Three levers that work, a budget argument that was wrong, and fifty that did not

ARC White-Box Estimation Challenge 2026, Phase 1 Participant: `amalgonim` · Best submission: 324409 (2.0925e-07)

> **Security disclosure — please read section 15.1 before the rest.**
>
> `GET https://www.aicrowd.com/api/v1/submissions/<id>`, authenticated with an ordinary participant API key, returns `score` and `score_secondary` for **any** submission id, not only the caller's. We used it, it changed what we did, and we are reporting it rather than keeping the advantage. It is placed at 12.1 because that is where the narrative reaches it; it is flagged here because this document is the only channel we have — the challenge exposes no contact address or discussion endpoint through its API, and the machine running this work has no mail transport. If any part of this write-up is read, it should be that one.

---

0. Summary

We submit an unbiased sampler with two exact closed-form corrections on top. It scores **2.0925e-7** on the grader, against **8.37e-6** for the bundled covariance-propagation baseline — a factor of 40, obtained without any learned component, approximation, or tuned constant. The ladders below were measured against an earlier submission of the same estimator at 2.877e-7; the ratios are paired and unaffected, and where an absolute score appears it is labelled.

Two levers account for all of it, and neither is clever:

lever	measured gain	why it is exact
cast weights to float32	**2.00x**	the dataset ships float64; the meter bills it at 2x
randomly-shifted rank-1 lattice + baker transform	**1.73x**	shift gives unbiasedness; the fold preserves the measure

Everything else we built is worth nothing. Measured at *equal cost* — 6,178 forward passes per MLP throughout, 24 MLPs x 6 shifts, paired within (MLP, shift):

estimator	MSE	vs plain i.i.d.
plain i.i.d. Gaussian, 2n draws	8.757e-6	1.000x
on the sphere of radius `E||x||`	8.663e-6	1.014x
+ antithetic pairing, n pairs	8.886e-6	**1.007x**
randomly shifted lattice + baker, 2n points	3.656e-6	**1.727x**
+ antithetic pairing, n pairs	4.864e-6	1.462x
+ input whitening	3.724e-6	1.717x
+ exact first-layer mean (deployed)	3.622e-6	1.755x

The deployed estimator is 1.755x over plain Monte Carlo and a bare lattice is 1.727x. Antithetic pairing — the device this note spends three sections on, and the one the estimator was designed around — is worth **1.007x** on its own, and *loses* 0.847x when a lattice is already present, which whitening and the exact first moment then claw back. Radius normalisation is 1.014x. We report this because it is what the measurement says; sections 3, 4 and 6 record what we believed while building them, and section 9 is about the error bar that let us believe it for a month.

The second half of this note is about a move we still think is the right one even though the ladder above prices it near zero: not estimating quantities that can be written down. Radius normalisation makes every draw `R u` with `u` uniform on the sphere, and `ReLU` of a linear form is homogeneous, so every Gaussian expectation of a homogeneous functional transfers to the sphere exactly (section 6). Three such quantities exist, all at the first layer:

closed form	**paired gain**	**deployed**
post-ReLU column means, `R ‖w‖ E|u_1| / 2`	1.055x	yes
input second moment `(R^2/d) I` (whitening)	1.052x	yes
the whole post-ReLU Gram, via the arc-cosine kernel	1.048x further	**no — see below**

Each is exact and each is worth a few percent, which is a fair description of what a first-layer identity can buy when 31 layers of sampling error sit on top of it. The third one's gain is three standard errors clear of zero and it still does not pay. Chasing *why* produced the one structural result here we would defend as new. The meter's untracked term is charged essentially **per array call**, not per FLOP: two of our own submissions differ by +38 calls and −4.9e7 FLOPs, and the grader charged the difference at +15.87 ms per MLP, about 0.4 ms a call. The whole free tier is worth on the order of a thousand array operations before a single FLOP is spent inside one. That is what kills the Gram correction — 23% more calls — and, counted the other way, it is a lever: the same estimator rewritten to make 128 calls instead of 168 is 1.6% cheaper in FLOPs as well, at bit-identical draws (section 8).

The rest of this note is mostly about what we *could not* make work, because that is where the information is. We measured eighteen candidate improvements to destruction, including five we expected to win, and we report every conclusion we got wrong the first time together with the bug or the statistic that caused it. There are more of those than we would like: two came from dividing noisy numbers, one from a mis-specified control, one from profiling a warm-up, and one — the expensive one — from a three-run local measurement that we built the whole estimator around and did not check against the grader for a month (section 1.1).

Four results we would put forward as more than a score.

The meter has a price list, and we publish it (section 10.13). For a `1/n` sampler the score is `k*c/B` with `k` the variance per row and `c` the *billed* FLOPs per row — the point count cancels, so the budget axis is flat and the cost axis is not. We priced twenty-six operations under `flopscope` on one core to find out what a row can be bought for. Every spelling of a dense contraction bills identically to the last digit; billing follows the output dtype and bottoms out at float32, so float16 costs exactly what float32 costs while running 110x slower; and the symmetry system prices a declared symmetric operand by orbit count rather than cell count, giving an exact `k!` discount at degree `k` — 2.00x at degree 2, 5.64x at degree 3 — **on pointwise and reduction operations only, and never on a contraction**. That last line is a standing invitation to moment-propagation methods and worth nothing to a sampler, and it is the clearest statement we can make of where this cost model wants the field to go.

The budget-placement argument in section 1 is wrong, and we falsified it ourselves. It says an unbiased sampler with `MSE ~ N^-alpha`, `alpha < 1` should buy the *smallest* budget the multiplier floor allows and stop. We measured `alpha = 0.89` locally, believed it, and shipped an estimator that used 10% of the budget. Four submissions at 10 / 44 / 69 / 94% utilisation give `alpha = 1.01 +- 0.03` on the grader — the score is flat in the budget, not decreasing — and the best of them is the one that spends 94%.

We derived a ceiling on everything in this note, withdrew it, and then found the withdrawal was the mistake (sections 8 and 14). Antithetic pairing annihilates every odd spherical-harmonic degree exactly and whitening annihilates degree 2, so the variance those devices can never reach is the even part of degree four and above. Measured first on the highest-variance neurons of three networks that came to 37.3%, an equal-cost bound of `1.34x` — which our own deployed estimator (1.755x) violates, so we withdrew it. The scored quantity is the mean over all 256 output neurons, and remeasured with that weighting on eight networks and 65,536 pairs at each of twenty-one inner products, degree 2 carries **18.1%** of the variance rather than 3.6% and the unreachable remainder is **26.1%**. The equal-cost bound is `1/(2 x 0.2611) = 1.915x` and the deployed estimator is at `1.731x` against the same baseline: **the ceiling is real and we are at 90.4% of it.**

And the remainder is unreachable for a reason that can be counted rather than argued (section 14.3). Every device here is a moment matcher; `N` points on `S^255` carry `N(d-1)` degrees of freedom and matching degree `I` imposes `dim(I)` conditions. Degree 2 needs 129 points and degree 3 needs 11,093, which is why the two devices that work are the two that work. Degree 4 needs **718,229** — `8.02x` our shipped `N` — and degree 6 needs `18,146x`. The point count is not budget-limited (`score = k c / B` is independent of it) but it is wall-clock-limited: we spend `8.0 s` of the allowed `60 s`, so `N` can grow `7.5x` and the necessary condition for degree 4 misses by a factor of `1.07`. That condition is also not sufficient — no spherical 5-design on `S^255` with `~7e5` points is known, and the cheap candidate, the `+-1/sqrt(d)` image of a strength-5 binary orthogonal array, is only a *spherical 3*-design because the hypercube has `E[x_1^4] = 1/d^2` against `3/(d(d+2))` on the sphere. And if all three obstacles vanished the prize would be **1.286x**, against an equal-cost gap to the field of `1.73x` to `2.61x`. Korobov moduli, CBC construction, scrambled Sobol', interlaced higher-order nets, the baker transform, stratification and per-network selection are not thirty-four independent disappointments; they fail for one reason and it fits on a line. Neither escape is open either: degree-4 matching inside an active subspace would need the integrand to live in a few directions and it needs 117 of 256 to hold 90% of the Jacobian energy, and buying cost instead of variance by carrying a rank-`r` factor needs rank 196 at layer 24 to hold `1 - 1e-6` of the centred energy, not the 16 the participation ratio suggests (section 14.4).

The submission endpoint is not access-controlled. `GET /api/v1/submissions/<id>` with our own token returns the score of *any* submission id, ours or not. We used it to read the field rather than guess at it, which is how we know the leaders sit 65x below us on raw error and that no sampler priced by this meter reaches where they are. It is reported to the organisers in section 15, with the observation that a competition whose scores are readable by every participant is a different game from one whose scores are not.

Reading scores was the first version of that; reading the *cost ledger* was the useful one. The public submission page carries per-network telemetry — billed FLOPs, effective compute, wall time, residual wall time, exhaustion flags — for every submission, and it does not require the key at all. Sections 10.12 and 10.13 are what it says, and it corrected us twice: we had inferred from the multiplier alone that the leading entry was a converged deterministic method running the full budget, and both halves of that are wrong. Its FLOP counts spread over 36 distinct values across networks, and it spends 9.5% of the

budget, not all of it. What is true is narrower and stranger: it is billed for 45 seconds of flopscope-backend time at 0.5% of a single core's measured throughput, and we could not find an operation in the library that sells a sampler's arithmetic at that price (section 10.13).

Our error is made evenly across all thirty-two layers, which caps almost everything we tried. Forcing the sample's mean to the truth at layer `l` and carrying on gives an oracle floor on every correction acting at that layer; the curve is smooth from `1.11x` at the input to `48.6x` at layer 30, and differencing it gives a per-layer share of two to four per cent everywhere, with no dominant layer (section 12.2). So a correction that acts at one site cannot buy more than about three per cent, which is not a disappointment about section 6's exact layer-1 closed forms — it is their ceiling, and section 6 measured it to within noise. Two more ceilings follow: the only surrogate whose exact mean we can write down is the one that keeps the ReLU at layer 1 alone, so the entire control-variate family is capped at layer 1's share, **1.11x** — the same number the layer-0 oracle returns, reached independently. And the sample's own second moment at the output would be worth **7.69x** if we knew its true value, with a generous tolerance of `3e-4`; we cannot get it, because a sampled anchor must be more accurate than the estimate it corrects and the flat budget axis forbids buying accuracy at a discount (worse at every target, best case `1.06x` worse), while the closed-form route is off by three orders of magnitude (section 12.5, 12.6).

The lattice is redundant with the whitening, and we found that out last. Section 5 measured a rank-1 lattice at about 1.4x over independent sampling and we built a generating-vector search on top of it. Running all four combinations — lattice, whitening, sphere projection, plain Monte Carlo — says the whitening alone is worth `2.00x +- 0.32`, and the lattice *given* the whitening is worth `0.97x +- 0.07`. Both devices do the same job, forcing the sample's low-order moments to be exact, and having done the cheap one there is nothing left for the expensive one to supply (section 12.4). It is why section 7.34's scrambled Sobol' net tied with Korobov, and why the budget exponent is 1.01: we have been running at the Monte Carlo rate the whole time with a lattice bolted to the front.

The deterministic route is closed too, and we can now say what the top of the board is not doing. One step of Gaussian moment closure, given the true moments one layer down, is a clean `width^-1.03` central-limit error (12 networks at four widths, log residual 0.056) — and at the width we are given that is still 6.7 times worse than our sampler. Chained, its per-layer errors add in quadrature to a one-per-cent method that stops improving with width entirely. The activation covariance does collapse with depth, from a participation ratio of 165 to 2.9, but the spectrum keeps a heavy tail and a paired projection oracle says a rank-8 carrier commits 87 times our error at layer 30 and 6100 times it at layer 8; the only truncation that beats us is rank 64 at layer 30, which is not a quadrature. So the leaders are not propagating moments, not integrating a low-rank subspace, not sampling (section 12.7), and not running a control variate. We suspected the arithmetic path itself — unmetered work billed at the residual rate — and for four entries in the top sixteen that is exactly right, but **not for the front**: the leading three are 71% to 94% instrumented, so whatever they run, the meter sees it (section 14.6). We also correct ourselves: our layer-0 closed form used the sphere's radius as the arc-cosine scale, which is exact for the mean and wrong by `1/width` for the second moment (section 13).

The information is joint, not marginal, and the front's ledger prices it per neuron pair. Three measurements close the last open routes and, between them, say what Phase 2 has to build. Substituting a matched Gaussian for the law of a **single** layer's pre-activation — exact mean, exact covariance, handed over by oracle — costs `2.9e-06` to `8.1e-06` at the output against a `2.65e-08` control, twelve to thirty-four times our whole error, and the cost is flat in depth, so there is no shallow prefix to

skip and the cheapest remaining cost lever is gone (section 14.7). Reconstructing the final marginal from `2n` exact moments as an `n`-atom Gauss rule converges as `n^-2` and would need about **340 moments** per neuron to reach the front — while two moments read as a smooth Gaussian density beat twelve moments read as atoms, which says the assumed shape, not the moment count, is what buys accuracy. And giving a layer its **exact** per-neuron marginals, 4,096 quantile knots each, tied by a Gaussian copula, removes only **12 to 18 per cent** of what the Gaussian substitution destroyed — at layer 1, none of it. Four fifths of the missing information is in the dependence, so the whole cheap family of per-neuron cumulant propagation is not merely weaker, it is looking in the wrong place (section 14.8). That is the same conclusion the front's cost ledger reaches from outside: after subtracting the `2.08e09` a covariance backbone alone would cost, the top three spend `2,021`, `10,957` and `22,046` billed FLOPs **per neuron pair per layer**, the size of a `20x20` to `66x66` two-dimensional quadrature, at 0.13 to 0.89 GFLOP/s — and since only *un-instrumented* time is charged, being slow costs them nothing (section 14.6).

Sections 14.9 through 14.14 follow that brief to its end and come back with a state variable. The problem is not 256 neurons but the 66 within two standard deviations of their kink, which carry 97.5% of the error while the saturated three fifths already sit at `1.1e-09`; all of the costly non-Gaussianity is created by the *first* rectifier and never grows (14.9). Moments do not determine the answer — a Markov–Krein program puts twelve exact moments no closer than `7.3e-06` — yet a four-moment two-normal mixture reads `3.77e-08` out of four of them, 193 times inside what twelve leave open, while maximum entropy on the identical four has **no solution for 99.2% of the neurons** because a quartic exponent cannot have heavy tails. Same information, different family, factor of twenty-five (14.10). That factor is unavailable to us and to anyone else who samples: measured across a budget ladder from one thousand to one million rows, `MSE(plug-in) = MSE(sample mean) + bias floor` to three figures, because homogeneity makes the plug-in the same linear statistic of the same rows — a shape prior has no variance to buy and can only spend bias (14.12). Finally, the pair map. Sixty-five per cent of the Gaussian closure's pair-block error is a lookup on `(c_i, c_j, rho)`, it saturates at twelve cells an axis, and **it is identical across networks** — the condition a Phase-2 entry must meet. The remaining third is one joint third moment, `E[u_i^2 u_j]` with the square on the *less saturated* neuron, worth `+0.559` correlation against a `+0.012` control where every marginal skewness and kurtosis is worth three to six points; and that object is a `W x W` slice of nearly **rank one** at depth, so carrying it costs less than the covariance already does (14.13, 14.14).

We then built that route and it does not work, and we report the number that kills it. The pair integral turns out to have a closed form nobody needs to quadrature: expanding `relu` in Hermite polynomials and applying Mehler's formula makes `E[relu(X)relu(Y)]` a sum of rank-one terms in elementwise powers of the correlation matrix, which reproduces our 512-node quadrature to five figures on all eight networks at **1/2,511 of the billed cost** — 30.6% of the budget down to 0.012% — so the whole deterministic chain costs 0.79% of `B` and cost stops being a constraint at all. Accuracy does not follow. Injecting the true pair correction at controlled amplitude (anchored at both ends: `f = 1` reproduces 14.9's independently measured `9.33e-07` to 2.4%) shows the map must be right to **99%** of its residual variance before the chain matches the sampler we already have — `f = 0.79` is worth `0.12` times our own score. A table actually baked on four networks and run on four others makes the pair block *worse* than doing nothing (0.68x), the one-argument mean table carries what little there is (1.87x against 18x needed), and held-out fits say `79%` is the ceiling of the variables `(c, rho, K)` rather than of the table, since eight co-skew bins are no better than four and finer `c` bins are worse. A Gaussian closure recovers 1.4% of the third moment at layer 30, so `K` would have to be carried — but a state that

delivers 79% delivers 0.12, so it does not matter. **Even flawless moment propagation with the four-moment read-out would score `3.77e-09` against the leader's `4.00e-10`.** The front is not doing this either. The one door we did not measure shut is that the last layer only ever asks for `E[relu(w_j . a_30)]` along 256 fixed directions — a one-dimensional marginal, not a joint law (14.16).

We also localise the error of the bundled analytic baseline to a single channel — its propagated *variance* is accurate, its propagated *mean* carries essentially all of the error, and the Gaussian assumption everyone treats as the wall is worth 1.2% of it (section 7.10), a diagnosis that section 7.13 confirms from the opposite direction; and we measure the public score's own standard error at **8%**, which is larger than most of the differences being acted on in this competition, including four of our own (section 9).

---

1. Where an unbiased estimator should sit on the budget axis

Score is `final_layer_mse * max(0.1, C/B)`. For a sampler with `MSE = sigma^2/N^alpha` and `C = c*N`, the score above the floor is proportional to `sigma^2 c N^(1-alpha)/B`. The exponent decides everything:

- `alpha < 1`: the score *increases* with N. Sit on the floor, `C = 0.1B`.
- `alpha > 1`: the score decreases without bound. Spend the entire budget.
- `alpha = 1` exactly: the score is flat, and only `sigma^2 c` matters — variance times cost per sample.

So the first question to settle is not which estimator to use but which side of `alpha = 1` this problem lies on. We measured it rather than assuming it, running the same estimator at three budgets on the 100-MLP local split:

C/B	raw MSE	adjusted score
0.100	3.33e-6	**3.33e-7**
0.305	1.28e-6	3.89e-7
0.703	5.91e-7	4.17e-7

The local slopes are `N^-0.86` and `N^-0.92`, and `N^-0.89` overall — below the canonical rate, not above it. **The prediction that a lattice rule would beat `N^-1` and thereby reward a larger budget was wrong.** The randomised lattice does buy a constant factor, but its asymptotic rate on this integrand is no better than i.i.d., so the score rises monotonically with the budget and the floor is optimal.

A fourth run at `C/B = 0.94` failed 89 of the 100 MLPs on `budget_exhausted`: the check is per-MLP and unforgiving, and any budget-shaped estimator needs a margin wide enough to absorb the variation in the harness's own accounting. Sitting at the floor also leaves the wall-clock margin at its widest — we use 0.9 s of the per-MLP cap, single-threaded.

1.1 That conclusion is wrong, and the grader says so directly

The table above is a local measurement of three runs, and `alpha` is a slope between two of its numbers. The score's own standard error is 8% (section 9), which puts the difference between `alpha = 0.89` and `alpha = 1.05` inside the noise of the very measurement used to rule the second one out. We built the entire estimator on the wrong side of that line and stayed there for the whole competition.

What settles it is a field of the submission API we had not read. `GET /api/v1/submissions/<id>` returns `score_secondary` alongside `score`, and the ratio of the two comes back as **exactly the multiplier the grader applied** — 0.1000 for a submission on the floor, and matching the predicted utilisation to three digits for two submissions deliberately placed above it. So `score_secondary` is the raw final-layer MSE, and every submission is a noise-free measurement of `MSE(N)` on the grader's own hundred networks rather than on a local replica.

Four submissions of the *same estimator* at four point counts:

`npair`	**multiplier**	**raw MSE**	**adjusted score**	`alpha` **from the row above**
3,089	0.1000	3.0877e-6	3.0877e-7	
14,207	0.4432	7.7243e-7	3.4225e-7	0.908
22,111	0.6888	4.2428e-7	2.9225e-7	1.355
30,323	0.9442	3.0474e-7	**2.8772e-7**	1.047

`alpha = 1.014` **across the whole range**, against the 0.89 section 1 measured and acted on. Segment to segment it reads 0.91, 1.36, 1.05, and we do not believe the 1.36: a raw MSE averaged over a hundred networks carries about 5% of standard error, which over a 1.37x step in `N` is 0.24 of exponent. The honest statement is `alpha = 1.01 +- 0.03`, which is the one value the whole design hinged on and the one we could not distinguish from 0.89 with three local runs.

At `alpha = 1` the score is flat in the budget — every point count is equally good and the placement argument evaporates. The two large-budget submissions came back 5% and 7% better than the floor, which is inside the 8% score noise and which we therefore do *not* claim as a gain. What we claim is narrower and more useful: the floor was never optimal, it was merely not costly, and a week of work went into an estimator shaped around a constraint that was not there.

The same table prices the cliff that scared us off in the first place. The multiplier came back at 0.6888 where the tracked FLOPs alone are 0.6880 of the budget, so `lambda * R` contributed 0.0008 — **about 2 ms of residual wall per MLP** at 22,111 points. Our local machine reported 200 ms for the same code, all of it contention from unrelated jobs. The `C/B = 0.94` disaster was a mis-specified FLOP model, not a wall-clock margin that was too thin, and with the point count set from the *measured* per-pair cost (8.4752e6 FLOPs, from est_v11's 2.61794e10 at 3,089 pairs) the combined-budget check is roughly 150 ms away rather than 2 ms away.

We leave section 1 standing as written above, because the argument in it is correct and only its input was wrong, and because the failure mode is worth more than the conclusion: a quantity whose sign decides the whole design was measured once, to a precision that could not resolve the sign, and then treated as settled for a week.

1.2 And `alpha = 1.01` is also not the end of it

The four submissions above span 3,089 to 30,323 points on the `est_v11` family. The estimator we ended up shipping runs at 40,361, and at that end of the range the exponent is not 1.01. Two submissions of `v16` differing in one constant — 40,361 points against 44,753 — return raw MSEs of 2.7066e-07 and 2.3817e-07, an exponent of **1.258**; the paired local measurement on the same hundred networks gives 1.243 without touching the grader. Both are far enough above 1 to act on, and acting on it is worth 3.0% (section 11.4).

So the axis is not flat, it is mildly favourable, and it is favourable in the direction we spent the first week avoiding. The reason the two measurements disagree is not that either is wrong: `alpha` is a local slope and there is no reason for a lattice rule to have one exponent across a 13x range of `N`. The mistake was treating a slope fitted over 3,089-30,323 points as a property of the problem rather than of that interval.

2. The float64 trap

`flopscope` 0.10.0 (released 3 Aug 2026, one day before this work) made the meter dtype-aware: float64 now costs twice float32. The dataset hands weights to `predict()` as **float64**. An estimator that does its arithmetic in the dtype it was given therefore pays double for every matmul, which is 99.6% of a sampler's bill.

```
weights = [w.astype(fnp.float32) for w in mlp.weights]   # 0.015% of the tier
```

This is the single largest lever we found, and it is one line. We flag it because the cost-model change is recent enough that entries built against the older meter may still be paying it.

We checked whether the ladder continues: float16 is billed at the **same** rate as float32 (measured, rate 1.00), so there is no further discount and no reason to take the precision risk.

3. Antithetic pairing is exact at layer 1

With no biases, `ReLU(z) + ReLU(-z) = |z|`, and for `z ~ N(0, sigma^2)`, `E|z|/2 = sigma/sqrt(2*pi) = E[ReLU(z)]`. So the antithetic *pair average* at layer 1 has zero variance — first-layer Rao-Blackwellisation falls out of the pairing for free, with no special-casing. Deeper layers are not antisymmetric, but the pairing still removes the whole odd component of the integrand at every depth. Measured: 3.65x over i.i.d. at equal cost.

That last sentence is false and it is the load-bearing claim of this section. The 3.65x was never measured against a control at matched forward passes; a proper equal-cost ladder puts the pairing at **1.007x** alone and 0.847x on top of a lattice (section 0). The algebra above is correct — layer 1 really is exact under pairing — and it buys nothing measurable at layer 32, which is the only layer scored. Section 9 records how the claim survived a month. The rest of this section is left as written.

