47.07	0.299	[0.210, 0.369]
ARC-e/adam	30	70.32	0.449	[0.354, 0.515]
OBQA/adam	30	38.70	0.565	[0.466, 0.636]
MMLU/muon	30	61.08	1.086	[0.686, 1.411]
MMLU/adam	30	49.60	8.644	[6.578, 10.188]

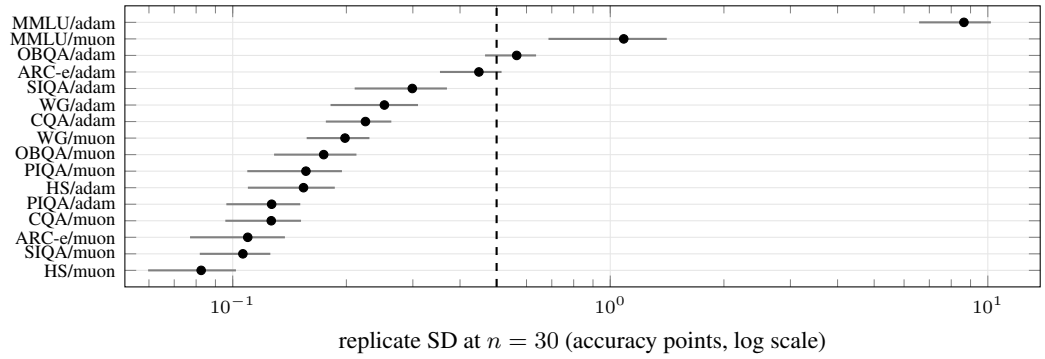

Figure 1: The measured floor, all 16 conditions, with 10,000-sample bootstrap 95% CIs on the SD. The dashed line is a 0.5-point proposal; conditions to its right have a replicate SD larger than that gain, and since a one-run comparison has SD $\sqrt{2}\sigma$, most to its left cannot resolve it either. The axis is logarithmic because the spread is 105×.

B PER-CONDITION PROXY CORRELATION AT $n = 30$

THE OPTIMIZED PROXY RUNS BACKWARDS

The pipeline's search minimises a validation loss and the paper reports benchmark accuracy, so the search is only as good as the agreement between the two. If the proxy worked, lower loss would go with higher accuracy and the within-condition correlation would be *negative*. Pooling the 480 runs after standardising both quantities inside each condition, it is $r = +0.276$ (cluster-permutation $p < 10^{-4}$, $B = 20{,}000$): replicates whose search reached a lower validation loss tend to score lower. 12 of 16 conditions carry the wrong sign, 8 of them significantly, against 3 significantly in the right direction. The pooled sign survives dropping MMLU ($p < 10^{-4}$), but a cluster bootstrap over conditions then spans zero: well resolved within conditions, and split by optimizer across them (Table 5).

In a paper whose thesis is that five replicates are not a measurement, the depth matters here too: on the 80-run sub-block at $n = 5$ per condition the same analysis gives $r = +0.328$ ($p = 0.0072$). Recovering `best_val_loss` for all 480 replicates of the deep set, with no re-execution, moves it to $n = 30$ per condition: a smaller effect, far better resolved. That is what applying our own standard to our own claim costs, and we report both.

Table 5: Within-condition Pearson correlation between the search's own objective (best validation loss) and the reported accuracy, at $n = 30$. The search minimises the objective, so a working proxy gives $r < 0$. 12 of 16 conditions carry the wrong sign. Pooled, $r = +0.276$ (cluster-permutation $p < 10^{-4}$); dropping MMLU gives $r = +0.188$ ($p < 10^{-4}$) and dropping MMLU with SIQA gives $r = +0.167$ ($p = 0.0017$), so the sign is not an artifact of the collapsed-basin condition. A 10,000-sample cluster bootstrap over conditions is less sure: with MMLU dropped its 95% interval is $[-0.048, +0.420]$, which covers zero.

Condition	n	r	p
ARC-e/adam	30	$+0.661$	<0.001
ARC-e/muon	30	$+0.521$	0.003
CQA/adam	30	$+0.544$	0.002
CQA/muon	30	-0.034	0.858
HS/adam	30	$+0.694$	<0.001
HS/muon	30	-0.484	0.007
MMLU/adam	30	$+0.916$	<0.001
MMLU/muon	30	$+0.859$	<0.001
OBQA/adam	30	$+0.843$	<0.001
OBQA/muon	30	-0.595	0.001
PIQA/adam	30	$+0.006$	0.976
PIQA/muon	30	-0.417	0.022
SIQA/adam	30	$+0.600$	<0.001
SIQA/muon	30	$+0.032$	0.869
WG/adam	30	$+0.013$	0.947
WG/muon	30	$+0.257$	0.171

C ACCEPT-RULE SURVEY: PER-SYSTEM EVIDENCE

Table 6 is the audit trail behind every verdict in Sec. 4 and Tables 7, 8 and 9: for each of the 26 surveyed systems, the exact artifact we read and the exact lines the accept rule was determined from. 19 of the 26 were read from code at a pinned commit, giving 76 file and line references; for the remaining 7 no code was inspected, and the rule was read from the method section of the identified preprint. Commit hashes are abbreviated to ten characters here and given in full in the survey document released with this paper. No verdict rests on a search snippet or on a model's description of a repository; where a `README` appears below it is quoted at a line number, as the project's own statement about its behaviour.

Table 6: Per-system evidence for the accept-rule survey. "Source of record" is the commit inspected (with its commit date) or, where no code is published, the preprint read in full. "Evidence" lists the file and line references the verdict rests on.

System	Source of record	Evidence
karpathy/autoresearch	`228791fb49`, 2026-03-25	`program.md:103-104`, `program.md:99`, `program.md:77`, `train.py:458-459`, `train.py:337`
trevin-creator/autoresearch-mlx	`766a25ff22`, 2026-07-02	`rigor.py:120-122`, `rigor.py:79-88`, `rigor.py:134-135`, `rigor.py:110-113`
drivelineresearch/autoresearch-claude-code	`e05758fa8c`, 2026-09-07	`README.md:11`, `skills/autoresearch/SKILL.md:64-67`, `skills/autoresearch/SKILL.md:123-125`, `skills/autoresearch/references/state.md:27`, `SKILL.md:130-131`, `SKILL.md:132-133`
davebcn87/pi-autoresearch	`939ede8220`, 2026-09-10	`README.md:11`, `skills/autoresearch-create/SKILL.md:141`, `extensions/pi-autoresearch/index.ts:426-456`, `index.ts:808-811`, `jsonl.ts:29,124`, `index.ts:2364-2371`, `README.md:280`
AIDE / aideml	`60b3978ddf`, 2026-09-03	`aide/journal.py:216-227`, `aide/utils/metric.py:48-49`, `aide/agent.py:190-193`, `aide/agent.py:391-394`, `aide/agent.py:237-240`, `aide/utils/config.yaml:38`, `aide/agent.py:413-435`
AI-Scientist v1 (Sakana), arXiv:2408.06292	`1de1dbc1f4`, 2025-12-19	`ai_scientist/perform_experiments.py:66-78`
AI-Scientist-v2 (Sakana), arXiv:2504.08066	`96bd51617c`, 2025-12-19	`ai_scientist/treesearch/journal.py:420-502`, `journal.py:444-445`, `journal.py:430`, `journal.py:496-497`
MLAgentBench	`5d71205cc2`	`agent_research.py:39-40`, `high_level_actions.py:78,117`, `high_level_actions.py:237`, `benchmarks/feedback/env/train.py:9-13`
MLE-STAR (Google), arXiv:2506.15692	`63bfb33c4d`	`sub_agents/refinement/agent.py:48-84`, `shared_libraries/config.py:35`, `config.py:24`, `sub_agents/initialization/agent.py:279`
R&D-Agent (Microsoft)	`4834df2b6f`, 2026-09-15	`rdagent/core/proposal.py:182,316`, `scenarios/data_science/dev/runner/__init__.py:207-217`, `app/data_science/conf.py:193`, `proposal/exp_gen/prompts_v2.yaml:724-726`
ML-Master	`6a7bf55007`, 2026-03-30	`search/node.py:2`, `search/journal.py:60-68`, `search/mcts_node.py:37-55`, `agent/mcts_agent.py:790-806`, `agent/mcts_agent.py:845`
Agent Laboratory	`d9017d90e3`, 2025-08-20	`mlesolver.py:295-316`, `mlesolver.py:213,215`
AIRA-dojo (Meta / FAIR)	`c795d8649d`, 2025-09-26	`src/dojo/core/solvers/utils/journal.py:241-245`, `src/dojo/core/solvers/utils/metric.py:50`, `src/dojo/solvers/mcts/mcts.py:76`, `src/dojo/analysis_utils/meta_data_wrangling.py:1699`, `src/dojo/utils/logger.py:354`
Darwin Gödel Machine	`a565fd2d1d`, 2025-08-13	`DGM_outer.py:178-185`, `DGM_outer.py:234`, `DGM_outer.py:238`, `DGM_outer.py:232`
OpenEvolve	`411fb59c88`, 2026-07-18	`openevolve/database.py:1139`, `openevolve/config.py:385`, `config.py:389`
AlphaEvolve (Google DeepMind), arXiv:2506.13131	arXiv:2506.13131	*no code inspected; read from the method section*