One consequence worth stating: because pairing already annihilates every odd function, **any linear control variate is identically zero on the paired estimator**. Candidate control variates must be even.

4. Radius normalisation from positive homogeneity

A bias-free ReLU network is positively homogeneous, `f(cx) = c f(x)`. For `x ~ N(0, I_d)` the radius `r = ||x||` is independent of the direction `u = x/r`, so

```
E[f(x)] = E[r] * E[f(u)],    E[r] = sqrt(2) * Gamma((d+1)/2) / Gamma(d/2)
```

Rescaling every draw onto the sphere of radius `E[r]` removes the radial variance exactly. `E[r]` comes from the Gamma-ratio asymptotic series, `sqrt(d)(1 - 1/(4d) + 1/(32d^2) + 5/(128d^3) - 21/(2048d^4))`, which we verified against `lgamma` to a relative error of 6e-14 at d = 256 — far below any level that could bias the result.

The predicted gain is small and we can state it in closed form: the variance ratio is `1 + m^2/(2 d v)` where `m, v` are the mean and variance of `f(u)`. With the measured pair relative s.d. of ~0.20 this predicts **1.05x**. Measured: **1.044x**. We keep it because it costs 0.02% and the derivation is exact, but we note honestly that **once a lattice is used the gain disappears** (1.294x with and without) — the lattice already equalises the radii, so the two levers overlap almost completely.

5. Point sets: a modest, flat gain

Unbiasedness of a randomly-shifted lattice comes from the Cranley-Patterson shift alone. Every point is marginally uniform whatever the generating vector is, so **the generating vector may be chosen by arbitrary offline search without any risk to the estimator's mean**. We used that licence twice.

Korobov sweep. Over `a in {17, 101, 1021, 1571, 2311, 3001}` at N = 3229, the geometric-mean speedups were 1.51 / 0.91 / 1.52 / 1.24 / 1.35 / 1.26. We took a = 17, not because it was nominally first but because it had the shallowest downside — it never lost to i.i.d. on any MLP, whereas a = 101 lost on all four.

CBC construction. We then built the generating vector properly, by component-by-component minimisation of the shift-averaged worst-case error in the weighted unanchored Sobolev space, for three weight settings. On eight MLPs with 40 randomisations each, split into selection and held-out halves:

vector	all	selection	held out
Korobov a=17	1.31	1.28	1.35
CBC gamma=0.02	1.32	1.23	1.43
CBC gamma=0.05	1.39	1.38	1.39
CBC gamma=0.15	1.32	1.24	1.41

CBC does not beat the one-parameter guess. All four sit inside the ~8% noise of the comparison. We then tried to settle it in the grader rather than the harness: a submission identical to ours except for the CBC vector scored **3.85e-7** against our **3.33e-7** on the full 100-MLP split, and we recorded that as "the vector buys nothing". That was an overreading, and section 9 retracts it — a 16% gap on a single 100-MLP run is inside the score's own error bar, so the grader did not settle anything the offline harness had not already said. The conclusion survives, and we later re-established it properly. Once the paired harness existed we re-ran all four vectors inside the *deployed* pipeline — tent transform, sphere normalisation, whitening, mean correction — on 24 MLPs x 6 independent shift draws:

vector	paired gain vs Korobov a=17	wins
CBC gamma=0.02	0.981x (+-5.3%)	73/144
CBC gamma=0.05	1.024x (+-6.0%)	74/144
CBC gamma=0.15	0.990x (+-5.7%)	66/144

Every one is a coin flip: win rates of 51%, 51%, 46%, and no gain distinguishable from zero. This is the same conclusion we reached at first, reached for the right reason — the original evidence (a single grader delta) could not have supported it. A generating vector is the cheapest improvement available in this problem, a precomputed constant with no runtime cost at all, so it was worth the second look; there is simply nothing there.

That conclusion later got stronger than we wanted. Every figure in this section compares point sets against an i.i.d. baseline that has *not* been whitened. Section 12.4 runs the comparison the other way — with the sphere projection and the whitening applied to both arms, so only the point set differs — and the lattice's marginal contribution is `0.97x +- 0.07`. The 1.4x is real and it is also entirely reproduced by the Cholesky solve that the deployed pipeline was already doing for other reasons. Read this section as the record of a search that was well conducted and turned out to be for something we already had.

Our reading is that the CBC criterion assumes coordinate importance that decays, whereas here the input coordinates are exactly exchangeable (isotropic Gaussian, He-initialised first layer), and the ReLU kinks put the integrand outside the smoothness class the criterion optimises for. The practical conclusion is the useful one: **any decent rank-1 lattice buys about 1.3x here, and effort spent choosing among them is wasted.**

Latin hypercube gave 1.14x. A randomly-shifted Kronecker sequence with `alpha_j = sqrt(p_j)` was actively harmful, 0.3x.

The fold, however, is not. Applying the baker transform `u -> 1 - |2u - 1|` after the shift and before the inverse-CDF map is worth **1.17x** on the offline harness and **1.29x** end-to-end, and it wins on 7 of 8 MLPs:

	all 8	**first 4**	**last 4**
Korobov a=17	1.39	1.55	1.24
Korobov a=17 + tent	**1.63**	1.80	1.47
CBC gamma=0.02	1.38	1.46	1.31
CBC gamma=0.02 + tent	1.73	1.87	1.59

None of this is our invention: tent-transformed lattice rules are a standard construction (Hickernell, *Obtaining O(N^-2+eps) convergence for lattice quadrature rules*, MCQMC 2002), and the reason we tried it is that its usual justification — periodising a non-periodic integrand — is exactly the defect a shifted lattice suffers from here. What we contribute is the measurement that it survives contact with a 32-layer ReLU network, whose kinks put the integrand well outside the smoothness class the `O(N^-2)` theory assumes.

The fold is measure-preserving on `[0,1]` — it is 2-to-1 and piecewise linear with slope +-2 — so the transformed points remain marginally uniform and the estimator remains unbiased; the empirical MSE against ground truth confirms this, since any bias would show up there directly. One detail matters: the fold satisfies `tent(u) = tent(1-u)`, so it would collapse a lattice's point `n` onto its point `N-n` if the shift were zero. With a random shift they do not coincide, but the fold must be applied *after* the shift, not before. It costs four operations per element, 0.012% of the budget. We take Korobov + tent rather than the nominally better CBC + tent, because the CBC vector is constructed for one specific point count and we prefer an estimator that degrades gracefully if the affordable point count changes.

Note that the tent and the CBC criterion are two attacks on the same weakness (the integrand's non-periodicity), and the cheap one wins: the fold buys 1.17x where the principled construction bought nothing.

---

6. Closed forms the sample can be forced onto

Everything up to here is unbiased, and by the estimates other participants have filed it is close to the floor for that class: an unbiased sampler at this budget has a per-pair variance around 0.012 ([t/18105]), and ours measures 0.0109. If there is anything left, it is not in drawing better points. It is in refusing to estimate quantities we can write down.

Three such quantities turned up, all consequences of one fact: after radius normalisation every draw is `R u` with `u` uniform on the sphere, and `ReLU` of a linear form is homogeneous of degree one. Writing a Gaussian `x` as `||x|| u` with the two factors independent, any degree-`k` homogeneous functional `F` obeys

```
E_gauss[F(x)]  = E[||x||^k] E_u[F(u)]
E_sphere[F(x)] = R^k        E_u[F(u)]
```

so `E_sphere[F] = (R^k / E||x||^k) E_gauss[F]`, and the angular integral — the part we cannot do — cancels. Every Gaussian expectation of a homogeneous functional therefore transfers to the sphere for the price of one scalar.

These corrections are **not** unbiased. Each replaces a sample statistic with an exact value, and the downstream nonlinearity does not commute with that substitution, so the estimator picks up an `O(1/N)` bias. Every number below is measured MSE against the 1e9-sample ground truth, paired per MLP across independent shift draws, so the bias is inside the measurement rather than argued away. None was adopted on a leaderboard delta; at a standard error near 8% (section 9) the grader cannot see effects this size.

6.1 The first-layer mean

`k = 1`. For a first-layer column `w`, `z = R ||w|| (u . w_hat)` and `u . w_hat` is distributed as one coordinate of a uniform unit vector, so

```
E[ReLU(z)] = R ||w|| E|u_1| / 2,
E|u_1| = 2 Gamma(d/2) / ( sqrt(pi) (d-1) Gamma((d-1)/2) )   = 0.04991651 at d = 256
```

Antithetic pairing already forces the sample mean of `z` to zero *exactly*, but it does not force the sample mean of `ReLU(z)` to this value: the pair statistic is `|z|/2` and its mean still fluctuates. Replacing it with the known one, as a broadcast shift of the first activation, is worth **1.055x (+-1.5%)** over 40 MLPs x 6 shifts, and costs one column mean and one broadcast add — 0.013% of the free tier. The grading image has no scipy, so the Gamma ratio goes through `math.lgamma`.

6.2 Whitening the input point set

The sample second moment of the draws is not `(R^2/d) I`; on our lattice its diagonal is right to 0.2% but the off-diagonal block carries an rms of 3.7e-3. Forcing it costs a Cholesky of `S = x^T x / n` and one

triangular solve, and the whitener folds into `W_1` so it never touches the `n`-row array.

Worth **1.052x (+-1.5%)** for 1.7% of the budget, net **1.036x**. It stacks with the mean correction almost perfectly — the two act on different objects, one on the second moment of `x` and one on the first moment of `ReLU(x W_1)` — giving **1.092x (+-2.0%)** paired, **1.076x** after cost. That combination is what we submitted as our second entry; it scored 3.22e-7 against 3.33e-7, a 7% gain in raw MSE, consistent with the offline prediction and (as section 9 argues) not by itself distinguishable from noise on the grader.

A diagnostic that mattered: the whole gain is **off-diagonal**. Forcing only the diagonal of `S`, or forcing it before rather than after the tent transform, is worth 1.000x to three digits in every arrangement we tried. We had read "max deviation 5.4%" as diagonal dominance and were wrong; the off-diagonal energy exceeds the diagonal deviation by about 3000x.

6.3 The whole first-layer second moment

`k = 2`, and this is the one that pays. The full matrix

```
G_ij = E[ ReLU(z_i) ReLU(z_j) ],   z = x W_1
```

is available in closed form, and the diagonal we matched in 6.1 is 256 of its 65536 entries. Applying the homogeneity identity with `E[||x||^2] = d`,

```
E_sphere[ReLU(z_i) ReLU(z_j)] = (R^2 / d) ||w_i|| ||w_j|| k(rho_ij),
    k(rho) = ( sqrt(1 - rho^2) + rho (pi/2 + arcsin rho) ) / (2 pi),
    rho_ij = w_hat_i . w_hat_j
```

`k` is the order-1 arc-cosine kernel. Nothing here is a Gaussian approximation: the pre-activations are exact linear images of a spherical vector, and both steps are exact. Two internal checks come for free — `k(1) = 1/2` reproduces `E[ReLU^2] = ||w||^2 R^2 / (2d)`, the diagonal, and `k(0) = 1/(2 pi)` reproduces the independent case — and we checked the whole matrix against 1.28e8 Monte Carlo draws at `d = 32`, agreeing to 5e-4 and falling as `1/sqrt(n)`.

The correction centres the sampled activations, maps their sample covariance onto the exact one, and re-adds the exact mean:

```
A <- (A - mean A) M + m_exact,     M = L_hat^-T L_exact^T,   L L^T = cov
```

`M` is a constant `256x256`, so it folds into `W_2` and the per-sample work is a broadcast subtract and a broadcast add. The first reported row becomes the exact first-layer mean rather than a sampled one.

The antithetic structure halves what the covariance costs. With `A = [ReLU(z); ReLU(-z)]` and `ReLU(+-z) = (|z| +- z)/2`,

```
A^T A = ( |z|^T |z| + z^T z ) / 2
```

and after 6.2 the second term is exactly `n (R^2/d) W_1^T W_1`, a single `256^3` matmul rather than an `n`-row one. So the whole thing costs one `n`-row product (1.53% of the free tier) plus 0.62% of fixed `width^3`

work: 2.1% of the sample count, and so 2.1% of MSE at the `alpha = 1.01` that section 1 eventually measured on the grader. We wrote 1.9% here originally, converting at the local `N^-0.89`; the difference is immaterial to the decision and the stale exponent is not, so it is corrected rather than left.

variant	paired gain vs uncorrected	wins
first-layer mean (6.1)	1.058x (+-2.4%)	56/96
whiten + mean (6.2, submitted)	1.103x (+-3.1%)	59/96
covariance match, sampled mean	1.096x (+-3.7%)	58/96
covariance match, exact mean	1.152x (+-4.4%)	56/96
whiten + covariance match	1.169x (+-4.6%)	56/96

24 MLPs x 4 shift draws, each against the uncorrected sampler. Read naively that table says the covariance match beats the deployed `whiten + mean` by 1.169/1.103 = 1.060x, and we believed that for about an hour. It is not a measurement. Both entries are noisy estimates against a *third* quantity, and dividing them compounds two error bars instead of cancelling them — the same mistake, in a subtler dress, that section 9 is about.

Measured properly — 60 MLPs x 6 shifts, the two estimators paired against **each other** on every run:

`whiten + mean` set-mean MSE	3.2494e-06
`whiten + cov` set-mean MSE	3.0279e-06
geometric paired gain	**1.048x (+-1.5%)**
arithmetic set-mean ratio	**1.073x**, 95% CI [1.016, 1.140]
wins	190/360

So the effect is real — 1.048x is three standard errors clear of nothing — and about a third the size of the naive estimate. The arithmetic ratio is the larger one, which says the correction helps most on the hard networks in the upper tail, the same tail that gives the score its own noise.

Whether it is worth **deploying** is a separate question, and here the answer turned out to be no. Break-even on tracked FLOPs alone is 1.019x, which 1.048x clears. But the Cholesky factorisations and triangular solves are billed as *residual wall time* rather than as FLOPs, and at the measured `gamma = 1.06e11 FLOP/s` that is the expensive way to spend a second. Submitting it moved compute utilisation from 0.10391 to 0.10956, and the entire increase was the wall term:

	tracked	residual wall	utilisation
`whiten + mean`	0.0975	0.0064	0.10391
`whiten + cov`	0.0970	0.0126	0.10956

A 5.4% utilisation penalty against a 4.8% gain, and the submission came back **3.72e-7** against 3.22e-7.

We first wrote that up as a lever defeated by how the meter prices five LAPACK calls. The arithmetic does not support that reading, and we correct it here. Dividing each score by its own multiplier recovers the MSE the server actually measured: 3.0988e-06 for `whiten + mean` against 3.3954e-06 for `whiten + cov`. The meter cost 1.054x, as advertised — but the MSE came back **1.096x worse**, when the paired

local measurement said it should be 1.048x *better*. The meter explains a third of the 1.155x loss. The rest is the score's own error bar, and we should have recognised it: section 9 puts a single submission's noise at a scale that swallows a 5% effect whole, our 1.048x was measured on 60 MLPs of the public split, and the server grades 50 different ones.

So the honest statement is narrower than the one we made. The correction is exact and the local gain is real at three sigma; it costs 5.4% of utilisation, which is more than half of what it buys; and one submission cannot resolve the remainder either way. We left it out because a lever that needs more than its own size in budget is not worth carrying, not because we watched it lose.

6.4 What stopped working once the covariance was matched

Matching the first-layer *second moment* per unit — the diagonal of `G`, which we had deployed as a separate multiplicative rescale — is worth 1.021x on its own and **exactly nothing** once the input is whitened. That is the expected behaviour and a useful check that the two are hitting the same structure: after whitening, `z^T z / n` is already exactly `(R^2/d) W_1^T W_1`, whose diagonal is `E[z^2]`, and rectification halves it exactly. There was nothing left to fix.

We looked for the same closed form one layer deeper and there is none. Layer 2's pre-activation is `a_1 W_2` with `a_1` neither spherical nor Gaussian, so the homogeneity identity has nothing to act on. A Gaussian surrogate is available — `a_1` is a sum of 256 terms — but that is an approximation rather than an identity, and section 7.10 measures what that class of assumption is worth here.

---

7. Thirty-four things that did not work

7.1 Control variates from a collapsed linear surrogate

For **any** fixed matrix `R`, `z = R^T x` is exactly Gaussian, so `E[ReLU(z_i)] = ||r_i||/sqrt(2*pi)` in closed form. This gives a free unbiased control variate for any `R` whatsoever — an unusually generous setup. We built `R` by collapsing the network's ReLUs into fixed diagonal gates, `R = W_1 G_1 W_2 ... G_{L-1} W_L`, with three choices of gate.

Result: **correlation 0.08, variance reduction 0.7%**. The majority-vote gate (`1[P(on) > 0.5]`) collapsed the 32-fold product to zero; the mean-calibrated gain gate diverged to 1e24. Only the mean-field gate `diag(P(on))` produced a usable matrix, and it barely correlates with the network it was derived from. After 32 layers the realised activation pattern retains essentially no mutual information with any fixed reference pattern.

7.2 Rotating the lattice onto the network's own directions

The integrand reaches `x` only through `W_1`, so its structure lives in a rotated frame while a lattice's good projections live in its leading coordinates. Since `x ~ N(0, I)` is rotation-invariant, `x = V y` is unbiased for *any* orthogonal `V`, so we are free to align them. We took `V` from the singular basis of the mean-field Jacobian.

The premise checked out, strikingly. **The mean-field Jacobian of a 32-layer width-256 He-initialised ReLU network has a participation ratio of 4.5-6.0 out of 256**, with `s_0/s_255 ~ 3e16` — its linear response is effectively rank 5. We had expected low effective dimension; we did not expect *that* low.

The exploit still failed: 1.45x plain lattice became 1.24x (right singular basis) or 1.39x (left). Two reasons, and we think both matter. The mean Jacobian identifies where the *signal* is, not where the *estimator's variance* is; and with a spread of 1e16, every singular vector past the first handful is numerical noise, so `V` is close to an arbitrary rotation — which destroys the lattice's axis-aligned quality without buying anything.

The second half of that explanation is wrong. Section 7.15 tests it with the control this section is missing and finds no alignment effect of any kind, so "destroys the lattice's axis-aligned quality" is a phrase we wrote because it sounded like a mechanism, not because we had measured one. The first half — that the mean Jacobian points at the signal rather than at the variance — survives. We have left the original sentence above rather than quietly editing it, because the pattern it belongs to is the subject of section 9.

7.3 Dead-neuron pruning

A neuron that is off for every input contributes exactly nothing downstream, so deleting its row and column is exact rather than approximate. Measuring against a 65,536-sample reference, 8-13% of neurons are genuinely dead, and the pruned forward pass would cost 0.78-0.86x.

But a *certificate* is needed, and the affordable one is moment propagation. At the threshold where it kills no live neuron (`alpha > -6`), the cost ratio is only **0.88-0.95** — about 8%. Tightening to `alpha > -3` reaches 0.70x but falsely kills ~450 live neurons per network. We judged 8% not worth a bias channel and dropped it.

The mirror-image idea is worth more than the 8%, and it fails for a reason that turns out to be structural. A neuron that is *on* for essentially every input is the identity, and consecutive always-on units compose — so the entire always-on part of the network collapses into a single 256x256 matrix computed once per MLP rather than once per sample. With `K` genuinely straddling units the per-sample cost falls from `32 * 2NW^2` to about `2NW^2 + 4NWK`, which at `K = 1600` is 2.5x cheaper. At `alpha ~ 1` a sample 2.5x cheaper is worth 2.5x in MSE, and that is larger than every variance lever in this note combined.

There are no always-on units. Across 3 MLPs and 20,000 sphere samples, mean `P(on)` is **0.50 at every one of the 32 layers** — layer means run 0.458 to 0.525 with no trend in depth, against 0.500 exactly at layer 1 where it is forced. It does not decay into structure; it stays put. In hindsight it could not have been otherwise: a bias-free He-initialised network has `z = a W` with `W` zero-mean and symmetric, so every pre-activation is symmetric about zero at every depth, and half the units fire by construction. The network never saturates because nothing in it can break the symmetry.

The cost model reads accordingly — the tolerance has to be *loosened* to buy anything, and even then it does not:

tolerance on `P(on)`	**straddling units**	**speedup vs full**
1e-2	4,427 / 8,192	0.90x
1e-3	5,231 / 8,192	0.76x
1e-4	5,758 / 8,192	0.70x

Note the orientation flips between the two halves of this section: the pruning figures above are cost ratios, where 0.78x is a saving, and this column is a speedup, where 0.90x is a loss. Below 1.0x means *more* expensive: the collapsed-linear rewrite costs more than the network it replaces as soon as most

units straddle, and most units always do. This is the same wall as 7.6 seen from the cost side, and it is the reason we stopped looking for cheap forward passes.

7.4 A larger compute budget

This entry is the one that was wrong, and it is the most expensive mistake in this note. It is left in place, with its original reasoning, because section 1.1 is about how it survived.

Covered in section 1. The hypothesis that QMC's faster-than-`N^-1` convergence would make the score decrease in `N` was **false** for this integrand: the measured rate is `N^-0.89`, so the score rises with the budget.

That measurement was three local runs. Four grader submissions spanning 10% to 94% of the budget give `alpha = 1.01 +- 0.03`, and the 94% submission is our best score. A larger compute budget was not a thing that did not work; it was a thing we never tried until we could no longer avoid noticing that we had not.

This one is worth a warning, because our first measurement said the opposite and was wrong. The lattice points are generated as `(n * g) mod N`, and `n*g` reaches `N^2`; in float32 the 24-bit mantissa silently rounds that product above `N ~ 4096`, so every large-budget run was integrating over a corrupted point set. The apparent "error floor" we first reported was our own rounding. Generating the lattice in float64 and casting the result back to float32 costs 2x on one cheap `(N, width)` array and fixes it; the corrected sweep is the table in section 1, and it happens to reach the same conclusion for an entirely different reason.

A second trap in the same construction: with `g_j = a^j mod N`, the vector degenerates whenever the multiplicative order of `a` is at most `width`. At `N = 397` the order of 17 is 132 and at `N = 6337` it is 288, so both wrap inside the 256 components; a sweep across N that does not check this reads as violent non-monotonicity in the error (we measured a 5x spike at `N = 6337`) and is easy to mistake for real structure.

7.5 float16

Same billing rate as float32. No discount, so no reason to accept the precision loss.

7.6 A low-rank multilevel estimator

This was our best remaining idea and the one we most expected to work, so we give it in full.

The cost breakdown of a pure sampler is **99.6% matmul**, 0.2% ReLU, 0.2% reduction. Every variance lever we found saturates at 1.3-1.8x, so the only remaining headroom is on the cost side, and the only unbiased way to cut matmul cost is to make most samples cheaper and correct the difference. The natural construction is multilevel: a cheap level `f0` carrying only a rank-`r` projection of the activation fluctuation, coupled to the exact level through shared inputs, with total cost `(sqrt(V0 c0) + sqrt(V1 c1))^2` against `V c` for single-level.

Both factors have to cooperate, and neither does.

The spectrum is too flat. Fraction of activation-fluctuation energy captured by rank `r`:

layer	r=4	r=8	r=16	r=32	r=64
1	0.051	0.098	0.184	0.328	0.544
8	0.319	0.477	0.653	0.814	0.920
32	0.781	0.873	0.939	0.975	0.993

Layer 1 is essentially white — of course it is: it is a fixed linear map of an isotropic Gaussian through a pointwise nonlinearity, so there is no preferred subspace for it to have. The spectrum only becomes usable near the output, which is the wrong end: the cheap level has to be cheap *everywhere*.

The coupling is too loose. `Var[f − f0] / Var[f]`, which is what the correction term's variance actually depends on: 0.98 at r = 8, 1.00 at r = 32, 0.84 at r = 64, 0.67 at r = 128. And a rank-`r` surrogate costs `4*256*r` per layer against the exact `2*256*256`, so **r = 128 costs exactly as much as the network it approximates**. At the only ratio that is even nominally attractive, r = 64, the gain formula gives `(sqrt(0.5) + sqrt(0.84))^2 = 2.6` — a factor 2.6 *worse* than doing nothing.

We report this as a clean negative: for deep random ReLU networks, activation fluctuations do not admit a cheap low-rank surrogate, and multilevel Monte Carlo has nothing to stand on here.

7.7 Kronecker sequences and Latin hypercube

0.3x and 1.14x respectively (section 5).

7.8 Scrambled Sobol' sequences

A rank-1 lattice is a single generating vector stretched over 256 dimensions, which is a lot to ask; digital nets have good low-dimensional projections by construction and Owen scrambling makes them unbiased. If the lattice were the binding constraint, this is where it would show.

Averaged over MLPs, speedup over i.i.d. at `N = 2048 / 8192 / 32768`:

point set	2048	8192	32768	overall rate
tent lattice	2.01x	1.53x	1.70x	`N^−1.00`
scrambled Sobol'	2.18x	1.26x	2.30x	`N^−1.08`
Sobol' + tent	1.17x	1.06x	1.83x	`N^−1.22`

Sobol' is ahead at the ends and behind in the middle; the spread between the three columns is larger than the spread between the rows, so this is noise around a common value. Neither point set achieves a rate better than `N^−1`, and at the production size the difference is not worth the bit-twiddling a scrambled net would cost inside the meter. Stacking the tent transform on top of Sobol' *hurts* — a digital net is already effectively periodised, and the fold only halves the resolution.

Checked again at the size we actually deploy. Every sentence above was measured at `N <= 32768` because the lab runs on CPU, while the submission runs at `N ~ 6.2e4` — and since the lattice's advantage was *shrinking* with `N` and Sobol' held a slightly better rate, the ordering had no right to be assumed. On the GPU it is cheap to settle: 40 MLPs x 5 shifts, paired, every estimator at the same number of forward passes.

point set	N = 32768 vs i.i.d.	vs deployed	N = 65536 vs i.i.d.	vs deployed
i.i.d. gaussian	1.000x	0.547x	1.000x	0.544x
tent lattice	1.641x	0.898x (+-5.9%)	1.869x	1.016x (+-5.4%)
tent lattice + antithetic	1.738x	0.951x (+-2.2%)	1.614x	0.878x (+-2.7%)
scrambled Sobol'	1.855x	1.015x (+-5.7%)	1.935x	1.053x (+-5.7%)
Sobol' + tent	1.607x	0.880x (+-5.5%)	1.764x	0.959x (+-6.1%)
Sobol' + antithetic	1.914x	1.047x (+-5.9%)	2.017x	1.097x (+-5.7%)
deployed stack	**1.827x**	1.000x	**1.839x**	1.000x

Nothing beats the deployed stack by more than two standard errors at either size. Sobol' with antithetic pairing is the only candidate that leads on the point estimate — 1.047x and 1.097x — and neither reaches significance; folding in the multiplier it would pay for generating a scrambled net, the implied score at `N = 32768` is 3.565e-07 against the deployed 3.417e-07, so it loses on the quantity that is actually scored. The small-`N` reading was right after all, which we did not expect and would not have believed without measuring: a single MLP at `N = 65536` showed Sobol' + tent at 4.53x and it evaporated at 40.

One number in the table is a warning rather than a result. A Korobov lattice needs a **prime** modulus, and `N - 1` is composite for every power of two above four — `32767 = 7 x 31 x 151`. Running the same comparison with that modulus put the lattice at 0.149x, six times *worse* than i.i.d., and it looks exactly like a real collapse of lattice quality at large `N` rather than like a bug.

7.9 Orthogonal Monte Carlo

Each row of a Haar-random orthogonal matrix is marginally uniform on the sphere, so averaging over the rows is unbiased, and the rows are exactly mutually orthogonal — a much stronger spreading property than a rank-1 lattice can express when `N << 2^d`. Combined with the radius normalisation of section 4 this is an exactly unbiased estimator.

Speedups over i.i.d. at the production size: tent lattice **2.22x**, Haar blocks **1.56x**, one Haar rotation applied to the whole tent set **1.48x**, both **1.14x**. Orthogonality is the wrong invariant here: it controls pairwise angles, and 256 mutually orthogonal directions still leave the sphere in 256 dimensions almost entirely unvisited, whereas the lattice controls low-order projections, which is what a function reaching `x` only through `W_1 x` actually sees.

7.10 Making the analytic baseline exact

The bundled covariance-propagation estimator is exact everywhere except one step, the post-ReLU off-diagonal covariance, which it approximates by a product of gates:

```
cov_post[i,j] ~= Phi(alpha_i) Phi(alpha_j) cov_pre[i,j]
```

If the pre-activations are jointly Gaussian, that second moment has an exact value. Writing `z_i = mu_i + s_i zeta_i` and conditioning on `zeta_i` reduces `E[(zeta_i + a_i)^+ (zeta_j + a_j)^+]` to a one-dimensional integral over `t > -a_i`; the kink of the rectifier sits exactly at the endpoint, so the integrand is smooth on the domain and 48-node Gauss-Legendre matches the closed form to 1e-15 (checked against the arccos kernel at `a = 0`, and against 8e6 Monte Carlo draws at `a != 0`).

Replacing the gate product with the exact value changed the baseline's MSE from 8.37e-5 to 8.37e-5 — **no improvement at all**, and on some MLPs a small regression. The reason is visible once expanded: at `a = 0` the exact kernel is `(sqrt(1-rho^2) + rho(pi/2 + arcsin rho))/(2 pi)`, whose covariance after subtracting the product of means is `rho/4 + 0(rho^3)`, and the gate product gives exactly `rho/4`. **The gate approximation is already correct to first order in the correlation**, and with He initialisation the off-diagonal correlations are `0(1/sqrt(width)) = 0.06`. There is nothing to recover.

Where the error actually is. Having ruled out the covariance step, we located it by brute force: run 4e5 samples through the network, measure the **true** moments of the final-layer pre-activation, and feed those into the same rectifier formula the analytic method uses, one argument at a time. Over six MLPs:

what is fed in	MSE
bundled analytic (propagated mean, propagated variance)	7.07e-5
propagated mean, **true** variance	7.07e-5
true mean, propagated variance	**1.37e-6**
true mean and variance	9.38e-7
true mean, variance, skewness and kurtosis (Gram-Charlier)	**8.85e-8**
Monte-Carlo noise floor of the measurement	6.8e-8

Three things fall out at once. **The propagated variance is fine** — replacing it with the truth changes nothing. **The propagated mean carries the entire error**, and it is drift accumulated over 32 layers, not a one-step approximation. And the Gaussian assumption, the part that looks structurally unfixable, is worth 8e-7 of 7.07e-5 — **1.2%** — and even that mostly disappears under a two-term Gram-Charlier correction using the measured skewness and excess kurtosis (both about 0.4 at the final layer).

That last correction is worth writing down, because the obvious derivation gets its sign wrong and we did too. Repeated integration by parts gives

```
I(m, n) = int_{-a}^{inf} (t + a)^m phi(t) He_n(t) dt = m * I(m-1, n-1),
I(0, n) = phi(a) He_{n-1}(-a),        I(0, 0) = Phi(a)
```

so the Hermite polynomials are evaluated at **-a**, and the third-cumulant term of the rectified mean is `-(g1/6) a phi(a)`, not `+`. With the sign flipped the correction moves the error the wrong way — 3.4e-6 instead of 8.85e-8, a factor of 39 in the wrong direction — which is exactly the sort of failure that reads as "non-Gaussian corrections do not help here."

The consequence is a target, not just a diagnosis: a method that propagated the mean faithfully would sit near `b^2 = 1e-6`, and blended with our sampler that is **3 to 5x** on the score. That is the largest number anywhere in this write-up. Sections 13.2 and 13.3 are our attempt at it — one step of Gaussian closure, then the full chain — and both failed; 14.11 and 14.14 say why, and what state a version that worked would have to carry.

7.11 Learning the residual

The natural next thought is to train a model on the residual — ground truth is cheap to generate offline (a GPU produces a `1e7`-sample label in about a second, and the released `full` split ships 1000 pre-labelled MLPs), inference is nearly free against this budget, and bundling artifacts is permitted.

The reason not to is arithmetic. A deterministic predictor cannot be used alone; it has to be blended with an unbiased sampler, and the blend of a biased predictor with squared error `b^2` and an unbiased estimator of variance `s^2` is a *harmonic* mean:

```
MSE(lam m + (1-lam) M) = lam^2 b^2 + (1-lam)^2 s^2,   MSE* = b^2 s^2 / (b^2 + s^2)
```

so the gain is `1 + s^2/b^2` and the predictor has to reach the *sampler's own* accuracy before it doubles anything. On the 100-MLP split:

predictor	`b^2`	**blended with our sampler**	**gain**
bundled covariance propagation	8.37e-5	3.18e-6	1.04x
+ linear probe on 8 cheap features	2.82e-5	2.96e-6	1.12x
what would be needed for 2x	3.31e-6	1.66e-6	2.00x

The bundled baseline already explains 99.987% of the variance in the answer; a 2x score improvement requires 99.9995%. That is a 25x reduction in an error we just showed (7.10) is not caused by the one approximation in the recursion. Two diagnostics say the remaining error is systematic rather than noisy — 22% of it is a fixed vector shared across MLPs, and a single scalar rescale per MLP removes 76% of it — which is encouraging for a learned correction, but a 3.34x linear probe converting into a 1.12x score is the honest measure of the leverage available.

There is also a cost side that the table ignores. Running the covariance recursion at all costs `O(depth * width^3)` — about 3.4e9 FLOPs, or 12.5% of the budget at the floor — which shrinks the sampler and raises `s^2` by roughly the same fraction. A predictor that only reaches `b^2 = 2.8e-5` therefore does not buy 1.12x; it buys nothing.

7.12 Propagating a third cumulant (and why the cheap version cannot work)

The diagnosis above says the mean recursion needs to stop assuming its input is Gaussian, which means carrying a third cumulant. The companion paper's [arXiv:2605.05179] general algorithm does exactly this; its `K = 2` special case (Appendix B, Algorithm 2) is *identical* to the bundled baseline — the gate `c_i` in its line 12 is `Phi(mu_i/sigma_i)` by Stein's identity — which is why everything in 6.10 came out flat.

Its `K = 3` version costs `(7/3) n^4 L`, which at `n = 256, L = 32` is **3.2e11 FLOPs against a per-MLP budget of 2.72e11**. The factorized variant is worse here, `30 n^3 L^2 = 5.2e11`, because its extra factor is the depth. Neither fits, so we tried the obvious cheap surrogate: track only the **diagonal** of the third cumulant, propagated as

```
kappa3[z_i] = sum_{jkl} W_ji W_ki W_li kappa3[a]_{jkl} ~= sum_j W_ji^3 kappa3[a_j]
```

which costs `n^2` per layer — free — and is the exact answer if the activations entering a layer are independent. Over 20 MLPs it gives **1.01x**: 8.43e-5 to 8.32e-5. Nothing.

The reason is measurable and decisive. The skewness this recursion produces at the final layer is **0.005**; the true skewness measured by sampling is **0.431**, a factor of 90. The diagonal term is not the

leading term — the activations are correlated enough that the `j != k != l` part of the contraction dominates completely. Any scheme that avoids the `n^3` tensor is avoiding the part that matters.

We also tried the reverse: inject the *true* per-layer skewness and kurtosis as an oracle into the K=2 recursion. That made things **worse** (7.7e-5 against 4.2e-5 on the first MLP), for a structural reason worth stating — a cumulant correction is only valid at the mean and variance it was derived for, and applying it on top of already-drifted moments lands it at the wrong operating point. Cumulant propagation has to be self-consistent or not done at all.

We should say clearly that this particular negative is **not ours first**. `jamesrahenry` reported it in forum topic #18097 before we ran it, with a better explanation than ours: the layerwise errors of a deep moment-propagation chain are *anti*-correlated and partially cancel, so the chain is an error-compensating dynamical system rather than an error-accumulating one; zero-mean perturbations damp, while a small consistent bias is amplified (measured there at 16:1). Injecting true cumulants replaces a compensating error with an uncompensated one. Our contribution here is only the quantitative localisation in the table above — that the *variance* channel is clean and the mean channel carries all of it — and the diagonal-`kappa3` measurement, which we have not seen elsewhere. Two other published results point the same way: `pscamillo` (#18063) found the Gaussian closure overestimates the final-layer mean by a stable factor ~0.992, so a single scalar buys 3x for free, consistent with our finding that one scalar per MLP removes 76% of the residual; and `radiant-allomancer` (#18085) built the full hybrid — RQMC through 31 layers, Gaussian closure, final-layer Edgeworth correction, offline ridge corrector — and scored **4.47e-7**, behind the plain unbiased sampler we submit here. That is the strongest single piece of evidence that the analytic direction does not pay at this depth.

Our reading is that the challenge is set exactly where the companion paper stops. Its Appendix D reports mean squared error growing like `L^K` and says plainly: "This depth scaling is worse than Monte Carlo sampling... We believe that a sample-free algorithm can be developed whose error also does not increase with depth either, but we leave this problem to future work." Phase 1 is depth 32, four times the deepest network in that paper's depth study, at the same width. The sample-free route is not merely hard here; it is the open problem the benchmark was built around.

7.13 Smoothing the one layer that is scored

Only the final layer is measured, so the whole variance budget can be aimed at one quantity, `E[ReLU(z_L)]`, and the sample mean is not the only estimator of it. `z_L` is a sum of 256 terms; for a Gaussian marginal the rectified mean is exact and closed,

```
E[ReLU(X)] = m Phi(m/s) + s phi(m/s),      X ~ N(m, s^2)
```

and `m`, `s` are linear and quadratic functionals rather than kinked ones, so they should be cheaper to estimate than the thing itself. The cost is one column mean, one column variance and 256 evaluations of `Phi` — free at this scale. We deployed it as a blend so that `lam = 0` recovers the current estimator.

It loses, monotonically, at every weight:

blend weight on the Gaussian fit	gain
0.15	0.992x (+-0.1%)
0.3	0.967x
0.5	0.914x
1.0	**0.729x**
Edgeworth (adds sample skewness)	**0.512x**

24 MLPs x 4 shifts. The error bars are tiny because the correction is a small uniform perturbation, so this is not a noise result — the direction is simply wrong, and the third-cumulant version is worse still.

The interesting part is *why*, because the obvious explanation is wrong. We measured the non-Gaussianity directly, at a reference point count of 40k pairs so that sampling error is out of the way, comparing `mean(ReLU(z))` against `rect_gauss(m, s)` computed from the same large sample, layer by layer:

layer	Gaussian gap	our sampling error	ratio
1	1.64e-3	5.58e-3	0.29
2	1.80e-3	4.72e-3	0.38
8	2.38e-3	3.74e-3	0.64
16	2.01e-3	3.28e-3	0.61
32	**1.32e-3**	**2.00e-3**	0.66

(relative, 8 MLPs.) The Gaussian target is *more accurate than our estimator at every layer*, including the last: the assumption costs 1.3e-3 where our sampling error is 2.0e-3. Non-Gaussianity is not what defeats it.

What defeats it is that the formula has to be fed. `m` and `s` come from the same sample, and their errors do not merely add to the assumption error — `m` at layer 32 is `E[a_31] W_32`, so its accuracy is bounded by the layer-31 mean we are already struggling to estimate. The closed form is better than we are, and we cannot afford its arguments. That is the same diagnosis section 7.10 reached from the other direction, where the analytic recursion's error turned out to live in the drift of the propagated moments rather than in the Gaussian assumption everyone treats as the wall.

A later run closes the argument by pushing it to the other extreme. If the closed form is more accurate than our sample at every layer, the natural move is to use it at every layer — replace the sample average by `rect_gauss(m, s)` all the way down, feeding each layer's `m` and `s` from the smoothed layer below. That compounds the feeding problem thirty-two times:

where the closed form replaces the sample average	gain
nowhere (deployed)	1.000x
final layer only	0.726x
every layer, moments from the smoothed value below	**0.021x**
every layer, moments from the raw sample	0.726x

The 48x collapse is the diagnosis stated loudly: the formula's accuracy is irrelevant because it is never given accurate arguments, and errors in `m` at layer `k` are amplified by every layer after it. The last row is the control — smoothing everywhere but feeding from the unsmoothed sample recovers exactly the final-layer-only number, so it is the *feedback*, not the smoothing, that explodes.

7.14 Choosing the lattice's modulus

This one we did not go looking for; a botched control handed it to us. To decide whether the covariance match of section 6.3 was really losing, we built a variant that pinned its point count to the plain sampler's so that the lattice, the shift and therefore the draw would be identical and the correction would be the only difference. The pin was wrong — the plain sampler runs at `n = 3089`, not the `3137` we pinned — so what the run actually measured was the same estimator at two nearby moduli:

n	raw MSE
3067	3.39e-6
3137	4.13e-6

2.3% more points and 22% worse. An unbiased sampler cannot get worse in expectation by being given more points, so either that is a draw, or the quality of a Korobov lattice with the multiplier pinned at `a = 17` swings erratically with `n`. The second reading would matter: `n` is ours to pick, every prime below the affordable ceiling costs the same, and a lattice that integrates better is free. It is also not the question section 5's CBC search asked — that varied the generating vector at fixed `n`, and CBC vectors are near-optimal by construction, so of course they were all alike. `(1, a, a^2, ...) mod n` with `a` pinned is a different animal.

Twenty-eight primes in `[2950, 3200]`, 12 MLPs x 2 shifts each, the deployed pipeline throughout. Point count is confounded with lattice quality, so we report `MSE * (n/3167)^0.89` using the measured `MSE ~ N^-0.89`, and we split the MLPs in half so that a modulus chosen on one half has its gain re-read on the other.

best / worst over 28 moduli	1.599x
pooled per-modulus standard error	15.5%
expected best/worst from that noise alone	1.85x
correlation between the two halves	+0.15 (se ~0.20)
best on half A	1.402x
the same modulus on half B	1.064x

The observed spread is *smaller* than pure noise would produce, the halves do not agree, and the winner does not survive the split. There is no modulus effect to find; the 22% was a draw, which is what the score's 8% standard error (section 9) already implied at this sample size.

The by-product is a calibration we should have had earlier. At 12 MLPs x 2 shifts an *unpaired* comparison carries a 15.5% error bar — wide enough to manufacture any conclusion one likes out of 28 candidates. Pairing on the same MLP and shift is what took the covariance A/B from that to +-1.5%, and every offline number in this note that is quoted to a percent is paired.

7.15 Aligning the lattice's coordinates with the network's inputs

A rank-1 lattice is not symmetric in its coordinates. Its low-index two-dimensional projections are well spread; its high-index ones are whatever `a^j mod n` happens to give. Meanwhile we hand `x` to `W_0` in whatever order the coordinates arrived in, which is to say we never chose the alignment at all.

The test is exact, and it is the control section 7.2 is missing. A true Gaussian `x` is exchangeable, so `E[f(x P W_0)] = E[f(x W_0)]` for any permutation `P` and the shipped `final_means` stay valid. The lattice is *not* exchangeable, so permuting the rows of `W_0` re-aligns the point set against the network while leaving points, shift, radius, whitening and budget bit for bit identical. A permutation is also the only orthogonal map that re-aligns while leaving every axis exactly where it was, which is what separates the two claims in 7.2: if permutations move the score, that experiment failed for the reason we gave and a permutation is the tool it should have used; if they do not, the reason we gave is wrong.

First pass, 12 MLPs x 2 shifts, eleven orderings including two informed ones built from the row norms of `W_1 W_2 ... W_32`:

important input -> low lattice index	0.994x
important input -> high lattice index	0.722x
ratio between them	1.376x
per-comparison standard error	12-16%

That is the shape of an exploit — feed the well-spread coordinates the inputs that matter — so we re-ran it with the sample the error bar demanded: 40 MLPs x 3 shifts, three random permutations against the deployed order, paired within (MLP, shift).

random permutation 1 / 2 / 3	0.916x / 0.888x / 0.906x
the three pooled	0.904x, se 5.5%, `t = -1.84`
comparisons the permutation won	52 / 120
best/worst over the four orderings	1.126x
the same, expected from noise alone	1.155x (bootstrap median)
best/worst at 12 MLPs	1.385x

Nothing here is an alignment effect. No ordering beat the deployed one, the pooled difference points the *wrong* way for any exploit and is under two standard errors from zero, the spread across orderings is smaller than resampling noise alone produces, and — the decisive one — that spread fell from 1.385x to 1.126x when the sample grew fivefold. A real effect does not shrink when you measure it better.

Two consequences. The 1.376x contrast that motivated the re-run was noise, so there is no coordinate ordering to exploit. And section 7.2's post-hoc explanation is unsupported: alignment does not matter, so "destroys the lattice's axis-aligned quality" cannot be why rotating onto the Jacobian's singular basis lost. We have marked that sentence rather than deleted it; section 9 is about what it cost us.

One honest limit: the 40-MLP re-run re-measured the three random permutations, not the informed orderings. What it establishes is that re-aligning the point set against the network does not move the score — which is exactly the premise the informed orderings needed.

7.16 Splitting the budget across independent lattices

A randomly shifted lattice is unbiased for any number of shifts, so the budget can be spent as one lattice of `n` points or as `S` independent replicates of `n/S` points each. The usual reason to prefer replicates is that they give an error estimate for free; the reason to suspect they might also be *better* here is that a rank-1 lattice's quality is erratic in `n` (7.14), so averaging over several draws might beat betting the whole budget on one modulus.

Equal total cost throughout, 3 MLPs, primes chosen at most `n/S`:

replicates	points each	MSE	vs one lattice
1	3167	1.971e-6	1.000x
2	1583	2.871e-6	0.704x (se 8.4%)
4	787	2.864e-6	0.769x (se 11.1%)

One big lattice wins, and it should: the whole point of a lattice rule is that its error decays faster than `N^-1/2` in the number of points, so cutting `n` in half and averaging two of them throws away exactly the property you are paying for. We record it because the free error estimate is tempting and because the `S = 8` arm could not be run at all — whitening estimates a 256x256 second moment from `n` points, and below `n ~ 2 * 256` the Cholesky is of a singular matrix. That constraint, `n >= 2 * width`, is a floor on how finely the budget can be subdivided at all, and it is worth knowing before designing around replicates.

7.17 Is the residual variance high-degree, or just high-dimensional?

Section 8 shows 37.3% of the integrand's variance sits at even harmonic degree four and above, where nothing in this note reaches. That sounds terminal, but degree is not dimension: a function can be of very high degree and still depend on only a handful of directions, and a point set that resolves a handful of directions finely is exactly what a lattice rule can be built to be. If the deep network collapses its input onto a small subspace, the high-degree residual is a function of few variables and a better rule integrates it properly.

The object that decides this is the active-subspace matrix `C = E[J^T J]` with `J = d(final activations)/du`, whose eigenvalues say how much of the output's first-order sensitivity lives in each input direction. It costs almost nothing to estimate without ever forming `J`: for a random probe `v`, `J^T v` is one backward pass and `E_v[(J^T v)(J^T v)^T] = J^T J`, so the whole spectrum is a few thousand backward passes.

input directions needed to hold	measured	flat spectrum would give
50% of the Jacobian energy	35	128
90%	**117**	230
99%	190	253
leading eigenvalue's share	3.46%	0.39%

There is real anisotropy — 35 directions instead of 128 for the first half — and it is nowhere near enough. 117 of 256 directions are needed for 90% of the energy, and no lattice rule, rotation or importance scheme resolves a 116-dimensional integrand better than a good 256-dimensional one. The residual is genuinely high-dimensional as well as high-degree, and this is the measurement that closed

the search: it says the remaining variance is not sitting in a corner of the input space waiting to be found.

7.18 Fitting the analytic recursion's drift offline

Section 7.11 rejected the whole learned-correction direction with an argument that is wrong, and it is worth correcting before the experiment that followed from noticing.