Table 6, continued

System	Source of record	Evidence
Curie, arXiv:2502.16069	`db1b1f5615`, 2025-09-28	`curie/nodes/exec_validator.py:59`, `exec_validator.py:79`
Wu & Hsieh, "Closing the Auto-Research Loop: An AI Co-Scientist for Production Search Ranking", arXiv:2603.22376	arXiv:2603.22376	*no code inspected; read from the method section*
MLE-bench (OpenAI)	`507f92e113`, 2026-04-24	*read at the stated commit; references in the survey document*
METR RE-Bench	`93b98062e5`, 2025-01-30	`metr/task_protected_scoring/scoring.py:69-73`, `ai_rd_triton_cumsum.py:48`, `ai_rd_triton_cumsum/assets/score.py:29`
PaperBench (OpenAI)	`51052cede8`, 2026-04-21	`paperbench/judge/graded_task_node.py:145-153`, `simple.py:679`, `paperbench/reproduce.py:56`, `reproduce.py:325-330`, `paperbench/metrics.py:97-99`, `metrics.py:140`
Jain & Linares, arXiv:2606.20394	arXiv:2606.20394	*no code inspected; read from the method section*
Roth et al. (NVIDIA), arXiv:2607.01366	arXiv:2607.01366	*no code inspected; read from the method section*
PACE, arXiv:2606.08106	arXiv:2606.08106	*no code inspected; read from the method section*
Sengupta, arXiv:2607.00871	arXiv:2607.00871	*no code inspected; read from the method section*
POPPER (Huang, Jin, Li, Li, Candès, Leskovec), arXiv:2502.09858, ICML 2025	arXiv:2502.09858	*no code inspected; read from the method section*

Table 7: How autonomous research loops decide to keep a candidate change. Every rule was read from primary source (repository at the stated commit, or the paper's method section) on 2026-09-16/17; per-system file and line evidence is in Table 6. k is runs per candidate *before* the accept decision. *Var.* marks whether run-to-run variability enters the accept decision, by an estimate or by construction; ∘ marks a variance estimate that is computed but never reaches it. *Enf.* distinguishes a rule enforced by code from one stated in a prompt file the agent is asked to follow; "paper" means the rule was read from a method section and no code was inspected. The last row is the only surveyed accept rule that estimates its noise scale from replicates, at 20 runs of overhead.

System	Accept rule	k	Var.	Mult.	Enf.
Community autoresearch loops					
`karpathy/autoresearch`	`val_bpb` lower ⇒ keep; else reset	1	–	–	prompt
`autoresearch-mlx`	bootstrap $P(\text{better}) \geq 0.95$, unused[a]	1	∘	–	prompt
`autoresearch-claude-code`	gain > stated `noiseFloor`	1	–	–	prompt
Research agents					
AI-Scientist (v1)	LLM reads means; no rule	3	∘	–	prompt
AIDE	greedy $\arg\max$ scalar metric	1	–	–	code
Darwin Gödel Machine	$\geq$ incumbent $- 0.1$ (constant)	1	–	–	code
Deployed in production					
Wu & Hsieh (search ranking)	best single offline number	1	–	–	paper
Proposed accept rule					
Jain & Linares	$\mu_R - \mu_0 \geq 2\sigma$ (directional)	10+10	✓[b]	–	paper

Two notes change how a row is read: [a] a 1–3-run bootstrap gate exists in code, but the loop's instruction file never invokes it and keeps the single-run rule; its "seeds" are repeats at a fixed seed, exploiting GPU nondeterminism, and the incumbent is never re-run; [b] the 10 baseline seeds and 10 verification reseeds are disjoint, so the 20 runs are overhead per accepted candidate, before that candidate's own runs.