The argument was that a deterministic predictor "cannot be used alone; it has to be blended with an unbiased sampler", which makes the blend a harmonic mean and the threshold for a 2x score improvement `b^2 <= 3.3e-6`. Nothing requires a predictor to be unbiased — the score is `MSE * max(0.1, C/B)` and asks no questions — and the bundled baseline is itself deterministic and biased. Worse, the blend framing hides the reason a deterministic estimator is attractive: the covariance recursion costs `O(depth * width^3)` = 3.4e9 FLOPs, **1.25% of the budget**, so it lands on the 0.1 multiplier floor and scores `b^2 / 10`. Against our 2.877e-7 that makes the target `b^2 < 2.9e-6`, and against the best public submission `b^2 < 6.9e-7`. The bundled recursion is at 8.4e-5, so the ladder is 29x to tie and 122x to win — and section 7.10 already showed that replacing the propagated mean with the truth is worth 61x on its own. That is a live target, and we had ruled it out on a premise we never checked.

The drift is a per-layer scalar, and the scalar is a constant of the architecture. Fitting the best scalar `g_k` between the true layer-`k` mean (400,000 sphere samples) and the recursion's, over 6 MLPs:

layer	**mean `g_k`**	**sd of `g_k` across MLPs**	**residual after `g_k`**
1	0.9999	0.01%	0.22%
17	0.9924	0.16%	0.53%
32	**0.9915**	**0.24%**	**0.49%**

The recursion loses about 0.85% of the mean's magnitude over 32 layers, and it loses the *same* 0.85% in every network — the spread across MLPs is a quarter of a percent. That is a constant, fitted once offline and free at inference, which is exactly the shape a cheap fix wants.

It is not enough, and three variants agree on how much it is not:

correction	**held-out `b^2`**	**gain**
none (bundled recursion)	9.409e-5	1.00x
32 layer scalars, fitted with feedback	3.457e-5	2.4x
one global scale on the final layer	2.755e-5	**3.42x**
+ a fitted skewness term `-(g1/6) a phi(a) s`	2.754e-5	3.42x
+ a fitted excess-kurtosis term	2.755e-5	3.42x

(The last three are fitted on 50 of the mini split's MLPs and read on the other 50, against the shipped `final_means`, so there is no Monte-Carlo noise in the target at all.)

Two things die here. The scalar family tops out near 3.4x against a requirement of 29x — the 0.49% that survives the scalar is a *direction* in the 256-neuron output, not a magnitude, and it carries 97% of the remaining squared error. And that direction is not the one the theory names: the Gram-Charlier basis, which is the exact first-order shape of a non-Gaussian rectified mean and which section 7.10 showed is

worth a factor of 10 *when fed true moments*, adds **0.00x** here. Fed drifted moments it is fitting the wrong residual.

So the honest statement is narrower than "learned corrections do not work". A correction that is cheap enough to be free — a handful of constants — recovers 3.4x of a required 29x. Whatever the leading submissions are doing, it is not this, and it is not a fitted constant of any kind we could find.

7.19 The off-diagonal update is not the recursion's bottleneck either

Section 7.10 localised the recursion's error by elimination: the propagated variance is accurate and the marginal Gaussian assumption is worth 1.2% of the error, which leaves the off-diagonal update as the only candidate. The bundled code takes

```
cov_post_ij = Phi_i Phi_j cov_pre_ij
```

and that is not the Gaussian answer — the Gaussian answer is `E[X+ Y+] - m_i m_j` for a bivariate normal. It also does not need two-dimensional quadrature. Conditioning `Z2 = rho Z1 + sqrt(1-rho^2) W` makes the inner expectation an ordinary one-dimensional rectified mean, so

```
E[X+ Y+] = E_u[ (mu_i + s_i u)+ ( a(u) Phi(a(u)/b) + b phi(a(u)/b) ) ]
a(u) = mu_j + s_j rho u,    b = s_j sqrt(1 - rho^2)
```

is a single Gauss-Hermite quadrature over `u`, evaluated for all `W^2` pairs at once. At `Q` nodes it costs `Q W^2` elementwise work per layer against the `2 W^3` already spent on the congruence — a 15% surcharge at `Q = 48`, on a recursion that uses 1.25% of the budget. It is affordable.

It is worth **1.19x** (geometric mean over MLPs, `Q = 32`; one of four MLPs got *worse*), against a required 30x. So the elimination in 7.10 was sound and its conclusion still wrong: the error is not in the marginal, not in the variance, and not in the off-diagonal formula. It is in the closure itself — in propagating only a mean and a covariance through 32 layers when the third and higher cumulants are what accumulate. Making each step exactly Gaussian does not help when the distribution being propagated is not.

That closes the cheap analytic route completely. `research/bivar.py`.

7.20 A regression control variate on the exactly-known first layer

Section 6.1 uses the exact first-layer mean the crude way: it shifts the sampled `a_1` so its sample mean equals the known exact mean, then propagates. That is a control variate with its coefficient pinned to the identity. The proper estimator regresses the thing we want on the thing whose mean we know,

```
ahat = mean(a_D) - B ( mean(a_1) - E[a_1] ),   B = Cov(a_D, a_1) Cov(a_1)^-1
```

with variance `(1 - R^2)` times the plain one. This ought to be the strongest version of the idea available: `a_D` is a *deterministic function* of `a_1`, so every bit of the estimator's variance enters through the control. It costs one `256x256` cross-covariance and one solve — about one extra layer out of 32.

control	`Var(plain)/Var(residual)`**, in-sample**
`a_1` (mean known exactly)	1.64x
`a_2`	1.88x
`a_4`	2.37x
`a_8`	3.38x
`a_16`	6.52x
`a_31`	170.3x

The first row is the only one whose mean we know, and 1.64x is an *in-sample* number: fitting 256 coefficients on 6,178 points and then using them on the same points. Out of sample it does not survive — measured against the shipped `final_means` the regression control variate scores **0.929x**, slightly worse than the pinned version's 0.940x and worse than doing nothing. Thirty-one ReLU layers leave 39% of the final variance linearly predictable from the first, and recovering that costs more in estimation noise than it returns.

The last row is worth stating for what it is not. Knowing `E[a_31]` would be worth 170x, which is well past the front of the field — but that is a restatement of the problem, not a route into it. Every intermediate layer's mean is exactly as hard to obtain as the final one.

We also checked the meter itself, since a 5x gap invites the suspicion that someone is billed differently. Every spelling of the same matmul — `a @ w`, `matmul`, `dot`, `einsum` with and without `optimize`, `tensordot` — bills 133,955,584 FLOPs for a `1024x256 @ 256x256` product, and float16, float32 and *int8* all bill the same while float64 bills exactly double. There is no cheaper spelling. `research/cvreg.py`.

The same probe run over our own billed path confirms the float32 cast of section 0 is fully collected: matmul, `x^T x`, Cholesky, `solve`, `sum` and `mean` all report float32 output and exactly half the float64 figure. One operation is not ours to control. `flops.stats.norm.ppf` returns **float64 whatever it is given** — feeding it float32 changes neither the output dtype nor the bill — so the inverse-CDF step is billed at the float64 rate no matter what, at 166 FLOPs per element. At the deployed `npair` that is 1.32e9 FLOPs, **0.48% of the budget**, of which at most half could ever be recovered. We mention it because a reader auditing the estimator will notice the `.astype(fnp.float32)` sitting *after* the `ppf` and wonder whether it is too late. It is too late, and it does not matter.

7.21 The closure's residual is not predictable from the closure

Section 7.18 fitted a handful of constants to the recursion and reached 3.42x of a required 29x. The obvious next move is a real regression, and the obvious question before building one is whether the target carries any signal at all.

After the best global rescale, write the residual per output neuron as `r_j = t_j - g m_j` and regress it on twenty features that the recursion already computes: the final-layer mean and standard deviation, the pre-activation mean, standard deviation and their ratio `alpha`, `Phi(alpha)` and `phi(alpha)`, `alpha^2` and `alpha^3`, four norms of the final weight column (`L2`, `L1`, third and fourth power sums), a second-order projection `mu_{D-1} (W_D * W_D)`, two normalised versions of the mean and standard deviation, and three MLP-level summaries. 100 MLPs, 50/50 split by MLP, so 25,600 training rows against 20 features.

	train	held out	held-out residual `R^2`
raw closure	7.32e-5	9.41e-5	—
+ global scale (`g = 0.99247`)	2.42e-5	2.75e-5	—
+ ridge, `lambda = 1e-6`	2.31e-5	2.91e-5	**-0.048**
+ ridge, `lambda = 1e-2`	2.32e-5	2.87e-5	-0.033
+ ridge, `lambda = 1`	2.37e-5	2.78e-5	-0.006

The held-out `R^2` is negative at every regularisation strength: the features explain *none* of the residual, and the best thing ridge can do is regularise itself back to the global scale it started from. This is not a sample-size problem — 25,600 rows and 20 features — and it is not a linearity problem in the usual sense either, because the features include the exact nonlinear quantities the closure's own error expansion is written in.

There is a reading of this that we think is right. Both `t_j` and `m_j` are *deterministic* functions of the weights; the residual is not noise, it is a chaotic function of 2.1 million numbers. Thirty-two layers of rectification mix it thoroughly enough that no summary statistic the recursion produces along the way retains a usable correlation with where it ended up — the same conclusion 7.5 reached about the network's realised structure, arrived at from the analytic side. And it bounds the learning route as well: a trained model on these features cannot beat a ridge that finds nothing, so it would have to work from the raw weights, against a target whose best cheap predictor explains 0% of it.

`research/feat.py`. The label generator for a raw-weight attempt is written and validated — reference means from `1e7` i.i.d. samples reproduce the shipped `final_means` to 4.5e-09, three orders below the 2.9e-06 that would be needed — but on this evidence we would not start it expecting to win.

7.22 The order-`K` rate has already collapsed at this depth

Section 7.12 argues from cost that `K = 3` does not fit and from the paper's own depth study that the sample-free route degrades with depth. Both are arguments about whether we *can* run a higher order. Neither says what we would get if we could, and that is the number that decides whether the cost is worth fighting for. So we measured it.

The companion paper's error rate is `O(1/n^K)` in the width. That exponent is the whole question. If it holds, each extra order is worth a factor of `n = 256` and the third order lands somewhere near `2e-07`, which at the multiplier floor would score `2e-08` and beat everything we have by an order of magnitude. If it does not hold, each extra order is worth a constant and the family is closed.

One ratio was already in hand and it was ambiguous. Implementing Algorithms 1 and 2 from Appendix B directly — mean propagation with a scalar variance, and full covariance propagation with the `Phi(mu/sigma)` gate — and scoring them against the eight bundled networks' own layer means gives `9.3720e-04` at `K = 1` and `5.7560e-05` at `K = 2`. That is a factor of 16.3 in mean squared error, or 4.0 in root mean squared error. The rate predicts 256. But a single ratio cannot separate "the rate is broken" from "`K = 1` is simply a poor member of the family", which it might well be — it propagates one scalar variance for the whole layer, and the rate may not describe it at all.

A rate in the width has to be measured in the width. Fresh He-initialised networks at 64, 128, 256 and 512, depth held at 32, truth by 4.2M antithetic moment-matched Monte Carlo points, four networks per

width. (The bundled networks' weight standard deviation is `0.08826` against `sqrt(2/256) = 0.08839`, a ratio of `0.9986`, so the synthetic ensemble is the same ensemble.)

```
depth 32 fixed, width swept        relative rmse
  width        K=1          K=2       K=1/K=2
     64   1.9104e-01   4.0627e-02          4.70
    128   6.3051e-02   1.8917e-02          3.33
    256   3.2908e-02   8.5863e-03          3.83
    512   2.4385e-02   5.5448e-03          4.40

d log(rel rmse) / d log(width)    K=1  -0.985    K=2  -0.976
the rate would give               K=1  -1        K=2  -2
```

`K = 2` decays as `n^-0.98`. It decays *at the same rate as* `K = 1`. Going from the first order to the second does not change the exponent at all; it multiplies the constant by about four and stops. Whatever the second order is buying at depth 32, it is not the rate the method was designed around.

Sweeping the depth at fixed width says the same thing from the other side, and adds the mechanism:

```
width 256 fixed, depth swept
  depth        K=1          K=2       K=1/K=2
      4   2.7708e-02   4.4486e-03          6.23
      8   3.3627e-02   6.6733e-03          5.04
     16   3.5074e-02   8.6559e-03          4.05
     32   3.2908e-02   8.5863e-03          3.83

d log(rel rmse) / d log(depth)    K=1  +0.081   K=2  +0.322
```

The order is worth 6.2x at depth 4 and 3.8x at depth 32. The value of an extra cumulant is itself being eroded by depth, monotonically, over the whole range — which is a sharper statement than "the method degrades with depth", because the absolute error barely moves (`+0.08` and `+0.32` in the depth exponent, far milder than the `L^K` the paper's Appendix D reports). The error is not growing much with depth. What depth destroys is the *benefit of the correction*: each extra order of cumulant is buying less the deeper the network is, and by depth 32 it has been reduced to a constant near four.

That closes the family by arithmetic rather than by cost. Extrapolating one more order at the measured constant puts `K = 3` at `3.5e-06`, which even pinned to the multiplier floor scores `3.5e-07` against the `2.09e-07` we already have. Two more orders would be needed to win, and `K = 4` costs several times the budget by section 7.12's accounting. The gap between what the third order would have to deliver and what the second order's measured behaviour says it will deliver is a factor of about seventeen, and nothing in the two sweeps above suggests where seventeen would come from.

A caution about reading the field, since we got this wrong once and it changed the conclusion. On a first scan of 424 submissions the multiplier floor looked like a graveyard: twenty-eight entries there, median raw error `5.98e-05`, essentially identical to our `K = 2` number, which reads as "the field walked to this wall and stopped". Widening the scan to 1405 submissions replaces that picture entirely. There are 150 entries at the floor, their median raw error is `3.44e-06` — seventeen times better than `K = 2` — and the

best of them is `9.13e-07`, which at a multiplier of `0.1` scores `9.13e-08` and beats the `2.09e-07` we ship by a factor of 2.3. Somebody has a deterministic estimator that costs a tenth of the budget and is sixty-three times more accurate than the second cumulant order. Whatever it is, section 7.22 says it is not the third order, and section 7.23 says it is not a better point set.

7.23 Interlaced higher-order digital nets

Section 10.10 measures our error as pure variance falling at `n^-1`, and the front of the field at roughly thirty-two budget-multiples ahead of us and deterministic. A deterministic rule can only be that far ahead at our point count if its squared error falls with a *better exponent*, and there is exactly one classical construction that supplies one. Dick's interlacing takes `alpha` source coordinates of a digital net and weaves their binary digits into a single output coordinate, so that bit `i` of source `a` lands at output position `alpha*i + a`. The resulting order-`alpha` net integrates functions with `alpha`-th order mixed derivatives at `n^-2*alpha` in squared error, against `n^-1` for an ordinary lattice or net. Two rules that agree at a thousand points are a factor of forty-nine apart at forty-nine thousand, which is the size of the gap we are trying to explain.

The reason this is a measurement and not a calculation is that our integrand has no mixed derivatives to speak of: it is a composition of thirty-two rectifiers, and every kink is a place where the theory's smoothness hypothesis fails. The theory then bounds a quantity that is infinite, which says nothing about the error that actually occurs.

Sobol' in `256*alpha` dimensions at 30 bits, interlaced, randomised by digital shift, `n = 2^15` for the nets against the nearest prime for the lattice, eight networks, four randomisations each:

rule	mean mse	median	vs lattice	wins
lattice	2.4438e-07	2.0433e-07	1.000	0/8
sobol a=1	3.0659e-07	2.1426e-07	0.797	4/8
interlaced a=2	3.3202e-07	3.0697e-07	0.736	1/8
interlaced a=3	3.8999e-07	2.9030e-07	0.627	2/8

Every rule is worse than the lattice on the mean, and interlacing makes it monotonically worse as the order rises: 1.36x at `alpha = 2`, 1.60x at `alpha = 3`. There is no trace of a better exponent — the whole table spans a factor of 1.6, where the hypothesis needed a factor of thirty. Interlacing does what it does by spending digits: an order-`alpha` net keeps only `30/alpha` bits per source coordinate, so it buys smoothness-exploiting structure with resolution, and on an integrand that has no smoothness to exploit only the cost is left. Plain Sobol' losing to the lattice by 1.25x is the same story without the interlacing.

This closes the exponent branch of section 10.10's dichotomy by measurement. What remains is the cost-per-point branch.

7.24 Carrying the sample as a mean plus a low-rank factor

Section 7.23 closes the exponent branch, which leaves cost per point. A layer costs `n W^2` because every point is carried as a full 256-vector. If the sample's fluctuation about its own mean occupied `r << 256` directions, the layer could be carried as a mean plus a rank-`r` factor — reconstruct, rectify, recompress — at about `3 n W r`, which is five times cheaper at `r = 16` and ten at `r = 8`, and the saving goes straight into more points. Depth makes this plausible rather than absurd: a product of thirty-one rectified

random matrices is strongly rank-concentrating, and section 10.7 measures the effective rank of the *error* at about five.

The activations do concentrate, and not slowly:

layer	top8	top16	top32	top64	part.ratio	rank for 1e-6
1	0.157680	0.278551	0.459458	0.681472	105.93	256.0
2	0.217038	0.365576	0.564189	0.770150	75.74	255.8
4	0.322834	0.498690	0.696987	0.855882	46.08	253.0
8	0.505233	0.680470	0.831049	0.928158	18.76	231.0
16	0.660051	0.804222	0.902293	0.960486	9.79	220.5
24	0.790550	0.886398	0.948147	0.980911	4.40	196.2

The participation ratio falls from 106 to 4.4 across the depth, so by layer 24 the fluctuation really does live mostly in a handful of directions. It is the last column that kills the idea. Capturing all but one part in a million of the energy still needs 196 directions at layer 24 and 231 at layer 8 — the spectrum has a long, flat tail that carries very little energy each but a great deal in aggregate. Truncating at `r = 64`, which is only a 1.3x saving, discards two percent of the fluctuation energy, and two percent of the energy is fourteen percent in amplitude: a systematic error two hundred times larger than the `6.4e-04` relative error we are trying to beat, and a *bias*, so no number of extra points removes it. The trade is not close, and it gets worse as `r` falls because that is where the saving is.

Both branches of section 10.10's dichotomy are now closed by measurement, for every mechanism we could construct. That does not mean no mechanism exists — section 7.25 shows somebody has one — only that it is none of these.

7.25 The multiplier floor is exactly a wash, which is why so many sit on it

Section 11.8 argued from an extrapolated slope that the multiplier floor was not a door. The scan in 12.1 makes that worth redoing as a measurement, because 150 of the 1405 submissions we can see are sitting on the floor and the best of them scores `9.13e-08` against our `2.09e-07`.

Pinned to the floor, the budget buys 4793 lattice points. Measured on the eight bundled networks over eight shifts, and beside the closure, whose 0.4% leaves the other 9.6% for the sample:

arm	mean mse	median	score at floor
sample only	1.7551e-06	1.2147e-06	1.7551e-07
closure only	5.7560e-05	3.4501e-05	5.7560e-06
closure x beta	1.4850e-05	1.3845e-05	1.4850e-06
closure x oracle	1.4079e-05	1.3063e-05	1.4079e-06
a*c + b*s	1.7590e-06	1.4182e-06	1.7590e-07

Two results, and the first is a correction to our own record. Rescaling the closure by the single best scalar per network — the idea section 10.3 valued at about 100x — is worth **4.1x** here, from `5.76e-05` to `1.41e-05`, and that is the oracle, with the truth in hand. Estimating the same scalar from the small sample recovers almost all of the oracle's gain (`1.49e-05` against `1.41e-05`, so the projection is indeed cheap to measure), and it does not matter, because the ceiling itself is two orders above the sample it was supposed to improve. Fitting two coefficients offline on half the networks and scoring on the other

half lands on `a = 0.107, b = 0.892` and `1.76e-06` — statistically identical to dropping the closure entirely. The closure contributes nothing once a sample of any size is present.

The second result is the arithmetic of the floor itself, and it is cleaner than the extrapolation in 11.8. This arm's error falls as `n^-1.005`: at 48947 points the same estimator gives `1.74e-07` (section 10.10), at 4793 it gives `1.76e-06`, a factor of 10.1 for a factor of 10.2 in points. With a slope of exactly `-1` the score is

```
score(n) = A n^-1 * max(0.1, 0.98 n / n_full)
```

which for any `n` above `0.102 n_full` equals `0.98 A / n_full` — **independent of** `n`. The whole region from the floor to the full budget is flat. Sitting on the floor costs nothing and gains nothing, and that is why a tenth of the field is there. Our deployed estimator's slope is `-1.32` rather than `-1.0`, bought by the pruning and the contraction, and a slope steeper than `-1` tilts the flat region in favour of more points, which is why we spend the whole budget.

Measured head to head on the same eight networks, the deployed estimator scores `1.53e-07` (raw `1.74e-07` at a multiplier of 0.879) against the floor arm's `1.76e-07`. The floor is not a door in either direction. The submission at `9.13e-08` is therefore not exploiting the floor; it is a genuinely better estimator that happens to be cheap, and nothing in sections 7.22 through 7.24 identifies it.

7.26 What each piece of the estimator is actually worth

Every section above adds a piece and reports what the piece adds. None of them reports what the whole thing is worth against the thing it replaced, so we measured that: five arms at one row count (97894), eight shifts, the eight bundled networks, scored against `alm`.

arm	mean mse	vs iid	rung adds
iid	3.7335e-07	1.000	1.000
iid+anti	4.5392e-07	0.823	0.823
iid+anti+white	1.8452e-07	2.023	2.460
+sphere	1.8121e-07	2.060	1.018
lattice	1.7222e-07	2.168	1.052

The whole construction is worth **2.17x over plain Monte Carlo**, and whitening is 2.46 of it. The sphere projection adds 1.8%, and the entire quasi-Monte Carlo apparatus -- the Korobov lattice, the tent transform, the shift -- adds 5.2%.

We had not expected the lattice to be worth nothing. It is consistent with everything else in section 7: after the two exact conditions are imposed, what is left of the variance is high-degree and isotropic, and that is precisely the regime in which a lattice has no advantage over independent draws. Sections 7.23, 7.24 and 7.2 each tested a different way of exploiting structure in the integrand and each found none; this row says the same thing from the cost side.

The second row is the surprise. **The antithetic pair, on its own, is a loss.** Folding two rows into one pair halves the independent draws the even part of the integrand sees, and inverting 0.823 says the even part carries 61% of the variance, so the trade is bad. It earns its place only through what it does downstream, which is the subject of the next section.

7.27 Centring does not replace the antithetic pair

If the pair is a loss on its own and is kept for the exactness it supplies, it is worth asking what exactness that is. The obvious answer is the input mean: `{x} u {-x}` has sample mean exactly zero, which makes the degree-one component of the integrand integrate exactly, and the chaos spectrum puts 37% of the variance in degree one. Whitening supplies the other exact condition, the second moment, but it is built from the raw second-moment matrix and leaves the mean alone.

The mean condition does not obviously need the pair. Centre the sample and whiten its covariance instead, and both conditions hold exactly on `N` independent draws rather than `N/2`, for the cost of one extra vector: the shift folds into the first layer as a constant row, `z = x W0 - xbar W0`. On the arithmetic above that is worth 1.8x, which would have been the largest single gain in this document.

It loses, and it loses clearly:

```
          arm     mean mse       median   vs anti   wins
         anti   2.0604e-07   1.5904e-07     1.000    0/8
anti+centre   2.0604e-07   1.5904e-07     1.000    5/8
      centre   3.2807e-07   2.4092e-07     0.628    0/8   geo 0.666 +-0.052
 centre+sph   3.2570e-07   2.3711e-07     0.633    0/8   geo 0.668 +-0.053
```

The `anti+centre` row is the control, and it lands on `anti` to four figures, which is what it has to do -- the pair already centres, so the shift is exactly zero and the arithmetic is a no-op. The implementation is therefore right and the loss is real.

The reason is that the pair supplies a second exact condition we had not credited it with. For the antithetic set, `relu(z_i)^2 + relu(-z_i)^2 = z_i^2` identically, so the sample second moment of the first *post*-activation is exactly determined by the sample second moment of the pre-activation, which whitening has already made exact. **The pair makes the diagonal of the layer-one post-activation second moment exact**, 256 quantities with no sampling error at all, and centring supplies none of that. Section 3 proved the pair is exact at layer 1 for the mean; this is the same statement one moment higher, and it is the reason the pair survives a trade that looks bad in isolation.

7.28 The pruner is already at its optimum, and overshooting is why

The deployed pruner drops units that never fire in a 512-point pilot. Measuring what it actually removes turned up an apparent defect: at that pilot size it calls 1539 of 8192 units dead when only 928 of them truly are, so 611 units that do fire -- at rates below about one in five hundred -- are zeroed. That is a bias, and it does not average down.

It is not a defect. At a matched FLOP budget, with the pilot charged against it:

```
      arm    mean n     mean mse   vs none   wins
     none     48947   1.7485e-07     1.000    0/8
 pilot512     68377   1.3293e-07     1.315    7/8
   oracle     58872   1.4513e-07     1.205    6/8
```

The deployed rule beats no pruning by 1.315x, and it also beats the *exact* mask -- the units that never fire in 262144 draws, which is the most any pruner could safely remove. Cutting more than is safe is

better than cutting exactly what is dead, because a unit firing once in five hundred inputs contributes almost nothing to the mean and a full unit's worth of noise to the sample.

That inverts the usual reading, so the natural next question is how far it goes. Separating aggressiveness from pilot size -- measure rates on a fixed 2048-draw pilot, then cut at an explicit threshold -- gives a dial:

```
          rule   width       n     mean mse  vs deployed   wins
   tau=0.00000   218.9   63539   1.3721e-07        0.826    2/8
   tau=0.00195   210.3   68646   1.1330e-07        1.000    0/8
   tau=0.00781   202.3   73919   2.3846e-07        0.475    0/8
   tau=0.03125   190.1   83367   2.9242e-06        0.039    0/8
   tau=0.12500   171.7  101845   2.0343e-04        0.001    0/8
```

`tau = 1/512` is what we ship, and nothing on either side of it wins. The bias grows faster than the rows it buys, by a factor of twenty-five over the next two steps of the dial. This is the only knob in the estimator we have found sitting exactly on its optimum by accident, and it is worth recording that the accident was the pilot size rather than any reasoning of ours.

A cascade to a fixed point -- delete a dead unit's column and row, remeasure, repeat -- closes at a mean width of 227 of 256 at threshold zero and 196 at threshold `1e-2`, for a cost per row of 0.82 and 0.61. So the entire pruning lever, taken to its limit, is worth at most 1.6x in rows and less than that in score. The cost side has no factor of eight in it.

7.29 The meter charges float16 at the float32 rate

flopscope's cost model is dtype-dependent -- `est_cov.py` notes that a float64 buffer is billed at twice the float32 rate -- which raises the obvious question of whether half precision is billed at half. It is not:

```
matmul (4096,256)@(256,256)   ref 2*M*K*N = 5.3687e+08
  float64    1.0716e+09   1.9961 x ref      float64 / float32 = 2.0000
  float32    5.3582e+08   0.9980 x ref
  float16    5.3582e+08   0.9980 x ref      float16 / float32 = 1.0000
```

The symmetry discount is real but small: an `as_symmetric`-tagged sandwich is billed at 100,597,504 against a dense 133,955,584, a 25% saving, and it applies only to explicitly tagged buffers -- an untagged `A.T @ A` is billed at the full `2*M*K*N` even though its result is symmetric. Applying the 25% to the order-3 cumulant budget of section 7.12 brings `3.2e11` down to `2.4e11`, inside the `2.72e11` budget, so the cost objection to `K = 3` does dissolve. The accuracy objection does not, and section 10.11 makes it much sharper than section 7.22 managed to.

7.30 The residual-wall arbitrage is worth 1.03x on our hardware, which is the wrong hardware

Threads 18099, 18108 and 18122 describe, with measurements, that arithmetic on arrays reachable through `flopscope.numpy` can avoid the instrumented path and be billed only through the residual wall-clock term at `lambda = 1e11` FLOP/s. One participant put the arbitrage at 1.5x to 2x on the grounds that a

modern core runs dense float32 GEMM at 1.5e11 to 2e11 FLOP/s. If that were the size of it, moving our hot path off the meter would be the cheapest score we could buy.

It is not the size of it. The v0.10.0 announcement limits participant code to one physical core, and one core is exactly where `lambda` was set:

```
1-core numpy float32 GEMM   (97894,256)@(256,256)   1.028e+11 FLOP/s
1-core numpy float64 GEMM                            5.293e+10 FLOP/s
lambda                                               1.000e+11 FLOP/s
```

Our shipped estimator bills `2.391e11` instrumented FLOPs. The same arithmetic run outside the meter would take `2.391e11 / 1.028e11 = 2.33` s of residual wall and be billed `lambda * 2.33 = 2.33e11`. That is a 3% saving, and it is 3% in exchange for the eligibility risk the organisers set out in thread 18132. The float64 lane is worse than the meter, not better, by a factor of two.

The caveat, which we think is larger than the result. That table was measured on our development machine, and `lambda` is charged against the *evaluator's* core, which we have never benchmarked. The number 1.028e11 is a property of this box: numpy on OpenBLAS, no AVX-512. A core that has AVX-512 runs dense float32 GEMM at roughly twice that, which would make the arbitrage 2x rather than 1.03x, and the organisers' own statement in 18122 — "it would be cheaper for you to use flopscope instead for the same computations" — rests on the same unmeasured quantity from the other direction.

Two entries above us on the board say the quantity is not what either of us assumed. `ely2sh` (#323861) and `dstepanov` (#324810) have instrumented shares of 0.0000 and 0.0001: essentially all of their charged cost is residual wall time. Inverting `ely2sh`'s row — 3.09 s of residual per network, mean public MSE `5.57e-08`, and the `8.5e-03` variance-per-row constant our own sampler measures — puts the arithmetic they performed at about `6.4e11` FLOPs in 3.09 s, i.e. **2.1e11 FLOP/s on one physical core**, twice `lambda`. We do not claim that inversion is what their code does; it assumes they sample and that their variance per row matches ours. We report it because it is the only estimate of the evaluator's uninstrumented throughput anyone has published, and because if it is even approximately right then the residual lane is priced at half its cost and the sentence above about 1.03x is true only of our laptop.

The clean way to settle it is a probe submission that does a known number of uninstrumented float32 operations and reads `residual_wall_time_s` back out of the public telemetry. That is one submission, it scores badly by construction so it cannot displace anything, and it would replace both our guess and the organisers' with a number. We ran out of Phase 1 before running it; it is the first thing we would do in Phase 2, and we would rather flag the hole than leave a measurement standing that was taken on the wrong machine.

What does survive is the narrower claim: on hardware where a core runs float32 GEMM at `lambda`, the arbitrage is worth nothing, and the "1.5x to 2x" figure quoted on the forum in July predates the one-core limit and was never true at the core count now enforced. Whether the evaluator is such a core is open.

7.31 The network never becomes linear, so the kink set never shrinks

A rectifier unit that is positive on every input computes the identity, and one that never fires computes zero; only the units in between need to be sampled at all. Thread 18106 exploits exactly this at the last two layers. If the split sharpened with depth -- if a depth-32 network were mostly a linear map with a

thin nonlinear seam -- the sampled problem would be much narrower than 256 wide and both the cost and the variance would fall with it. That would be a structural gain rather than a constant factor, which is the only kind that could close a three-order-of-magnitude gap.

`research/onkink2.py` measures the split on 2^20 draws, eight networks:

```
tolerance 1e-3
  layer :      0     3     6     9    12    15    18    21    24    27    30
  dead  :      0     2    11    28    36    44    54    61    60    75    73
  on    :      0     2    12    27    39    42    51    60    63    66    79
  kink  :    256   252   233   200   180   170   151   135   133   114   104
  kink-only cost per row 0.5068  ->  2.0x more rows

mean units active on a single row, by layer
    128   126   127   126   130   129   129   126   132   123   131
```

The last line is the answer. A single input row lights up 128 of 256 units at layer 0 and 131 of 256 at layer 30: exactly half, at every depth, with no trend. The network does not collapse and it does not linearise. The kink set does thin -- 256 to 104 over 32 layers -- but even granting the optimistic accounting in which a kink unit only ever reads from other kink units, the whole structure is worth 2.0x, and the honest version where the on-units still have to be carried is worth what section 7.28's pruner already collects.

This also settles what thread 18106's estimator is: its raw 2.18e-07 and our 2.38e-07 are the same number, and its structural machinery is buying the same factor of two ours does by a different route.

7.32 The MLP names are public but the seeds are not recoverable

Every submission's per-MLP telemetry is served without authentication and carries a `mlp_name` for all 100 graded networks -- the private 50 included. `whestbench/naming.py` derives that name deterministically from the network's own seed:

```
def generate_mlp_name(seed: int) -> str:
    fake = Faker()
    fake.seed_instance(int(seed))
    return f"{_slugify(fake.first_name())}-{_slugify(fake.last_name())}"
```

A deterministic name is an invitation to invert it: recover the seed, regenerate the weights offline, spend unbounded compute on the answer, and ship a table. The invitation is refused by the generator. `create_dataset` draws each seed from `secrets.randbits(63)`, and faker's name space is of order `3e6`, so every name has roughly `3e12` preimages. A preimage is cheap to find and useless: it names a different network.

The exposure is worth reporting anyway, because the argument that closes it is an accident of the seed width rather than a design decision, and Phase 2 could narrow the seed space without anyone noticing the consequence.

7.33 Memorising the public half cannot reach the top of the board either

Phase 1 grades on 50 public networks and 50 private ones (thread 18118). The public half is distributed with the challenge, so its answers can be baked offline at any precision and looked up at run time from a hash of the weights. That is the obvious explanation for a leaderboard entry far below what any estimator can do, and it is arithmetically insufficient here. Scoring the public half perfectly and the private half at our own raw error gives

```
(50 * 2e-10 + 50 * 2.38e-07) / 100 = 1.19e-07
```

which is better than us and nowhere near `3.6e-09`. Reaching the top of the board requires the private half to be estimated at `7.2e-09`, and no lookup table reaches a network it has never seen. Whatever produces those scores works on unseen networks, which is what makes section 10.12 worth writing.

7.34 Scrambled Sobol' is not the difference between us and the entry above us

Section 5 swept generating vectors at one point count and concluded that any decent rank-1 lattice buys about the same and choosing among them is wasted effort. It never asked whether the *family* was right at the count we ship, and two measurements suggested it was not. Ours: a ladder of the deployed pipeline gives `mse * n` flat at `8.4–8.7e-03` through `n = 44,293` and then rising to `9.61e-03` at 68,639 and `1.05e-02` at 97,883 — which is what a rank-1 lattice whose quality is erratic in `N` looks like, and we ship `npair = 48,947`, just past that turn. Theirs: thread 18053 reports "Korobov lattice: much worse (rank-1 quality is violently N-sensitive)", and thread 18106 ships scrambled Sobol' at `N = 84,992` for raw `2.18e-07` against our `2.38e-07` at a comparable cost per row, about 26% better per sample.

A digital net has no `N`-sensitivity to have, so the prediction was specific: at `n` below the turn the two should tie, and above it Sobol' should stay flat while Korobov rises. Four arms, eight networks, eight shifts, everything else identical — whitening, sphere projection, antithetic pair, the deployed pilot-512 pruner. Paired geometric mean against the shipped point set, so a number above 1.000 is a win for the challenger:

n	sobol	sobol+tent	roberts+tent
22,147	0.853 +- 0.105	0.973 +- 0.059	0.849 +- 0.071
44,293	0.933 +- 0.088	1.004 +- 0.135	0.892 +- 0.143
68,639	1.169 +- 0.142	1.114 +- 0.111	1.070 +- 0.196
97,883	0.949 +- 0.129	1.036 +- 0.088	0.995 +- 0.119

There is no trend and every entry is within about one standard error of a tie. The prediction is falsified. Worse for the diagnosis that motivated it, the `mse * n` ladder is just as erratic for the digital net as for the lattice — Sobol' runs `9.84e-03, 9.23e-03, 8.90e-03, 1.07e-02` across the same four counts. The rise we attributed to Korobov's `N`-sensitivity is the measurement's own noise at eight networks, not a property of the point set. Section 5's original conclusion survives intact, and we withdraw the reading of the ladder that this experiment was built on.

What that leaves is the 26% per-row gap to 18106, still unexplained and now known not to be the point set. Whatever it is lives elsewhere in that pipeline.

8. A ceiling we derived, and the measurement that made it irrelevant

We spent the last night of the competition trying to bound what this whole approach could ever be worth, arrived at a clean answer, and then found that the answer bounds a mechanism that contributes nothing. Both halves are here, because the second half is the useful one.

The argument. Because the network is bias-free ReLU it is positively homogeneous, the radius factors out exactly, and the integral lives on the sphere of radius `E||x||`. On that sphere our devices act on the harmonic decomposition of the integrand. Antithetic pairing averages `f(u)` and `f(-u)`, which annihilates **every odd degree** exactly, not merely the linear term; whitening forces the sample second moment to `(R^2/d) I`, which annihilates degree two. Everything of even degree four and above is untouched by construction. Measured on 20,000 antithetic pairs against the highest-variance neurons of three networks:

	share of the integrand's variance
degree 1, killed by pairing	32.5%
surviving the pairing (all odd degrees gone)	40.9%
of that, degree 2, killed by whitening	3.6%
even degree >= 4, reachable by nothing here	**37.3%**

Those numbers are right, and the conclusion we drew from them — a ceiling of `1/0.373 = 2.68x` over plain Monte Carlo — is wrong twice over.

It is wrong arithmetically. 40.9% is the variance of a *pair mean*, and a pair costs two forward passes. At equal cost the pairing is worth `1/(2 x 0.409) = 1.22x`, and the whole family `1/(2 x 0.373) = 1.34x` — below the 1.755x our own estimator already achieves. Any bound our deployed code violates is not a bound. We had compared a per-pair variance ratio against an equal-cost empirical gain and not noticed they were different quantities.

It is wrong structurally, and this is the part that matters. The equal-cost ladder in section 0 says the pairing is worth 1.007x and whitening 1.017x. The harmonic argument bounds exactly the devices that turn out to do nothing, and is silent about the lattice, which does all of the work — a randomly shifted lattice reduces variance through the decay of the integrand's Fourier coefficients on the dual lattice, not by annihilating harmonic degrees, so no decomposition of the *pointwise* variance bounds it at all.

The retrospective value of the exercise is one honest number: the exact arc-cosine Gram correction of section 6.3 is worth 1.048x because degree two is worth 3.6% of the variance, and no correction to a 3.6% contribution was ever going to be worth its 23% in array calls. That much survives.

> **Superseded — read section 14 with this one.** Both halves of the withdrawal above are wrong, for reasons we found two days later and could not have found here. The 3.6% is an artefact of measuring the spectrum on the *highest-variance neurons*; the score weights all 256 equally, and under that weighting degree two carries 18.1%, the unreachable remainder is 26.1%, and the equal-cost bound is `1.915x` — above our deployed `1.731x` rather than below it, so nothing violates it and the ceiling stands (14.1, 14.2). The structural objection fails too: the lattice is *not* doing the work. Section 12.4 measures the lattice at `0.97x` and the whitening at `2.00x`, which is precisely a degree-2 annihilator, so the

> harmonic decomposition bounds the device that matters after all. What survives unchanged is the arc-cosine Gram correction being worth nothing — but for the opposite reason to the one given here: not because degree two is small, but because the input whitening has already taken it.

Where the front of the field is. The submission API answers `GET /api/v1/submissions/<id>` for any id, so the score distribution of the whole round is readable. Scanning ids 323300-323660 — all of them round 1428, the same round as ours — the leading public raw MSE is **5.211389986214954e-08 at a multiplier of 1.010**, which is 10.9x better than plain Monte Carlo at the same sample count and **four times beyond the ceiling above**. Two separate submissions report that raw MSE identical to sixteen digits, so whatever produces it is deterministic; it is not a sampler with a random shift.

Two further readings of the same endpoint, twelve hours apart, sharpen this. The front moved to a raw 5.969e-08 at a multiplier of 0.907 while the middle of the field sat at 9e-08 to 1.1e-07 — against our raw 3.047e-07 at 0.944, the leaders are **5.1x better at the same cost**, and the difference is entirely in the estimator: budget placement is flat (1.1) and no point set we have measured, lattice or scrambled Sobol', moves more than 2x (7.8). The scan also settles a detail of the meter. Several graded submissions report a multiplier *above one* — 1.154 and 1.237 for the same bit-identical raw error — without being zeroed, so the exhaustion check and the multiplier do not read the same quantity: a submission can be billed above `B` on the combined cost and still be scored. Those two entries differ only in wall-clock noise, which is a 7% swing on the same predictions, and the pattern of repeated identical raw errors across submissions says the participants above us are resubmitting to draw from it. Our own residual wall time is about 2 ms, so that lever is not available to us and not worth acquiring.

That is the most useful thing we learned all week, and it is a negative result about our own direction: no refinement of the point set, no further moment matching, and no rearrangement of these identities reaches 10.9x, because 10.9x is outside the space they span. We can also say what it is not. It is not a Gaussian closure: substituting `E[ReLU(N(m,s))] = m Phi(m/s) + s phi(m/s)` for the sample average at the final layer alone costs 1.38x, an implied bias of 8.3e-7 in MSE, which already exceeds the leaders' total error. It is not low-rank propagation: 13.5% of the centred activation energy sits beyond rank 32 even at depth 31 (7.6). It is not gate structure: `P(on)` measured over 20,000 samples stays in **0.458-0.525 across all 32 layers** — a bias-free He-initialised network never saturates, only 36% of its 8,192 units are decidable even at a 1e-3 threshold, and evaluating the remaining 5,231 individually costs 1.3x *more* than the dense matmul it replaces (7.3). It is not a subspace method: the active-subspace matrix `E[J^T J]` needs 117 of 256 input directions to hold 90% of its energy, against 230 for a flat spectrum (7.17). And it is not a cheaper dtype — flopscope's billing table rates float16 and float32 identically at 1.0.

What is left is the one thing we never built: a component fitted offline against many networks rather than derived from one. Nothing in the rules forbids it, the generating distribution is fixed and public, and a reference target costs about 1.4 s of GPU time per network at the accuracy that would be needed (the shipped ground truth uses 1e9 samples and 4.211e15 FLOPs, but its own Monte-Carlo noise is 3.5e-11 — a proxy 100x cheaper is still 100x more accurate than anything we need to beat).

The rest of this section is what we believed before the measurements above, and we have left it standing.

Score is variance times cost. On the **cost** side, 99.6% of the bill is the matmul, and the matmul is irreducible unless one is willing to approximate the network — pruning gives 8% with a certificate (7.3), and low-rank gives less than nothing (7.6). On the **variance** side we found four exact levers, and they are unusually *flat*: pairing 3.65x, lattice 1.3x, fold 1.17x, radius 1.04x, with the last two overlapping the

second. Nothing we tried that used the network's *realised* structure — its Jacobian, its gate pattern, its sampled covariance — reduced variance at all, and after 32 layers we think that is the rule rather than our bad luck: the activation pattern retains essentially no usable correlation with any quantity computable in advance.

What does work is the opposite move, and we found it late: not learning structure from the sample but refusing to sample structure we can write down. Section 6 turns three exact identities — the first-layer mean, the input second moment, the whole first-layer post-ReLU Gram — into corrections worth 1.169x together, at a cost of 2% of the sample count. These are not variance reduction in the estimator-theory sense; they are the deletion of estimation error for quantities that never needed estimating. That direction is not exhausted. It is bounded only by how far up the network a closed form survives, and we can show it does not survive past layer one (6.4).

The point-set direction looks similarly closed. Three independent constructions — a randomly-shifted Korobov lattice, a CBC-optimised lattice, and a scrambled Sobol' net — land within noise of each other at 1.3–1.5x, none achieves a rate better than `N^-1` in 256 dimensions at these sample counts, and the one construction with a genuinely different invariant (mutual orthogonality, 6.9) is *worse*. When four unrelated methods agree on the same number, the number is a property of the integrand.

One more thing the cost side taught us, which we did not expect, and which took us three wrong answers to get right. The meter charges two currencies at two rates. Work inside a tracked operation is billed at its nominal FLOP count no matter how long it takes: our 2.66e12 FLOPs occupy about 479 s of backend time, or 5.5e9 FLOP per second, and none of that second is charged. Work outside one is billed at `gamma = 1.06e11` FLOP per second of wall. The exchange rate is about **19 to 1**, and it is not neutral with respect to algorithm design.

We first read the untracked term as an overhead per operation and built two probe estimators with 200 and 800 extra no-op calls to measure it. They did not need to be run. flopscope reports four times that satisfy exactly

```
wall_time_s = flopscope_backend_time_s + flopscope_overhead_time_s
            + residual_wall_time_s
```

so the meter already subtracts its own bookkeeping. We then profiled the deployed `predict` stage by stage and got a confident, specific, wrong answer: 40% of the residual in `stats.norm.ppf`. The first call to it costs 35.1 ms and every call after costs 0.06 ms — a factor of 550 — and a once-through profile measures warm-up, not cost.

What is actually charged, once both mistakes are removed, is one untracked residue **per array call** — allocation, first touch, the wrapper — with almost no dependence on what is in the call. The cleanest measurement of it is a difference between two of our own submissions. `est_v9` differs from `est_v8` by **+38 array calls and −4.9e7 FLOPs**, and the grader charged it **+15.87 ms per MLP**: about 0.4 ms, or 4e7 FLOP-equivalent, per call. Against `est_v8`'s own 168 calls and 16.5 ms the figure is 0.098 ms. Either way:

> the free tier is worth on the order of a thousand array operations, before a single FLOP is spent inside any of them.

That reprices everything. A 256x256 Cholesky is 5.6e6 nominal FLOPs and a matrix inverse 3.4e7 — 0.002% and 0.012% of a 2.7e10 budget, and irrelevant. What they cost is a call each. The first-layer covariance match in section 6.3 is exact, measures 1.048x at three standard errors, and costs 2.1% in FLOPs; what kills it is that it is 23% more calls.

It also turns out to be a lever rather than only a tax, which is the one place this section is wrong about its own subject. We had written the residual off as irreducible — section 7 records the failed attempt to cache it away — because we were looking at Python loops, which are worth 0.004% of the budget. Counting calls instead, `est_v8` spends 33 of its 168 on `astype`, one per weight matrix, where `fnp.stack(mlp.weights).astype(fnp.float32)` costs two and indexing the result costs nothing at all. With `solve` in place of `inv` and a transpose, the lattice and the jitter matrix built once per split rather than once per MLP, and `ReLU(-z) = ReLU(z) - z` restoring a halved first layer that an earlier version had and we lost in a merge, the same estimator makes **128 calls instead of 168 and 1.6% fewer FLOPs**, with the point count deliberately held at `npair = 3089` so that the lattice, the shift and every draw stay bit-identical and the whole difference has to appear in the utilisation.

We note the obvious corollary and did not act on it: since untracked work is billed by the clock, a 256x256 factorisation done in raw NumPy would be charged roughly six times less than the same factorisation done through the meter. That is the class of manoeuvre this challenge has spent the week repricing, and the right response to noticing it is to report it, not to use it. But it does mean the rule currently penalises exact linear-algebra corrections relative to sampling, which seems like the opposite of what a white-box estimation benchmark would want to encourage.

That leaves the biased route, and it is the one place where our measurements say there is real room. Section 7.11 sets the bar — a deterministic predictor can only enter through a blend, the blend is a harmonic mean, so the predictor must match the sampler's own accuracy before it doubles the score, which is 25x beyond the bundled baseline. Section 7.10 then says where those 25x are: not in the covariance approximation (worth nothing) and not in the Gaussian assumption (worth 8e-7 of 7.07e-5), but in the drift of the propagated moments across 32 layers. A recursion that tracked the final pre-activation's mean and variance faithfully would clear the bar by a factor of four and take the score to 5.1x below what we submit here.