Table 8: Accept rules, remaining deployed loops (continuation of Table 7). Together with the seven deployed loops there, these eleven make up the 18 systems the prevalence counts are taken over.

System	Accept rule	k	Var.	Mult.	Enf.
`pi-autoresearch`	primary metric improved	1	○[a]	–	prompt
AI-Scientist-v2	LLM selects best tree node	1[b]	○[b]	–	prompt
MLAgentBench	none; LLM scratchpad, edits always applied	1	–	–	prompt[c]
MLE-STAR	`best_improvement <= 0.0` ⇒ revert	1	–	–	code[d]
R&D-Agent	LLM answers "Replace Best Result"	1	–	–	prompt[d]
ML-Master	UCT; reward $+1$ iff improvement > 0	1	–	–	code[e]
Agent Laboratory	$\arg\max$ of an LLM reward score	1	–	–	code[f]
AIRA-dojo (Meta)	$\arg\max$ scalar node metric	1	–	–	code
OpenEvolve	`fitness1 > fitness2`	1	–	–	code
AlphaEvolve	scalar score in evolutionary DB	1[g]	–	–	paper
Curie	LLM compares two run logs as text	2[h]	–	–	prompt

[a] MAD-based confidence score, displayed only; the documentation calls it advisory and says it never auto-discards. The MAD is taken across configurations rather than replicates, so it mixes the between-condition effect into the noise estimate, and carries no 1.4826 consistency constant despite being compared against "$1\times$/$2\times$" thresholds. [b] Three seeds *are* run (`multi_seed_eval: num_seeds: 3`), but only *after* the winning node has been selected, so they cannot correct the selection; the code comment says they are run "to get statistical metrics" and no standard deviation is computed anywhere in the repository. [c] The purest prompt-only case: no metric comparison exists in code, and `edit_script` writes every edit unconditionally. Reverting is a voluntary agent action the prompt merely suggests. [d] MLE-STAR and R&D-Agent are the cleanest code-versus-prompt contrast in the survey: the same problem resolved by a hard zero threshold in one and by the undefined adjectives "obviously better"/"similar" in the other, with near-ties broken on code aesthetics. [e] Reward $\in \{-1, 1, 2\}$, so a candidate that wins by noise is rewarded identically to one that wins by a wide margin; one `if`/`else` branch pair is inert (both add 1). [f] The score is a float an LLM is asked to invent, sampled once at temperature 0.6; the source carries the author's comment `# todo: have a reward function here`. With `max_codes = 1` the "best of k" is best-of-one. [g] Multiple random seeds in the tensor-decomposition application only, aggregated as best-of-k. The stage named "evaluation cascade (hypothesis testing)" is a test-case difficulty cascade containing no statistical test. [h] The control experiment is rerun exactly once (`iterations = 1`); the programmatic comparator is commented out and both result files are handed to an LLM as free text.

Table 9: Benchmarks that define the evaluation protocol, and proposed accept rules. These are not deployed loops and are excluded from the prevalence counts. "seq." denotes a sequential, anytime-valid procedure with no fixed number of runs per candidate.

System	Accept / scoring rule	k	Var.	Mult.	Enf.
Benchmarks					
MLE-bench	≥ 3 seeds, mean $\pm$ SEM[a]	3	✓	–	protocol
RE-Bench	max/min over all score-log entries[b]	1	–	–	code
PaperBench	LLM judge; weighted rubric roll-up	1	○[c]	–	code
Proposed accept rules					
Roth et al. (Auto-FL-Research)	single-seed campaign best[d]	1	–	–	paper
PACE	$E \geq 1/\alpha$, paired e-process	seq.	✓	–[e]	paper
Sengupta (SEA)	LCB $\geq -\varepsilon_{\text{tol}}$	1	✓	✓[f]	paper
POPPER	$E_i \geq 1/\alpha$, sequential falsification	seq.	✓[g]	–	paper