We are not claiming that recursion exists. We are claiming that the error budget has been mislabelled — the thing that looks like an approximation wall is a 1.2% contribution, and the thing that looks like bookkeeping is 98.8% of the error. A cheap learned correction, measured against the *unimproved* baseline, buys 1.12x before its own cost and less than 1x after; measured against a faithful one it would have a very different job to do.

9. The score's own error bar, and four conclusions we withdrew

Everything above compares numbers. Late in the work we stopped to ask how big a difference the comparison can actually see, and the answer changed our reading of four of our own results — including the one the estimator was built on and the lever it was built from.

The set score is tail-dominated. The mini-100 score is a mean of 100 per-MLP MSEs that span a factor of 42 — 5.83e-7 on the easiest network, 2.42e-5 on the hardest. A mean of a hundred numbers with that spread is not a precise quantity.

The grader is deterministic, which hides this. Our estimator draws its Cranley–Patterson shift from `default_rng(mlp.seed)`, and `mlp_seed` is fixed per network, so re-running the identical estimator reproduces its score exactly. There is no visible run-to-run scatter to warn you. But any change to the point set — a different point count, a different generating vector, a different transform — re-rolls all 100 draws, and that is precisely the situation in an A/B test.

Measured. Same 100 networks, same estimator, 20 independent shift draws:

mean of set means	3.838e-6
sd of set means	3.115e-7 = **8.1%**
observed range	3.203e-6 – 4.552e-6 (**1.42x**)
two runs of the *same* estimator, 95%	differ by up to **23%**

A bootstrap over *which* networks are in the suite — the right error bar for a public-versus-private comparison rather than for an A/B — gives a similar **9.0%**.

What we withdraw. Four conclusions were reached on differences inside this bar, or on no difference at all:

variant	change	grader	vs 3.33e-7	then	now
A	CBC generating vector (section 5)	3.85e-7	+16%	"the vector buys nothing"	not resolvable — but re-measured paired, and true
B	first layer as one matmul, N 3167→3217	3.62e-7	+8.7%	"worse, discard"	not resolvable
C	the budget placement itself (section 1)	—	—	"`alpha = 0.89`, sit on the floor"	**wrong**, `alpha = 1.01 +- 0.03`
D	antithetic pairing (sections 0, 3)	—	—	"3.65x over i.i.d."	**1.007x** at equal cost

C is not a discarded variant but a discarded *premise*, and it is the one that mattered. The 8% bar cuts both ways: it is why B's 8.7% penalty was not evidence of harm, and it is also why three local runs could not tell `alpha = 0.89` from `alpha = 1.05` — a distinction that decided where the estimator would live.

The table that finally settled it (section 1.1) carries the same per-row noise; what makes it decisive is the lever arm. Its raw MSE spans a factor of ten, so a 5% error on each row perturbs a slope fitted across `log 10` by about 0.02 in the exponent, where the three local runs spanned a factor of five with the same error and were read one adjacent pair at a time. The fix for a noisy measurement was not a better measurement; it was a wider range.

D is the same failure one level down, and it went unnoticed longer because it never looked like a comparison. The 3.65x was read off a version-to-version step in which the antithetic pair arrived together with the float32 cast, and it was never re-measured against a control at matched forward passes — the arithmetic that halves the number of independent points when you pair them was simply not done. Twenty-four MLPs by six shifts say the pairing is worth 1.007x alone and 0.847x once a lattice is present. Everything in sections 3 and 6 that reasons from "layer 1 is exact" is reasoning about a device

with no measurable effect on the only layer that is scored, and section 8 is the ceiling argument we built on top of it.

Variant B is the instructive one. Its change is an algebraic identity — `ReLU(-z) = ReLU(z) - z` lets the antithetic pair share a matmul, exactly, in float32 — and the freed 1.55% of the budget strictly increases the sample count. It cannot be worse in expectation. It measured 8.7% worse, because changing `N` changes the lattice and therefore re-rolls every shift.

What this implies for method. Three things, and we would have saved a week by seeing them earlier.

1. *Prefer changes that are provably beneficial to changes that need measuring.* At an 8% error bar, a 2% improvement is unverifiable in the grader; it is still an improvement if you can argue it in closed form. Conversely, a 16% grader gap is not evidence.
2. *Compare offline, paired.* A replica of the estimator in plain NumPy runs the 100 networks in 90 seconds against hours in the queue, and comparing variants *paired per network* cancels the difficulty spread that causes most of the noise. Two details make the replica faithful to within 4%: use the exact `E||x||` series (our first replica used one accurate to 4.9e-4, and because the network is homogeneous of degree one that alone moved every score by 5%), and read the shift from the dataset's `mlp_seed` so the replica draws what the grader draws.
3. *Do not accept the default Phase 2 designation.* Rule §6 uses the best-scoring public submission if none is designated. Taking the maximum over many noisy draws is the operation that maximises draw luck, and the private re-run does not repeat that luck.

Where this leaves our own score. `arianvassili` filed a floor for unbiased estimators of roughly 3.7e-7 adjusted, from a per-pair variance of about 0.012 (#18105). Our replica puts our per-pair variance at 0.0109 and our score on the same 100 networks at 3.46e-7 — slightly under the filed floor, and about 4% above the 3.33e-7 the grader returned. We read that as: this estimator is sitting *at* the floor, and the differences we spent the second half of this work chasing were never resolvable. That is consistent with `Kerensa`'s independent report in the same thread of a similar floor with at most 10% of remaining slack.

Our best submission at the time this section was written was 2.877e-7, under that floor, and the honest reading was that this does not contradict it. At `alpha = 1` the score is `sigma^2 c / B` at every point count — algebraically identical on the floor and off it — so the four submissions in section 1.1 are four draws of one quantity, and their spread from 2.88e-7 to 3.42e-7 is 19%, which is the 8% bar doing what an 8% bar does across four draws. We report 2.877e-7 because it is what the grader returned for the submission we designated, not because we can show it is better than 3.09e-7. (Our current best is 2.0925e-07 — the floor argument survives it, because the floor is a statement about per-pair variance and `v16` reduced that variance rather than out-sampling it; but the `alpha = 1` step in this paragraph does not, see section 1.2.) The floor argument stands; we are sitting on it, from slightly below, by luck of the draw. Section 8 is what we have instead of a way past it.

The organisers confirmed the algebra on 7 August, in reply to our own filing of it in the same thread: *"We reproduced your plateau arithmetic and it holds: for pure Monte Carlo, `mse = V/n` and `C = c*n` cancel, so the score sits at `V*c/B` regardless of `n`."* They then drew the boundary we had not: it bounds Monte Carlo estimators and says nothing about analytic ones, which have no `n` to cancel, or about control-variate hybrids, whose cost `c_a + c_mc*n` breaks the cancellation — so a score below the plateau is not by itself evidence of misaccounting. That correction is right, and everything in sections 13 and 14

is this write-up taking it seriously: the plateau is a statement about our family, not about the problem, and the whole of section 14 is an attempt to find out what the other family is doing. In the same reply the organisers stated that *"submissions that intentionally route computation around flopscope's accounting to obtain unmetered compute … will be disqualified based on final review by ARC"*, which is the disposition of thread 18132 and of the `F/C < 0.05` entries that section 10.12 measures.

10. Where the variance actually is

Sections 5 through 8 chased the cost side: better point sets, cheaper matmuls, a ceiling argument. This section is the other half, and it reverses three conclusions above.

It also adds eleven entries to section 7's twenty-one, which is where the title's count comes from: deterministic full-covariance propagation (10.5), the mean-anchored and covariance-anchored hybrids built on it (10.5), the independence surrogate (10.5b), pre-activation moments read off the sample and the row split meant to decorrelate them (10.5b), a deliberately biased forward pass and the same-row multilevel coupling that was supposed to debias it (10.6), a half-width nonlinear subnetwork and a column-pruned copy of the fine network as control variates (10.7), and a lattice of 2p points that turned out to be its own antithetic mirror (10.8).

10.1 The cost side is closed

Section 8 asked how much cost was still on the table and answered it by extrapolation. We measured it instead. For eight chal8 MLPs and 32,768 spherical samples, the fraction of activation *entries* that are nonzero:

layer	0	8	16	24	31
entry nonzero	0.500	0.483	0.495	0.515	0.495
live outputs	1.000	0.962	0.889	0.821	0.738

Exactly half the entries are zero, at every depth, and the depth profile is flat. Any scheme that still forms every needed product therefore bills at least **0.4977** of a dense forward. That is a floor, not an estimate.

Against it, what the shipped bucketed prefix scheme reaches:

buckets	prefix bill	x live outputs
1	0.8880	0.7886
16	0.7213	**0.6406**
64	0.7133	0.6334
4096	0.7107	0.6312

Sixteen buckets already gets 97% of what infinitely many would, so the bucket count is not a lever, and the whole prefix construction saturates at 0.71 against an entry floor of 0.50. The remaining 0.64 → 0.50 is per-entry sparsity, which a dense matmul cannot express and block sparsity cannot reach either: at 50% density a 16-wide column block is all-zero for a given row with probability 2^-16, so there are no skippable blocks. **There is 1.29x of cost left in total and no mechanism for it.**

Feeding the 0.4977 floor back through the leading honest participant's telemetry (flops 1.301e11, raw MSE 1.717e-7, read through the endpoint of section 10.1) bounds their cost advantage over our 0.641 at 1.50x, which forces at least **2.05x** of the remaining gap to be variance per sample. Section 8's estimate that the leader ran at 0.317, and section 7.3's at 0.24, are both below the floor and therefore impossible; those two numbers are withdrawn.

10.2 What our variance reduction is actually worth

Every ingredient, measured head to head at a matched *row* count so the ratios are variance ratios at matched cost rather than at matched sample count. Eight MLPs, forty repetitions, 16,384 rows per arm, reference 4e6:

arm	final MSE	gain over the row above
raw N(0, I)	2.7592e-06	—
+ sphere normalisation	2.3990e-06	1.15x
+ whitening (layer-1 input covariance)	1.7794e-06	1.35x
+ exact layer-1 mean	1.2745e-06	1.40x
+ antithetic	1.0057e-06	1.27x

Two things to take from this. The antithetic pairing earns its keep, but not for the reason section 3 gives: the correlation between a row and its partner *at the scored layer* is only **-0.049**, which on its own would be worth 1.05x. The 1.27x comes from layer 1, where the pair mean is exactly |z|/2 and the sign variance is removed outright; almost none of it survives to depth 32 as correlation.

More importantly, the two largest factors — 1.35x and 1.40x, together 1.9x — are the *same idea applied once*: forcing the sample's layer-1 moments onto their exact values. We do it at layer 1 because layer 1 is the only place the exact moments are free.

10.3 The oracle says that idea is worth 100x

So we measured what forcing the mean at *every* layer would be worth, with the target taken from a 2e6-sample reference rather than computed. Pinning E[a_l] for layers 0..L-1 and never touching the scored layer:

pinned	final MSE	gain
nothing	2.3327e-06	1.00
layer 0	1.4516e-06	1.61
layers 0-1	1.2270e-06	1.90
layers 0-3	9.8535e-07	2.37
layers 0-7	7.1762e-07	3.25
layers 0-15	3.9808e-07	5.86
every layer	2.2343e-08	104.4
mean and per-neuron std, every layer	1.2279e-08	190.0

The deep end of that table is not a lever — pinning layer 30's mean nearly determines the answer, so it is measuring how much of the answer the oracle was handed. The shallow end is not: layers 0..7 are far enough from the output that nothing about the final value has leaked, and pinning them is worth **3.25x against the 1.61x we already have, a factor of 2.02** — which is, to within the precision of the argument, the entire gap identified in 10.1.

The structural fact behind it: almost none of our final error is generated at the scored layer. It is inherited from the sampling error of the shallow means and compounded on the way down.

(A note on a wrong turn. We first read the final-layer coefficient of variation — a mean over neurons of std/mean, which runs 10.9 to 24.2 — as evidence of a heavy tail, and built a story about rare amplified directions. That statistic is dominated by near-dead neurons with tiny means. In aggregate, sqrt(mean var)/mean|a| is **0.41**, against 1.46 for a single relu(gaussian): depth *reduces* relative dispersion. The heavy-tail story was wrong and is withdrawn; the inheritance structure above was measured independently and stands.)

10.4 Why the obvious realisation is a wash, and what is not

The oracle's target cannot be sampled into existence. Estimating E[a_k] from a wider run costs in exact proportion to the variance it removes — spend f of the budget on a shallow pass over N rows and the correction is worth what f buys, no more. Reallocating rows across depth (many rows through the first layers, fewer through the rest, each layer's mean pinned to the layer above it) is not quite a wash because the sensitivity is very unequal, but with layer 0 already exact it is worth only about 1.16x.

The only route that is not proportional is a target that costs O(1) per MLP. So we propagate the moments analytically. z = x W0 on the sphere is exactly elliptical, so E[a_0] and Cov(a_0) are the order-1 arc-cosine kernel with no approximation. From there each u = a W_l is a sum of 256 correlated terms and is treated as Gaussian, which makes both rectified moments closed form:

```
E[u_i^+]       = s_i ( alpha_i Phi(alpha_i) + phi(alpha_i) )
E[u_i^+ u_j^+] = s_i s_j F(alpha_i, alpha_j, rho_ij)

F(a,b,r) = (ab + r) Phi2(a,b;r)
         + a phi(b) Phi((a - rb)/sqrt(1-r^2))
         + b phi(a) Phi((b - ra)/sqrt(1-r^2))
         + (1 - r^2) phi2(a,b;r)
```

with alpha = mu/s and s = sqrt(diag). We could not find this last identity stated for general means, so: it reduces to the arc-cosine kernel at a = b = 0, to (1+a^2)Phi(a) + a phi(a) at rho = 1 and a = b, and agrees with 4e6 Monte Carlo draws at ten (a, b, rho) including rho = ±0.99. Phi2 is evaluated as Phi(a)Phi(b) + int_0^rho phi2(a,b;t) dt by 16-node Gauss-Legendre, maximum error 2.9e-5 against a 64-node reference. The whole propagation is 2 matmuls and one 256x256 elementwise pass per layer, about 1.2e8 FLOPs, so twelve layers cost 0.5% of B — and because the score is flat in n (section 1), a *fixed* cost F only multiplies the score by 1/(1 - F/K). Half a percent.

10.5 How far the analytic target is better than the sample

It is biased, so it is not used raw, and the size of the bias depends entirely on how the question is asked. Run the propagation as a *free-standing chain* from layer 0 and it accumulates 31 steps of CLT error. But

a chain is not how an estimator would use it: the sample is right there, so every layer can restart from sample-corrected moments and commit only **one** step of error before being re-anchored. Both, against the sampling error of the 75,886 rows the budget affords:

layer	chain	one-step	sampling	one-step / sampling
0	4.63e-07	4.63e-07	8.96e-06	0.05
1	1.56e-06	1.49e-06	6.63e-06	0.22
2	4.07e-06	2.76e-06	5.14e-06	0.54
3	7.63e-06	3.27e-06	4.06e-06	0.81
4	1.15e-05	3.99e-06	3.38e-06	1.18
8	2.82e-05	3.51e-06	1.82e-06	1.93
15	4.11e-05	2.54e-06	1.02e-06	2.49
31	4.17e-05	8.08e-07	4.82e-07	1.68

The two columns are 16x apart at depth and they say opposite things. The chain diverges — by layer 8 it is 15x worse than simply averaging, and there is no reason to use it past layer 3. The one-step error does not: it rises to about 4e-06 by layer 4, saturates, and then *falls* with depth, ending at 8.1e-07. It never exceeds 2.5x the sampling error anywhere in the network.

That difference decides the design. Under the chain reading the analytic target is a shallow-layer trick worth using for three layers; under the one-step reading it is worth blending in at **every** layer, because the variance-optimal mix

```
a_hat_l = a_bar_l + beta_l ( mu_l - a_bar_l ),   beta_l = 1 / (1 + n r_l)
```

with r_l the analytic bias in units of one-sample variance is never worse than either end — beta -> 1 recovers the exact target, beta -> 0 the plain sample mean — and its residual is the harmonic mean of the two. Averaged over the 32 layers the one-step blend removes **36.9%** of the per-layer mean error, against the 5% the chain-based blend can reach past layer 3.

The estimator sits between the two arms and not by accident. Re-anchoring the *mean* each layer is free: the sample mean of the activations is the thing being estimated, it is already computed. Re-anchoring the *covariance* is not — it needs a W x W x n matmul per layer, 3.7% of B each, unaffordable past one or two layers. So which moment's drift actually causes the chain's divergence is the question that sets the usable table, and it is measured separately.

And the whole bivariate apparatus is unaffordable anyway, for the reason that already killed v17. We built the propagation above, priced its *flops* at 0.5% of B, shipped it in an estimator, and only then timed it: twelve layers take **2.86 seconds** of wall clock per MLP. At lambda = 1e11 one second is 36.8% of B, so the twelve-layer chain alone is 105% of the budget and the full thirty-two is 280%. The flop bill was never the binding constraint and we priced the wrong resource twice in a row — 6.3 and then this. The mechanism is the same both times: a small matrix decomposition or a 256x256 quadrature is nothing in flops and everything on a clock that is billed at 1e11 per second.

That is fatal to the construction as written, and it points at the construction that replaces it. The *mean* update

```
mu_n_i = s_i ( alpha_i Phi(alpha_i) + phi(alpha_i) )
```

touches the covariance only through its diagonal s_i^2 = Var(u_i). Drop the off-diagonal — treat the hidden units as uncorrelated — and

```
Var(u_i) = sum_k W_ki^2 Var(a_k)
```

is one matvec against the elementwise square of W. That is 2 W^2 flops per layer instead of 2 W^3 plus a W x W quadrature, and, measured the same way, **3.2 ms for all thirty-two layers against 7,627 ms**: 0.12% of B in wall time, 0.002% in flops, a factor of 2,350.

The correlations turn out to be the substance. With both moments re-anchored on the sample at every layer, so that nothing accumulates and the independence assumption is the only error:

layer	independent	full covariance	sampling
1	1.66e-04	1.49e-06	6.63e-06
4	2.09e-04	3.99e-06	3.38e-06
8	1.96e-04	3.51e-06	1.82e-06
31	3.37e-04	8.08e-07	4.82e-07

The damage is done at the first step and never recovers: 119x worse than the full propagation averaged over layers 4-31, and 698x worse than plain averaging at the scored layer, which drives the optimal blend weight to 0.001. Neurons in a hidden layer share their inputs, so their activations are strongly correlated and Var(sum_k W_ki a_k) is not close to sum_k W_ki^2 Var(a_k). The idea is worth 0.037 of the per-layer mean error against the full propagation's 0.369, and almost all of that 0.037 is layer 0, which we compute exactly anyway.

So the two ends of the analytic route are accurate-and-unaffordable and free-and-useless, with nothing between them. Which would close the line, except that the whole apparatus was answering a question we did not have to ask — 10.5b.

10.5b The covariance was never needed

Everything above propagates Sigma in order to predict two numbers per neuron: the mean and standard deviation of the *pre-activation* u = a W, which is all the rectified formula

```
E[relu(u_i)] = s_i ( alpha_i Phi(alpha_i) + phi(alpha_i) ),  alpha_i = m_i/s_i
```

actually consumes. But u is not a quantity that has to be predicted. It is the array the ReLU is applied to — it exists in the forward pass, on the rows we already paid for. Both moments can be measured rather than modelled:

```
m_i = mean_n u_i,    s_i^2 = mean_n u_i^2 - m_i^2
```

for one extra elementwise square and reduction, 2 n W flops per layer, 0.46% of B for all thirty-two, and no matrix product, no quadrature, no covariance and no propagation error anywhere. What survives is

only the Gaussianity of u_i itself, which is the irreducible part of the surrogate and exactly what the one-step column above measures.

This is not a cheaper approximation of the analytic target. It is the same target with the estimation of its inputs done by measurement instead of by a model, and the model was the only expensive part. It also changes what the target *is*: no longer a prediction to be blended against the sample, but a second reading of the same rows — the sample mean of a kinked function, against a smooth functional of two well-estimated averages of the same data.

And that last clause is why it fails. Eight MLPs, sixteen repetitions of 75,886 rows, both estimates formed on the *same* rows so the comparison is paired:

layer	empirical mean	gaussian fit	optimal weight	blend / empirical
4	3.364e-06	7.203e-06	0.027	0.999
8	1.791e-06	5.230e-06	0.009	1.000
16	9.270e-07	3.459e-06	0.001	1.000
31	4.678e-07	1.277e-06	0.003	**1.000**

The Gaussian fit loses at all thirty-two layers and the blend is worth nothing anywhere. We had mis-stated the blend's own precondition. The weight beta = 1/(1 + n r) is derived for a target that is *deterministic given the weights* — independent of the sample, so that the two errors are uncorrelated and the mix is a genuine averaging of independent information. Read the moments off the same rows and the target becomes a function of the very noise it is supposed to correct; the errors are strongly positively correlated, and the variance-optimal weight on the worse of two correlated estimators collapses. Splitting the rows to decorrelate them does not recover it either: the fit's bias alone, 8.1e-07 with *exact* moments, already exceeds the empirical error at full row count, so every split from f = 0.1 to f = 0.5 loses more on the empirical arm than it gains on the fit.

So the three routes are one scissors with no gap in it. A deterministic target is accurate and costs 280% of B on the clock; dropping the correlations makes it free and 119x less accurate; measuring the moments makes it free and exact and correlated with what it would correct. The analytic-target line is closed.

One correction we had to make to ourselves on the blend weight. r_l is measured against *plain spherical* sampling, and our sample is not plain — 10.2 says it is 2.32x better at matched rows — so the sample beats the table and beta must be scaled down by that factor. Getting this wrong is not symmetric: with the chain table at layer 8, the unscaled weight gives beta = 0.061 where the optimum against our actual sample is 0.027, and the mis-set weight makes the blend 1.4% *worse* than not correcting at all. Under-shooting beta costs some of the gain; over-shooting buys the bias.

10.6 The one mechanism that could go under the cost floor, and why it does not

10.1 gives 0.4977 as a floor for any scheme that still forms every needed product. Nothing in the rules asks for an unbiased estimator, though, and the score is a mean squared error — so a *biased* forward pass that skips small-but-nonzero work is the one construction that could go under it. Sort each layer's columns by batch mass, keep the leading fraction q, and cost falls to about q of dense while the affordable row count rises like 1/q. The trade is bias(q)^2 against V q / n.

It is not close. Eight MLPs, bias measured on 131,072 rows with the estimator's own sampling variance divided out, truth at 4e6:

q	bill/dense	rows	bias^2	variance	total MSE	vs exact
1.00	0.6410	75,886	5.29e-08	4.82e-07	5.35e-07	1.000
0.95	0.6089	79,880	7.17e-04	4.58e-07	7.17e-04	0.001
0.90	0.5769	84,318	1.79e-03	4.34e-07	1.79e-03	0.000
0.70	0.4487	108,409	2.27e-02	3.38e-07	2.27e-02	0.000
0.30	0.1923	252,953	4.38e-01	1.45e-07	4.38e-01	0.000

Dropping the lowest-mass **five percent** of columns — for a 5% saving — moves the answer by 1340x the entire error budget. There is no crossover to find; the bias is already three orders of magnitude past the variance at the first step off exactness. The 0.4977 floor stands, and with it 10.1's conclusion that at most 1.29x of cost remains.

The same two passes answer the unbiased version for free. E[f] = E[f_q] + E[f - f_q] needs no bias at all, and the difference is taken on the *same* rows, so the only question is whether the pair is coupled tightly enough to be worth splitting the budget over. With per-row costs q and 1+q, the optimal split gives (sqrt(Var[f_q] q) + sqrt(Var[f-f_q] (1+q)))^2 / T against V / T:

q	Var[f_q]/V	Var[f-f_q]/V	net gain
0.95	0.970	**0.362**	0.309
0.90	0.931	0.564	0.263
0.80	0.828	0.805	0.246
0.60	0.500	1.017	0.301
0.30	0.027	0.973	0.679

A network that is 95% identical to the fine one, evaluated on the *same input row*, already differs by 36% of the full variance, and by q = 0.6 the difference carries more variance than the thing being estimated. Every entry is a loss.

10.7 The four failures are one failure

That number is worth staying with, because it explains the shape of section 7.

- 7.1, a collapsed linear surrogate: rho = 0.08.
- 7.6, a low-rank multilevel estimator: Var[f - f0]/Var[f] = 0.98 at rank 8, 1.00 at rank 32, and still 0.67 at rank 128.
- cvlevel, a narrow nonlinear subnetwork as a control variate: rho = 0.29 at *half* the full width.
- 10.6, the fine network with 5% of its columns removed: 0.36 of the variance already lost.

Four different coarse models, spanning from a linear map to a 95%-complete copy of the network itself, and none of them tracks the fine model sample by sample. The composition of 32 random ReLU layers is pathwise chaotic: any perturbation of the map, however small, decorrelates its output on a fixed

input. No control variate, no multilevel scheme, and no biased shortcut can work here, and the reason is a property of the object rather than a failure of construction.

What survives is exactly what does not need pathwise agreement. Sphere normalisation, whitening, the exact layer-1 mean, antithetic pairing and the analytic moment blend are all statements about the *distribution* of the input set or of an intermediate layer, and the distributional map is smooth even where the pathwise map is not — which is why 10.5's one-step analytic error saturates at 4e-06 and then *falls* with depth, while every sample-level surrogate diverges over the same layers. That is the organising fact of this entire write-up, and we arrived at it by exhausting the other side.

10.8 The lever that does not need a better mean

Everything above tried to make a shallow mean *more accurate*. One thing is left that does not: making it more accurate only where accuracy is worth something. 10.3's table, differenced, decomposes the final error by the layer whose sampling noise produced it. With layer 0 already exact, of the error that remains:

layer 1	2-3	4-7	8-15	16-31
0.153	0.167	0.184	0.221	0.259

Per layer that is a tenfold spread in sensitivity — 0.153 at layer 1 against 0.016 each at the bottom — and every layer is currently run on the same number of rows. Run layer l on n_l rows and pin its mean before dropping to the next count, using the carry the exact layer-0 mean already uses, and

```
MSE = sum_l s_l / n_l   subject to   sum_l n_l = T
```

is minimised at n_l proportional to sqrt(s_l). The model says 1.13x. Measured at matched total row-layers, eight MLPs and twenty-four repetitions, it is more:

schedule	final MSE	measured	model
flat	1.5185e-06	1.000	1.000
one halving	1.2370e-06	**1.228**	1.097
two halvings	1.2669e-06	1.199	1.099
three halvings	1.3056e-06	1.163	1.088

The model under-predicts because it treats the layers as contributing additively. They do not: a shallow mean measured on more rows does not only remove its own noise, it improves the input every layer below it sees. And one halving is the whole lever — further steps spend rows on layers that were not the constraint.

Which halving is *available* is decided by the point set rather than by the schedule, and this is where it becomes less than it looks. The rows are a randomly shifted rank-1 lattice with a prime number of points, and a subset of a lattice is a lattice only when the index set is a subgroup; Z_p has none. Every obvious halving — the first half, every other point, a random half — destroys the equidistribution section 5 measures at 1.73x.

There were always two rows per lattice point, though. Antithetic pairing runs x and -x together, so 2n rows are n lattice points times a sign, and dropping the *partner* instead of the point leaves exactly the original lattice. That halving is free of structural cost — and it gives back only 1.106x, at a cut between layers 8 and 12, falling to 1.000x by layer 18. The gap to 1.228x is the antithetic pairing still being earned below the cut, which is more than the -0.049 correlation at the scored layer led us to expect: 3 above says the gain is banked at layer 1, and that is where most of it is banked, but not all.

There is one way to have both, and it turns on an accident of the modulus. Take the lattice size to be 2p rather than p. The even-indexed points

```
{ (2j z mod 2p) / 2p } = { (j z mod p) / p }
```

are a rank-1 lattice of p points with the same generating vector — the one subgroup Z_2p has that Z_p does not — so dropping the odd points *with their partners intact* halves the rows and keeps both structures.

We wrote that version and it is wrong, in a way worth reporting because the argument above is correct as far as it goes and still gives the wrong estimator. The Korobov multiplier is `a = 17`. Every component of the generating vector is therefore odd, and an odd product modulo an even modulus stays odd, so

```
(j + p) g_i  =  j g_i + p g_i  ≡  j g_i + p   (mod 2p)
```

which puts point `j + p` at exactly `(1/2, ..., 1/2)` from point `j` in every coordinate. The baker transform sends `u -> 1 - u` under a half shift — for `v < 1/2`, `tent(v) = 2v` and `tent(v + 1/2) = 1 - 2v`, and symmetrically above, so the identity holds whatever the random shift is — and `Phi^-1(1 - t) = -Phi^-1(t)`. Sphere normalisation and the whitening solve are both odd. So a `2p` lattice is *already* closed under negation: it is `p` points and their mirrors, and applying antithetic pairing to it duplicates every row. Checked directly at `p = 13, 31, 101`: 26 distinct rows out of 52. The version would have run at half the effective sample size and we would have read the loss as the lattice-quality risk we had just written down.

The failure names the fix. Self-pairing holds exactly when the half-shift vector lies in the lattice, which is to say when the modulus is even; a halving that preserves it needs the modulus *and half the modulus* to be even:

```
4q  ->  even subset is a lattice mod 2q  ->  still even  ->  still self-paired
```

with `q` prime the least composite modulus that does it. Verified at the same three sizes: the `4q` set and its even half are both closed under negation, and the `2q` half is not. It also makes the antithetic concatenation load-bearing instead of redundant — materialise the first `2q` points, and rows `2q..4q-1` are what `relu(-z) = relu(z) - z` already produces, so a `4q`-point lattice costs one `2q`-row matmul, exactly what is paid now.

What is genuinely risked is the thing the broken version would have masked: a Korobov vector with modulus `4q` is not the object the prime-modulus theory is about, and whether `a = 17` survives the change is not arguable, only measurable.

10.9 Two levers that section 7 rejected, re-examined

Dead-neuron pruning (7.3) was rejected on a cost model that assumed the pilot had to be large. A 512-row pilot is enough — 13.6% of neurons never fire — and v16 ships it. But the classification does not sharpen into a *usable* three-way split: with a hold-out of 65,536, only **7.0%** of neurons are always-on and 79.4% keep their kink, so collapsing the always-on runs into a single linear map would still bill 0.686 of dense against the 0.641 the prefix scheme already reaches. The linear-collapse mechanism is real and is a loss here.

The whole first-layer second moment (6.3) was dropped because its two Cholesky factorisations were billed as residual wall time and the budget was tight. We argued that had become obsolete at higher utilisation and built v17 on that argument. It had not: run_v17 fails **all eight** MLPs on the combined budget at utilisation 1.2745, with residual wall time of 1.164 s per MLP against v16's 0.579. At lambda = 1e11 that extra 0.585 s is 21% of B by itself, for a 4.8% accuracy gain. Section 6.3's original conclusion was right and its stated reason was right; our reversal of it was wrong, and the correction is a factor of four larger than the effect it was chasing.

10.10 The target is exact, we are unbiased, and the front of the field is thirty-two times our budget away

Everything above treats the graded error as an honest measurement of our distance from the truth. Two things could make that false, and both would have been invisible in any experiment we ran, because both put a floor under the error that looks exactly like a hard problem.

The first is that the target might not be exact. If the graded per-layer means were themselves a very large but finite sample, every submission would be scored against a noisy reference and the numbers at the front of the field would be partly an artefact of correlation with the grader's own draw. The second is that *we* might be biased. The estimator projects the input onto the sphere of radius `E||x||`, which is not the Gaussian the task specifies. The norm concentrates hard at width 256, so the difference is small — but a bias does not shrink with `n`, and a small one would eventually become the whole error.

Both have the same signature and it is easy to read. Our error against the target should equal our own variance across randomisations, exactly, if the target is exact and we are unbiased; a noisy target or a bias of our own adds a term that does not shrink, so the ratio of the two must grow as `n` rises. Four networks, six shifts each, four sizes spanning a factor of sixty-four, antithetic pairs, no carry and no pruning so the measurement is of the law and not of the tuning:

n	mse vs target	variance across shifts	ratio	slope
48947	2.4613e-07	2.3691e-07	1.039	
195787	6.3187e-08	6.2801e-08	1.006	-0.981
783151	1.1383e-08	1.1248e-08	1.012	-1.236
3132601	3.7140e-09	3.7011e-09	1.003	-0.808

The ratio is 1.00 at every size, including the last, where the error is `3.7e-09`. The target is exact to well below the front of the field, and the sphere projection costs us nothing — it is a free variance reduction, not a bias, and it stays free down to four significant figures at an error two orders below the one we ship. That closes a question we had left open since section 4.

The same table settles what the front of the field is worth in our own units. Over the full sixty-four-fold range the slope is `-1.009`: this arm is plain Monte Carlo in `n`, and the `-1.32` we measure on the deployed estimator is bought by the pruning and the contraction, not by the point set. Reading the best raw error we can see on this challenge, `4.87e-09`, back through this table puts it between our third and fourth rows — call it two and a half million points, against the hundred and thirty thousand rows the budget holds. **The front of the field is getting, inside the budget, what we get at thirty-two times the budget.**

That number is worth stating precisely because it rules things out. It is not a tuning gap and no amount of sizing, splitting or robustification reaches it; sections 11.4 through 11.11 measure every one of those and the largest is a few percent. It is not reachable by a better constant either — a factor of thirty-two is far outside the two-fold spread we measured across two hundred and twenty lattice moduli. It has to be either an exponent — a rule whose squared error falls faster than `n^-1` — or a cost per point far below ours. Those are the only two shapes that fit, and section 15.1 narrows it further: the leader's raw error repeats to sixteen digits across submissions, so whatever it is, it is deterministic.

One more negative belongs here, because it is the last first-layer idea we had. The deployed estimator replaces the first hidden layer's mean by its closed form before pushing it into layer two. The covariance is not corrected, and section 6.4 found that correcting it against a closed form did not help. The remaining version of the idea is to correct it against a *sample* instead: layer one is one matrix product out of thirty-two, so a sample thirty-two times larger through layer one alone costs the same as one full forward pass and estimates those moments about ninety times more precisely than the working sample does. Mapping the working sample's activations onto that target with the obvious linear map gives, over eight networks and six shifts:

```
       arm      mean mse       median   vs deployed   wins
  deployed    1.7387e-07   1.2146e-07         1.000    8/8
+cov match    3.9052e-06   1.9800e-07        22.460    4/8
  no carry    1.7354e-07   1.2281e-07         0.998    3/8
```

Two readings. The covariance match wins on half the networks and loses by a factor of twenty-two on the mean, because on two networks it explodes — 76x and 26x — for a reason that is structural rather than numerical: the covariance of a rectified layer is nearly singular, most of its small eigenvalues belong to neurons that almost never fire, and the whitening map's inverse is enormous in exactly those directions. Any moment match on a rectified layer inherits this.

The second reading is the more useful one. Removing the exact-mean carry entirely changes the error by two parts in a thousand. At the deployed sample size the carry is worth *nothing*. It is free, so it stays, but it is not doing work — and taken together with the covariance result it says that the error is not a first-layer phenomenon at all. Both moments of layer one can be set to their exact values, or to values ninety times better than the sample's, and the final error does not move. Whatever is left is made in the deeper layers, where no exact moment exists to match against, which is consistent with the effective rank of about five that section 10.7 measures on the error itself.

10.11 Every method that rectifies a Gaussian at the end has a floor at 8.6e-07

Section 7.22 closed the cumulant family on a rate argument: the order-`K` error scaling has already collapsed at depth 32, each extra order buys a constant near four rather than a factor of `n`, and

extrapolating from `K = 1` and `K = 2` puts `K = 3` at `3.5e-06`, losing to our sampler. The conclusion was right. The argument was weak -- it was an extrapolation from two points, and it would not have survived an implementation of `K = 3` that behaved better than expected.

There is a much stronger argument and it needs no cumulants at all. Every method in this family ends the same way. Having carried some description of the law of the final pre-activation `z`, it evaluates the mean of the rectifier in closed form, and the closed form that exists is the Gaussian one,

```
E[relu(z_i)] = mu_i Phi(mu_i / sigma_i) + sigma_i phi(mu_i / sigma_i).
```

So there is a floor under the whole family that does not depend on the order carried, on the factorised representation, or on how the moments were obtained. It is the error you get from the *exact* `mu` and `sigma`, and it is exactly the price of `z` not being Gaussian. Measuring it needs no closure: draw a large sample, take the sample mean and standard deviation of the final pre-activation, feed them to the formula, compare against `alm`.

```
                        quantity     mean mse
    Gaussian closed form, exact mu/sigma   8.6062e-07
      + one Edgeworth (third cumulant)     3.3560e-06
            plain MC on the same draws     1.3453e-09
         our deployed submission (raw)     2.3817e-07
                  leader dpskv5 (raw)     3.6000e-09
```

The floor is 8.6e-07, and it is 236 times above the leader's raw error. It is also 3.6 times above our own sampler, which is not in the family. No order of cumulant propagation, however well implemented, reaches even our score, and the question of what `K = 3` would have cost is moot.

The Edgeworth row is worth a line. Adding the third-cumulant correction makes the answer four times *worse*, and the reason is visible in the moments: the final pre-activation has a skewness of `+-0.03` and an excess kurtosis of `+0.30` to `+0.59`. It is symmetric and heavy-tailed. The third-order correction is therefore fitting noise, and the series is asymptotic, so there is no reason for the next term to rescue it.

Because the miss is only 0.12% in relative terms and looks systematic rather than random, we tested whether it can be tabulated. Writing `E[relu(z_j)] = sigma_j h(u_j)` with `u_j = mu_j / sigma_j` defines `h` exactly, one per neuron; if `h` were the same curve for every neuron of every network it could be measured once, offline, to any precision, and the run-time task would collapse to getting two moments right. Fitted on seven networks and scored on the eighth, eight folds, 24 knots, 16.7M draws per network:

```
           model  held-out mse  vs gauss floor
        gaussian    8.6062e-07           1.000
            h(u)    4.9323e-07           1.745
       h(u) + skew    4.6857e-08          18.367
     h(u) + exkurt    4.6716e-07           1.842
   MC on the draws    1.3453e-09         639.710
```

The shape is **not** universal in `u` alone -- that is worth only 1.7x. Letting the curve move with the neuron's own skewness is worth 18x, which is a real effect and, given that the mean skewness is zero, a purely per-neuron one. But it stops at `4.7e-08`, thirteen times above the leader, and it consumes exact moments from a sixteen-million-draw sample that no run-time budget contains. Getting `mu` to the accuracy this needs is harder than estimating the answer directly: `Var(z)` exceeds `Var(relu(z))`, so the mean of the pre-activation is the *more* expensive of the two to sample.

What this leaves is a sharper statement of ignorance than we had. The leaders are not sampling -- 30 gigaflops is about seven thousand rows, at which count plain Monte Carlo sits at `5e-06` and a thousandfold variance reduction in 256 dimensions is not available from any point set. They are not closing in the Gaussian family either, because the floor above is 236 times too high. And their multipliers say what kind of thing they are running: `0.111` for the leader and `0.117` for third place, both on the cost floor. A sampler with our `1/n` slope is indifferent to where it sits on the budget axis (section 7.25); a method that *plateaus* is not, because spending more buys it nothing and costs it the multiplier. **Sitting on the floor is the signature of a deterministic method that has converged to its own bias**, and both leaders are showing it.

The most useful single number to come out of this is not about the leader at all. Third place holds a raw error of `2.14e-07` against our `2.38e-07` -- the same accuracy -- at an eighth of the cost. The gap to the front of the field is 63x and we cannot see a mechanism for it. The gap to third place is 8x, it is entirely on the cost axis, and sections 7.26 through 7.28 have now excluded the cost side of our own estimator as the place it could come from.

10.12 What the top of the board's own telemetry says

Section 10.10 read the leaders through two numbers, the score and its secondary, and guessed at the mechanism from the multiplier. That was under-reading the available evidence. Every submission's detail page serves a `per_mlp` block without authentication, and each entry carries a full cost ledger: `flops_used`, `effective_compute`, `wall_time_s`, `residual_wall_time_s`, `flopscope_backend_time_s`, and the exhaustion flags. The mechanism does not have to be guessed.

Three graded entries from the top of the board on 7 August, against ours:

submission	flops_used	F/C	wall_time_s	residual_s	raw MSE
324969 (1st)	2.43e10	0.94	45.3	0.015	3.63e-09
324936 (2nd)	9.01e10	0.44	55.5	1.15	4.36e-09
324954	1.78e11	0.57	52.3	1.40	3.56e-09
324409 (ours)	2.39e11	0.99	8.1	0.017	2.38e-07

Four things fall out of that table, and only the first was visible before.

The leader is inside the meter. `F/C = 0.94` and a residual of 15 ms are our own profile, not an unmetered one. Whatever produces `3.63e-09` is not the residual-wall channel of section 7.30, which in any case is worth 3%.

FLOP count does not predict accuracy. 2.43e10, 9.01e10 and 1.78e11 -- a factor of seven -- all land within 20% of the same raw error. For a sampler the raw error would move by a factor of seven with them. For a converged deterministic method the FLOPs would be the same across entries, not spread over a factor of seven. Neither shape fits.

Wall time does predict it. All three sit at 45 to 56 seconds against the documented 60-second cap, while we finish in 8.1 and are limited by FLOPs. A run that stops at the wall while its meter reads a tenth of the budget is a run whose meter is not measuring what it is doing. Taking 45 seconds of backend time on the seven cores the announcement gives the flopscope backend, at the throughput measured in section 7.30, prices out at roughly `1.6e6` rows -- and our own sampling curve puts `1.6e6` rows at `3.5e-09`, which is the number on the board to two figures.

Nothing we can measure closes the gap honestly. At 2.43e10 instrumented FLOPs a plain Monte-Carlo estimator scores about `1.3e-05`. The distance to `3.6e-09` is 3,600x. The whole apparatus of this document is worth 2.17x over plain Monte Carlo (7.26); the sphere is 1.8% and the lattice 5.2% of that; rotating into the network's own sensitivity basis is null (7.19); the kink structure is 2.0x at its most optimistic (7.31); the pruner is 1.315x and already at its optimum (7.28); every closed-form rectification has a floor at `8.6e-07` (10.11); the best low-rank Gaussian surrogate we can install anywhere in the stack bottoms out at `7.1e-07` (`research/collapse.py`, which recovers `8.59e-07` at full rank and so reproduces 10.11 as a special case). Public-set memorisation caps at `1.19e-07` (7.33) and seed recovery is closed (7.32).

We are not claiming to know what those entries do. We are reporting that their published cost ledgers are inconsistent with every estimator we have been able to construct or exclude, that the inconsistency is on the metering axis rather than the accuracy axis, and that four open accounting defects were on the forum while those scores were posted -- negative charges for empty contractions (18076), undercounted batched right-hand sides in `linalg.solve` (18082), history-dependent residual accounting (18092), and the bypass of 18099. The organisers have said (18132) that the private re-evaluation will examine the instrumented share together with the implementation. The measurement above is offered as an input to that, and the harness that produces it is four lines of `urllib` against a public URL, so anyone can reproduce it for any submission.

The uncomfortable version of this section is that we may simply be wrong and someone has a method we cannot imagine. That is why the exclusion list above is stated as a list of measurements rather than as a conclusion: each line is a door someone else can re-open with a better instrument.

10.13 What the meter actually charges, per operation

Section 10.12 ends by saying that the entries above us are billed at a small fraction of what the hardware can do, and leaves open whether that is an accounting gap or an operation menu we had not read. This section reads the menu. It is a price list, produced by calling each candidate under `flopscope.budget` on one physical core and recording what it bills against what it costs, and it is the thing we would have wanted on day one.

The score decomposes as `MSE * C/B`. For a `1/n` sampler with per-row cost `c`, that is `(k/n) * (c*n/B) = k*c/B`: the point count cancels exactly, which is why section 11.8 finds the budget axis flat, and what survives is the variance per row `k` and the *billed FLOPs per row* `c`. The second factor is linear, not flat. Halving what the meter charges for a row halves the score. So the question is not whether we can buy more rows — it is whether the same row can be bought cheaper.

Every route to a dense product is priced identically. Six ways to compute `X @ M` for `X` of shape `(4096, 256)` and `M` of shape `(256, 256)`, against the honest `2*N*W*W = 5.37e08`:

```
matmul  dot  einsum  einsum(optimize)  tensordot  inner  linalg.multi_dot
5.358e08 for all six, i.e. 0.998 x 2NWW, agreeing to the last digit
```

Routing the same product through a solve costs 1.081x and through `lstsq` 1.935x. There is no cheaper spelling of a contraction.

Precision buys nothing below float32. Billing follows the output dtype, as thread 18127 reports, and the schedule bottoms out at single precision:

```
float64 @ float64   1.996 x        float16 @ float16   0.998 x
float32 @ float32   0.998 x        int8/16/32          0.998 x
```

float16 executes 110x slower in wall time on this box and bills exactly what float32 bills, so the only dtype lever is the one already in section 2: do not land in float64 by accident.

The symmetry system is a real discount, and it does not apply to contractions. `flopscope` carries permutation-group metadata on arrays and prices symmetric operands by orbit count rather than cell count. Measured on a `256 x 256` symmetric matrix under the degree-2 swap group, and on a `48^3` tensor under the full degree-3 group:

```
                       symmetric      dense      discount
pointwise, degree 2       32,896     65,536        2.00x
reduction, degree 2       32,895     65,535        2.00x
pointwise, degree 3       19,600    110,592        5.64x   = cells/orbits
matmul, symmetric lhs   3.349e07   3.349e07        1.00x   <- none
```

The degree-`k` discount is exactly `k!` in the limit, which is the right price for storing a symmetric tensor by its independent entries. It is clearly deliberate, and it is a standing invitation to moment-propagation methods: their pointwise stage on a `k`-th order symmetric moment tensor is billed at `1/k!`. It is worth nothing to a sampler, whose hot path is thirty-two contractions and whose sample rows carry no symmetry to declare.

That closes the cost side for us. Our per-row bill is already reduced by the bucketed prefix contraction of section 11, which physically slices dead columns rather than masking them and measures a 0.72x cut; the remaining `c` is dense `float32` matmul, priced at par. We cannot buy the row cheaper, so the 58x to the top of the board has to come out of `k`, and section 7 is a list of thirty-four attempts on `k`.

A note on where the top of the board sits in this price list. Reading the public telemetry with the same harness as section 10.12, and taking the median per-MLP row of each submission, the instrumented share `F/C` and the billed rate against a measured single-core float32 GEMM rate of `1.028e11 FLOP/s`:

(Corrected in 14.15: the flopscope backend is allotted seven physical cores, not one, so the `% peak` column below is about seven times too high — ours is 4.6%, not 32.2%. The column is uniformly rescaled, so nothing that depends on the ordering or on `F/C` changes. We have left the original numbers in place and flagged them rather than silently restating them.)

```
                wall  backend  comms   ovh  resid   F/C     billed GF/s  % peak
ours 324409    7.96     7.23   0.08  0.72   0.02  0.9928     33.05      32.2%
LB #1 324969 45.31     45.08   0.01  0.23   0.02  0.9405      0.54       0.5%
LB #2 324936 56.59     22.35   2.38 33.07   1.20  0.4395      4.03       3.9%
LB #3 324954 53.47     21.56   2.58 30.43   1.42  0.5606      8.29       8.1%
      323861   3.13      0.00   0.00  0.04   3.09  0.0000      4.74       4.6%
```

Two readings we are confident in. First, 323861 is billed for no metered arithmetic at all — `F/C` is zero to four places, its whole cost is 3.09 s of residual wall time per network, and it still returns a mean public MSE of `5.57e-08`. That is the sub-0.1% instrumented share thread 18132 asks about, and the organisers' answer there is that residual time is not a second compute lane. Second, LB #1 spends 45.08 s inside the flopscope backend — not outside it, not in residual — and is billed `2.43e10` for it, which is 0.5% of what one core can do in that time. We priced twenty-six operations looking for one that sells a sampler's arithmetic at that rate and did not find one. We are not claiming that entry is doing anything improper; we are recording that its rate is not reproducible from the menu we measured, which makes it a question for the review the organisers have already said they will run.