[a] The benchmark requires ≥ 3 replicates *of the agent* and reports mean $\pm$ one standard error; the agent scaffolds bundled in the same repository require none of themselves, and a grep of that directory for any variance vocabulary returns nothing. [b] Six of seven tasks call `get_best_score` with `select_best_fn=max` or `min` over every scoring call the agent chose to make, and the task instructions tell the agent so: "Your final score will be the lowest score (fastest time) you have achieved." This is uncorrected optimisation against measurement noise written into the benchmark's own scoring function. A grep of the entire repository for noise or variance vocabulary returns a single hit, and it concerns anti-cheating rather than statistics. [c] The only surveyed artifact that computes a standard error correctly (`np.std(scores, ddof=1) / np.sqrt(len(scores))`), and it is used for reporting, never to gate an accept decision. The contrast with AI-Scientist's `np.std(v)/len(v)` is exact: the correct line exists in a sibling repository from the same organisation and year. [d] Five-seed repeats exist but are explicitly post-hoc: "we treat single-seed campaign bests as selected candidates rather than final statistical claims" and "should not be interpreted as confirmatory hypothesis tests". [e] Per-decision only, and stated as such: "This is not a run-level familywise bound." [f] The only system in the survey with a multiplicity correction: a harmonic spending schedule $\delta_k = \delta_0/(Zk\log^2(k{+}1))$ giving familywise error over unboundedly many edits. [g] Distribution-free by construction (a p-to-e calibrator); requires no variance estimate.

D WHAT THE E-PROCESS'S EARLY STOPPING IS WORTH

Table 10: What the e-process's early stopping is worth, at oracle σ, for the corrected gate and the gate as written. "n" is the smallest fixed n per arm at which a one-sided Welch test, simulated on the real pools, matches the gate's power at that effect; "–" means the gate's power is below α, so a valid Welch test at any n beats it. Compare n with the gate's mean candidate runs; both also pay for an incumbent. An autonomous loop spends nearly all of its time in the top rows.

	corrected			as written		
Δ	power	runs	n	power	runs	n
0.10σ	0.003	7.99	–	0.053	7.87	4
0.25σ	0.005	7.98	–	0.091	7.77	5
0.50σ	0.018	7.95	–	0.182	7.51	6
1.00σ	0.108	7.67	3	0.484	6.47	6
2.00σ	0.690	5.35	4	0.962	3.47	7

E CROSS-HARDWARE ARCHIVE

The 3,627 result files this paper counts come from 5 sites, all produced by one container image; the supplement's archive also carries the 172 files of the instrumented arms and of an AMD/ROCm arm. The ROCm arm is excluded from this paper: it needs a second, ROCm-built image, and that would give up the single-image control the comparison rests on. Each arm carries a `provenance.txt` recording the GPU model as reported by the driver, the container digest and the harness version; Tables 11 and 12 key on those rather than on directory names. Site names are replaced by stable anonymous labels for review. N is read from each result file's own `n-samples` field and never hardcoded, and a cell whose repetitions disagree on N is excluded rather than summarised, because its arms would not be comparable. MMLU's $N = 14042$ is the micro-average over all 57 subject splits; ARC-Easy is $N = 2376$ and HellaSwag $N = 10042$.

Table 11: Cross-hardware evaluation of *fixed* weights in one container image: 3,627 evaluations, 39 cells, 5 sites, grouped by the compute capability the driver reported. Each cell is the range of correct-answer counts out of N over every repetition: one **bold** integer where they all agreed, "–" where we did not run it. Table 12 gives repetition counts and sites, and omits no cell.

		sm_80		sm_89		sm_90	sm_100	sm_120
Model	Task	A100-PCIe	A100-SXM4	L4	L40S	H100-SXM5	B200	RTX PRO 6000
Gemma-2-2B	ARC-e	–	–	1862–1868	–	–	**1864**	**1858**
Gemma-2-2B	HellaSwag	–	–	7277–7289	–	–	**7271**	**7291**
Gemma-2-2B	MMLU	–	–	7966–7983	–	–	**7985**	**7983**
Llama-3.2-1B	ARC-e	1510–1514	1510–1515	1512–1518	1510–1516	1508–1514	**1511**	**1514**
Llama-3.2-1B	HellaSwag	6190–6195	6191–6196	6188–6201	6181–6199	6188–6192	**6189**	**6186**
Llama-3.2-1B	MMLU	6776–6796	6777–6793	6786–6807	6766–6790	6764–6775	**6783**	**6800**
Llama-3.2-3B	ARC-e	–	–	1687–1692	–	–	1686–1690	**1694**
Llama-3.2-3B	HellaSwag	–	–	7181–7200	–	–	7191–7196	7186–7189
Llama-3.2-3B	MMLU	–	–	8717–8747	–	–	8731–8744	8722–8728

Table 12: The long form of Table 11: every (model, task, GPU) cell with its site and the number of repetitions behind it. "items" is the range of correct-answer counts observed, as integers out of N, and "Arch." is the compute capability the driver reported.

Model	Task	GPU	Arch.	Site	reps	items
Gemma-2-2B	ARC-e	B200	sm_100	Site B	53	1864 (pinned)
Gemma-2-2B	ARC-e	RTX PRO 6000	sm_120	Site B	53	1858 (pinned)
Gemma-2-2B	ARC-e	L4	sm_89	Site B	156	1862–1868
Gemma-2-2B	HellaSwag	B200	sm_100	Site B	93	7271 (pinned)
Gemma-2-2B	HellaSwag	RTX PRO 6000	sm_120	Site B	101	7291 (pinned)
Gemma-2-2B	HellaSwag	L4	sm_89	Site B	108	7277–7289
Gemma-2-2B	MMLU	B200	sm_100	Site B	117	7985 (pinned)
Gemma-2-2B	MMLU	RTX PRO 6000	sm_120	Site B	115	7983 (pinned)
Gemma-2-2B	MMLU	L4	sm_89	Site B	95	7966–7983
Llama-3.2-1B	ARC-e	B200	sm_100	Site B	107	1511 (pinned)
Llama-3.2-1B	ARC-e	RTX PRO 6000	sm_120	Site B	81	1514 (pinned)
Llama-3.2-1B	ARC-e	A100-PCIe	sm_80	Site C	120	1510–1514
Llama-3.2-1B	ARC-e	A100-SXM4	sm_80	Site E	45	1510–1515
Llama-3.2-1B	ARC-e	L4	sm_89	Site B	87	1512–1518
Llama-3.2-1B	ARC-e	L40S	sm_89	Site A	44	1510–1516
Llama-3.2-1B	ARC-e	H100-SXM5	sm_90	Site D	221	1508–1514
Llama-3.2-1B	HellaSwag	B200	sm_100	Site B	53	6189 (pinned)
Llama-3.2-1B	HellaSwag	RTX PRO 6000	sm_120	Site B	53	6186 (pinned)
Llama-3.2-1B	HellaSwag	A100-PCIe	sm_80	Site C	120	6190–6195
Llama-3.2-1B	HellaSwag	A100-SXM4	sm_80	Site E	40	6191–6196
Llama-3.2-1B	HellaSwag	L4	sm_89	Site B	132	6188–6201
Llama-3.2-1B	HellaSwag	L40S	sm_89	Site A	55	6181–6199
Llama-3.2-1B	HellaSwag	H100-SXM5	sm_90	Site D	113	6188–6192
Llama-3.2-1B	MMLU	B200	sm_100	Site B	113	6783 (pinned)
Llama-3.2-1B	MMLU	RTX PRO 6000	sm_120	Site B	89	6800 (pinned)
Llama-3.2-1B	MMLU	A100-PCIe	sm_80	Site C	120	6776–6796
Llama-3.2-1B	MMLU	A100-SXM4	sm_80	Site E	35	6777–6793
Llama-3.2-1B	MMLU	L4	sm_89	Site B	128	6786–6807
Llama-3.2-1B	MMLU	L40S	sm_89	Site A	55	6766–6790
Llama-3.2-1B	MMLU	H100-SXM5	sm_90	Site D	111	6764–6775
Llama-3.2-3B	ARC-e	B200	sm_100	Site B	105	1686–1690
Llama-3.2-3B	ARC-e	RTX PRO 6000	sm_120	Site B	77	1694 (pinned)
Llama-3.2-3B	ARC-e	L4	sm_89	Site B	48	1687–1692
Llama-3.2-3B	HellaSwag	B200	sm_100	Site B	80	7191–7196
Llama-3.2-3B	HellaSwag	RTX PRO 6000	sm_120	Site B	96	7186–7189
Llama-3.2-3B	HellaSwag	L4	sm_89	Site B	104	7181–7200
Llama-3.2-3B	MMLU	B200	sm_100	Site B	76	8731–8744
Llama-3.2-3B	MMLU	RTX PRO 6000	sm_120	Site B	112	8722–8728
Llama-3.2-3B	MMLU	L4	sm_89	Site B	116	8717–8747