The budget cliff we sized against does not exist. Thread 18129 reports that the deployed evaluator never fires `combined_budget_exhausted`, so the multiplier is `max(0.1, C/B)` uncapped above 1.0 rather than a hard wall at `B`. Our own telemetry pull confirms it independently on a different submission: 324954 has `C_m > B` on 100 of 100 networks, a mean multiplier of 1.1674, and predictions that were plainly not zeroed. We sized `est_v25` to land at `C/B = 0.885` specifically to keep a margin against that cliff. Since the axis is flat the margin cost us nothing, but it was bought against a rule that is not the one being enforced, and anyone sizing against a hard `2.72e11` should know that.

10.14 Accuracy per metered FLOP, across the top of the field

Sections 10.12 and 10.13 look at the two entries furthest ahead of us and ask how they are billed. The more useful question turned out to be the one across the whole top of the board: for each entry, how much *metered arithmetic* does it spend to reach the accuracy it reaches. That number is public — median `flops_used` per network from the submission page, and mean public MSE from the same blob — and it separates the field into three kinds of thing.

The board on 7 August, with our own row for scale (`F` is median metered FLOPs per network, `wall` and `bknd` are median wall and flopscope-backend seconds against a 60 s cap):

```
 rk  who               F/C     C/B    wall   bknd  resid         F   raw MSE
  1  dpskv5           0.940   0.095   45.3   45.1   0.02   2.43e+10  3.63e-09
  2  joe_wanza        0.429   0.772   56.6   22.3   1.20   9.01e+10  4.37e-09
  3  huang_chung_yi   0.207   0.012    1.5    0.2   0.02   6.76e+08  2.14e-07
  4  ednacob          0.931   0.506   42.7   39.3   0.09   1.28e+11  9.11e-08
  5  dstepanov        0.0001  0.561    1.6    0.0   1.53   1.5e+07   1.05e-07
  6  ely2sh           0.0000  1.135    3.1    0.0   3.09   ~0        5.57e-08
  7  thylinao         0.950   0.473   24.1   21.5   0.07   1.22e+11  1.71e-07
 11  mliston          0.893   0.113    3.5    2.3   0.03   2.74e+10  8.08e-07
 65  ours 324409      0.993   0.885    8.0    7.2   0.02   2.39e+11  2.38e-07
```

Two entries are billed for essentially no arithmetic at all. `dstepanov` has an instrumented share of 0.0001 and `ely2sh` of 0.0000 to four places; both pay for their whole run in residual wall time. This is the pattern thread 18132 asks about and the organisers have said they will examine. We take no position on it beyond recording the numbers.

Rank 3 is the interesting one, and it is not an accounting story. Its raw error, `2.140e-07`, is within 11% of ours. It reaches that on `6.76e08` metered FLOPs. Our sampler bills `2.44e06` FLOPs per row and needs `n = 8.5e-03 / 2.14e-07 = 39,700` rows to reach the same place, which is `9.7e10` FLOPs — **143x more arithmetic for the same answer.** No sampling scheme covers that factor; section 5 measures the entire quasi-Monte-Carlo family as worth 1.3x and section 7.34 measures the choice within it as worth nothing. Whatever rank 3 is, it is not a better sampler, and the write-ups on the forum (18063, 18085, 18097, 18106) contain nothing that spends `6.76e08` FLOPs and lands at `2.14e-07`.

It also tells us it has converged. It uses 1.5 s of a 60 s wall allowance and 1.2% of the FLOP budget, and it sits *below* the multiplier floor, where the first 8.3x of additional compute is free — the multiplier is pinned at 0.100 all the way from `C/B = 0` to `C/B = 0.1`, so any accuracy that more compute could buy in that range costs literally nothing in score. Nobody declines a free 8.3x unless it buys nothing. That is a bias floor near `2.1e-07`, in a method whose per-answer cost is two orders of magnitude below a sampler's.

This is the one place where the public board says something we could not have learned from our own runs, and it is the opposite of the lesson we drew in section 1. We spent Phase 1 asking how to place a sampler on the budget axis. The score is `k*c/B` (section 10.13) and we spent the whole competition on `k`, the variance per row, while the field's leverage was on `c` — and not `c` in the sense of billing tricks, which section 10.13 shows do not exist for a contraction, but `c` in the sense of *not doing the contraction at all*. Our best per-metered-FLOP figure is `2.11e-07`; rank 3's is `2.50e-09`. Section 8's ceiling argument bounds the devices we built. Nothing in this write-up bounds that.

We report it as an open problem rather than a result, because we did not solve it and the three closed-form directions we did try — Gaussian closure (7.10, 10.11), the arc-cosine Gram (6), and low-rank surrogates at every depth (7.29) — all plateau at or above `7e-07`, well short of `2.14e-07`. The gap between a converged cheap method and our own closure attempts is, on this evidence, where the remaining science in this task lives.

11. Why the estimator we shipped is the one we shipped

Sections 7 and 10 are a list of things that did not beat `v16`. This section began as the complement — eight more attempts at the cost side, and what they established about why `v16` was hard to beat — and turned into something else when four of those eight rejections were traced to a flag on our own benchmark rather than to anything in the estimator. What follows is both: the levers, and the accounting error that had been quietly deciding which of them lived.

11.1 The lattice is not a lattice

`_base_points` builds the rank-1 lattice the way the construction is written:

```
gv  = fnp.asarray(generating_vector(npair, width), dtype=fnp.float32)
n   = fnp.arange(npair, dtype=fnp.float32).reshape(npair, 1)
out = fnp.mod(n * gv, float(npair)) * (1.0 / npair)
```

At the deployed `npair = 40361` both factors reach the modulus, so the product reaches 1.6e9 against float32's 2^24 exact-integer ceiling. The mod of a rounded product is not the mod. Measured coordinate by coordinate against a float64 reference, **74.4% of the coordinates this produces for `a = 17` differ from the true lattice**, many by a full period. The 40361 points are still distinct — we checked — but they are not the point set the theory in section 4 is about.

The obvious correction is one line: compute the residue in float64 and store the `[0,1)` result as float32, where it is exact. That is `est_v22`, and on the hundred-MLP split it does not merely fail to help.

	score	final MSE	budget exhausted
`v16`, float32 residue	2.1716e-07	2.7603e-07	0 / 100
`v22`, exact residue	6.8449e-02	6.8449e-02	**18 / 100**

Repairing the arithmetic destroys the estimator. The reading we can defend is that a rank-1 Korobov lattice in 256 dimensions concentrates its points on low-dimensional hyperplanes, and the float32 rounding — which is deterministic but effectively arbitrary across coordinates — breaks that alignment. What we ship is not a lattice and not i.i.d. sampling; it is a stratified point set whose stratification we did not design and cannot currently characterise. We report this as a measurement and a hazard, not as a mechanism we understand.

The practical consequence is that `KOROBOV_A` cannot be tuned by lattice theory, because the object being tuned is not the lattice. We swept 300 multipliers against a float64 reproduction and found a 7.41x spread with `a = 17` at the 37th percentile — a result that looked like the largest lever in the whole project, and was worth nothing, because the sweep was measuring a point set the estimator never builds. Section 11.3 repeats the sweep on the point set the estimator does build, with a held-out split, and finds the spread is noise.

11.2 Every one of those failures was our own benchmark flag

The 43 failures are not wrong answers, and they are not the estimator's either. Every one carries `combined_budget_exhausted`, and `C_m = F_m + lambda * R_m` charges residual wall time at `lambda`. The baseline run of `v16` had been made with `--lambda-flops-per-second 1`; every candidate after it was run at the default `1e11`. Half a second of contention on a loaded laptop is 0.18 of `B` at the second setting and nothing at all at the first, so the two families were never comparable. Recovering `lambda` from each run's own numbers — `(effective_compute - flops_used) / residual_wall_time_s` — separates them exactly:

run	tracked FLOPs, max	exhausted at `lambda=1e11`	exhausted on FLOPs alone
`v21`, `a = 12789`	0.867	43 / 100	**0 / 100**
`v22`, exact residue	0.867	18 / 100	**0 / 100**
`v24`, `a = 12789`, frac 0.88	0.795	2 / 100	**0 / 100**
`v25`, frac 1.063	0.960	100 / 100	**0 / 100**

No run in that family ever exceeded the FLOP budget. Which of the two settings matches the grader is not a matter of opinion: `GET /api/v1/submissions/<id>` returns `score_secondary`, and for our shipped submission the ratio comes back at **0.797170** against tracked FLOPs of 0.7989 measured locally. The grader charges under 0.2% of `B` for residual wall — about 2 ms per MLP, the same number section 1.1 arrived at independently. Our local harness was pricing wall time roughly five thousand times higher than the thing it was standing in for, and four rejections were issued on that basis.

11.3 The multiplier really is noise, measured properly this time

With the false negative removed, `KOROBOV_A` had to be swept again on the deployed point set: float32 residue, random shift, baker transform, sphere normalisation, whitening solve, antithetic concatenation, `npair = 40361`, and the grader's own final-layer means as the target rather than a local Monte Carlo (the bundle ships them; they agree with the closed form at layer 0 to 3e-5). 320 multipliers coprime to the modulus, ranked on four networks and confirmed on four the ranking never saw:

	ranking split (MLPs 0-3)	held-out (MLPs 4-7, 24 shifts)
`a = 38515`	1.6485e-07 — **best of 320**	1.5365e-07 — 0.986x
`a = 32419`	1.8521e-07	1.5300e-07 — 0.990x
`a = 12789`	—	1.5243e-07 — 0.994x
`a = 17` (shipped)	2.6018e-07 — **rank 193 / 320**	**1.5145e-07 — best**

On the ranking split `a = 17` sits at the 39th percentile and the winner is 1.58x better. On networks the search never touched, `a = 17` wins outright and the entire top eight lands within 12% of it. The 1.58x was the search fitting four networks and six shifts.

This is the fourth time the multiplier axis has been measured and the third distinct way it has produced a number that was not real: first on a float64 lattice the estimator never builds, then without the whitening step, then against a deployment test that was really a benchmark flag. The honest conclusion is that `a` is not a lever at all — the float32 residue scrambles three quarters of the coordinates (11.1), and what survives has no memory of which multiplier generated it.

11.4 The budget fraction was not a cliff, and the sample is worth more than it costs

`v16` spends 0.799 of `B` and reports a worst case of 0.867. Fifteen percent of the budget sits unused, and 11.2 says nothing prevents spending it. `v25` is `v16` with `BUDGET_FRAC` raised from 0.96 to 1.063 — one constant, nothing else — and it was submitted:

	npair	tracked FLOPs	raw MSE (grader)	score (grader)
v16	40,361	0.7989	2.7066e-07	2.1576e-07
v25	44,753	0.8851	**2.3817e-07**	**2.0925e-07**

A 3.0% improvement, and the first one this project has had in a while. It also settles the exponent that section 1 said decides everything. Two submissions of one estimator differing only in point count give `alpha = ln(2.7066/2.3817) / ln(44753/40361) =` **1.258**, and the paired local measurement on the same hundred networks gives 1.243 independently. Section 1.1's `alpha = 1.01 +- 0.03` was measured across 3,089 to 30,323 points on an earlier estimator; at 40,000 points on this one the exponent is above 1 by enough to matter, and **the budget should be spent, not conserved**.

The consequence is not symmetric. At `alpha = 1.258` a variance reduction of `v` improves the score by `v` directly, while a cost reduction of `c` only buys `c^0.258` — a sample 10% cheaper is worth 2.6%, which is why every lever in 11.5 was worth chasing and none of them would have paid even if they had worked.

11.5 Five levers from the cost side, all dead

Sections 7 and 10 attack the variance. These attack the price per sample, on the observation that the leader spends *more* budget than we do (multiplier 0.935 against our 0.799) and still reports 5.3x less error, which cannot be bought with rows at any 1/N rate.

The activations do collapse. Effective rank of the layer-l activation, at 99% of the spectral energy, falls from 214 at layer 1 to **7.6 at layer 31**. Four levers were built on that collapse and none survived:

- **Low-rank truncation.** Project the activation to rank r and run the rest of the network. Deep layers truncate cheaply but have no layers left to pay for it; shallow layers have the layers but the error at any useful r is at or above 2.76e-07, our whole MSE. And ReLU restores full rank every layer, so a low-rank representation survives exactly one matmul while the SVD that produced it costs what the matmul did.
- **Layer merging.** A neuron whose gate never closes is linear, and a consecutive pair of such layers can be multiplied together in advance — exactly, not approximately. The break-even is 50% of gates frozen open. The measured maximum is **24%** at layer 30; merged cost exceeds unmerged at every layer (1.50x down to 1.26x).
- **Subspace projection.** If the answer lives in 8 of 256 dimensions and the noise is isotropic, killing the other 248 is worth 32x. Handing over the true subspace for free buys **1.035x**. The noise is not isotropic: every row lives in the same collapsed subspace the signal does, so there is no outside to discard. (A second arm appeared to give 1.50x and was withdrawn — it used the true mean to centre, and knowing the mean is the whole problem.)
- **Bucket refinement.** `CUT_ASSUMED = 0.72` against a measured duty cycle of exactly 0.500 looks like 22 points of unclaimed sparsity. Raising the bucket count 32-fold moves the effective kept fraction from 0.7036 to 0.6932, and sorting columns by duty rather than magnitude moves it by 0.0005. The reason is structural: `keep` is set by each row's *last* nonzero column, and 128 nonzeros scattered at random across 256 columns put that last one near the end almost surely. Prefix-based sparsity can only collect the always-dead columns, which is 25%, and 0.70 is already against that bound.

The fifth is the one piece of QMC machinery we had built and never used. `est_m0.60` carries a genuine component-by-component generating vector for the weighted unanchored Sobolev criterion, 256 components at `n = 3167`, guarded behind "only used when the affordable count lands on it" — and the

affordable count is 40361, so it never has. Measured head to head at the one `n` where it exists, **CBC scores 0.917x against Korobov** `a = 17`. The guard was not costing us anything. Given 11.1 this is not surprising in hindsight: a vector optimised for lattice quality has no particular advantage once float32 has scrambled three quarters of the coordinates.

11.6 What this section is worth

One of the ten attempts improved the score, by 3.0%, and it is the one that had already been tried and rejected. The rejection was a benchmark flag. That is the finding here, and it is worth more than the three points: for a full day the project's own measuring instrument was pricing a quantity — wall-clock contention on a laptop — at five thousand times what the grader charges for it, and every candidate was being scored against a baseline that had been measured under a different rule. The failures it produced did not look like measurement error. They looked like collapse: 43 networks out of 100 returning partial results, which reads as a numerical instability, not as an accounting mismatch.

Two constants of `v16` survive with their status changed. `KOROBOV_A = 17` is not a tuned value and not a position against a wall; it is an arbitrary choice that a properly held-out search cannot beat, because the float32 residue has already destroyed whatever the multiplier was supposed to control. `BUDGET_FRAC = 0.96` was simply too low, and 1.063 is better because the exponent is 1.258 rather than the 1.01 we had measured on a different estimator over a different range.

The general lesson we would give a reader who wants to beat this estimator: before comparing two runs, recover the cost rule from each run's own numbers and check that they match, and check both against whatever the grader will tell you. Ours would tell us, and did — one field of the submission API, read a day late.

11.7 How much this benchmark can actually resolve

Reopening the multiplier put a number on something we had been assuming. Eight networks with twelve independent shifts each is the standard harness in this write-up, and it resolves about eleven percent.

The figure comes out of `research/anti.py`, which compared two arms of nearly equal cost across the eight networks. The per-network log ratios were 0.451, 0.289, −0.387, −0.071, 0.448, −0.148, −0.310 and 0.073: a mean of 0.043 against a standard deviation of 0.317, so a standard error of 0.112. A lever worth one percent cannot be seen by this instrument at all, and a lever worth five percent is a coin flip. Every "1.4x on eight networks" in an earlier draft of this document was reporting a number of that size.

Two consequences run through the rest of this section. Anything below about ten percent has to be measured a different way or not claimed, and the way we found is to measure things that have no error bar at all — arithmetic on the deployed activations, and the meter.

11.8 The multiplier has a floor, and it is not a door

The score of one network is `mse * max(0.1, C/B)`. Below `C/B = 0.1` the cost stops being charged. Nothing in this document had ever asked what is down there, and the question has a clean answer: spending down to the floor wins exactly when `mse ~ n^-alpha` has `alpha < 1` between the floor and the ceiling, because at `alpha = 1` the mse rises at precisely the rate the multiplier falls. There is no interior optimum to find — the floor is a corner and the answer is one end or the other.

Two existing numbers disagreed about which end. Section 1.1 fitted `alpha = 1.014` to `est_v11` between 3,089 and 30,323 points. The two graded submissions of the `v16` family give 1.238 across a 10% interval at the top. A concave curve through both would put `alpha` under one somewhere below thirty thousand points, and the floor sits at about 4,750 — deep inside a range nobody had measured.

`research/alphacurve.py` measures it: the deployed pipeline on a geometric grid from 1,999 to 44,987 points, eight networks, twenty-four shifts, scored against `alm`.

`n`	**mse**	`C/B`	**multiplier**	**score**
2,539	3.1591e-06	0.057	**0.100**	3.1591e-07
4,099	2.1179e-06	0.087	**0.100**	2.1179e-07
8,389	8.6917e-07	0.171	0.171	1.4893e-07
21,937	3.6830e-07	0.437	0.437	1.6104e-07
44,987	1.7700e-07	0.890	0.890	1.5747e-07

Across the whole upper range, `alpha = 1.025`: the score is flat in `n` to within the resolution of section 11.7, and the floor is worse than either end of the flat part, not better. The best grid point, 8,389, beats the operating point by 5.4%, which is half a standard error — and the grid point next to it, 10,691, is 13.7% *worse* than the operating point, which is the same noise in the other direction.

So the floor is closed, and the reason to keep spending the budget is not this curve, which cannot see a five percent effect, but the two graded submissions, which can. Their gradient is `d log score / d log n = -0.297`: 2.1576e-07 at 40,361 points and 2.0925e-07 at 44,753. Both instruments agree the answer is at the ceiling; only one of them can measure how far.

11.9 Two more variance levers on the deployed point set, both noise

The first-layer Gram. The estimator already forces the *input* sample covariance to its population value before the first weight. The next moment is also closed-form and was not being used: for `u = W_0' x` with `x` on the sphere of radius `R`, the order-1 arc-cosine kernel gives

$$\mathbb{E}[\mathrm{relu}(u_i),\mathrm{relu}(u_j)] = \frac{R^2}{W},\lVert w_i\rVert\lVert w_j\rVert,\frac{\sin t + (\pi - t)\cos t}{2\pi}$$

with `t` the angle between columns `i` and `j` of `W_0`. This is not the Gaussian surrogate that section 10.5b disposed of; it is sample moment matching, the layer-1 analogue of the input whitening, and the antithetic pair leaves the target unchanged because `relu(-u)` has the same second moment as `relu(u)`.

`research/actwhiten.py` maps the sampled activations by the Cholesky factor that makes their sample Gram exactly this matrix. Over eight networks and twelve shifts it wins on three of them, mean 1.0108x, per-network ratios spanning 0.842 to 1.174. A diagonal-only version is 1.0000x on every network to four figures — the whitening and the antithetic pair had already fixed the diagonal exactly.

The negative is worth more than it looks. Forcing the layer-1 second moment to its exact value does nothing at layer 31, so whatever error survives at the scored layer is not carried by the layer-1 moments, and no amount of further first-and-second-moment work at layer 1 will reach it.

The antithetic pair. `[relu(z), relu(z) - z]` gives `2n` rows for `n` points, and the saving is one layer-0 matmul and one inverse-normal per row — about 3% of the total. The pair was introduced to kill the

odd part of the layer-1 error, but the `carry` term now removes the layer-1 mean exactly, and the paragraph above shows the second moment is not the channel either. If the pairing's whole contribution was those two moments it is now redundant, and `2n` antithetic rows span only `n` points of the lattice, which is what QMC is paid for.

`research/anti.py` compares 40,361 paired points against 80,713 unpaired ones — the same row count, 2.9% more FLOPs, so break-even is 1.0076x. Unpaired comes back at 0.9773x, winning four networks of eight. The pair stays, on a measurement that could not have shown a small win either way; what it can rule out is a large one, and there is no large one.

11.10 The contraction is within 1.5% of everything its own scheme can reach

`_bucketed` contracts each row over a *prefix* of the columns in one global order, cut at the widest row of its bucket. Three questions about that scheme have exact answers, none of which needs a shift or an error bar, because they are arithmetic on the deployed activations.

How much is being left. A row fires on 49.8% of the columns and pays for 70.1% (`research/covercost.py`, four networks, all sparse layers). Contracting each row over exactly its own firing set would cost 0.7097 of the deployed bill, which at `-0.297` is a 7.4% better score. That is the prize, and it is out of reach: a per-row gather of the weight rows costs `out * k` elements at flopscope's `take` weight of 4, which is more than the matmul it replaces.

Whether a bucket can do better than a prefix. No, and not marginally. At sixteen buckets the union of a bucket's firing sets is exactly its prefix — five thousand half-dense patterns cover everything — so the oracle that is allowed an arbitrary column set per bucket prices at 1.0075, i.e. worse than the prefix once its gather is paid for. Ordering the columns by nonzero count rather than by mass is 0.9990. Re-sorting the columns per bucket is 1.0075.

Whether the buckets are the right size. `research/bucketsweep.py` prices NBUCKET from 1 to 1024 against the deployed 16:

NBUCKET	1	4	8	16	32	64	256	per-row ideal
bill	1.2436	1.0476	1.0158	**1.0000**	0.9922	0.9883	0.9854	0.9845

Sixty-four buckets save 1.18% of the arithmetic and everything past that saves nothing. We did not take it. Sixty-four buckets is 1,440 more matmul calls per network, and that time lands in residual wall, which the canonical lambda charges at 1e11 FLOPs per second — at the per-call overhead this run shows, roughly 2.6% paid for 1.18% saved. It is the only lever in this document that is positive in FLOPs and negative in the score.

Spending the bias budget instead. The pilot drops a neuron it never sees fire, and section 7.3 measured that bias at 1/150 of the error we already carry, so 99% of the bias budget is unspent. `research/masscut.py` spends it by dropping neurons that fire on fewer than `tau` of the rows, pricing the bias exactly by running pruned and unpruned forwards on identical points:

tau	keep	bill	bias²	total error	vs today
0.0001	0.7044	0.9976	4.9e-12	2.3747e-07	0.9971
0.001	0.7040	0.9945	4.7e-10	2.3700e-07	**0.9951**
0.003	0.7030	0.9917	5.8e-09	2.4148e-07	1.0139
0.01	0.6990	0.9847	1.2e-07	3.5605e-07	1.4949

The optimum is real, interior, and worth half a percent. The reason it is so small is structural: the prefix length is set by the columns that fire *rarely but somewhere*, and those are exactly the neurons a rare-firing cut removes last. Cutting a neuron that fires often would shorten the prefix, and the bias column shows what that costs.

11.11 Sizing the sample from the meter instead of from a constant

Everything above tries to make a point cheaper or worth more. The largest remaining number was neither: it was the budget the estimator declines to spend.

`v16` chooses `npair` from a closed-form cost model with one fitted constant, `CUT_ASSUMED = 0.72`, calibrated once and applied to every network. The model is conservative by a different amount on each of them, so the realised spend runs from 0.7585 to 0.9604 of the budget around a mean of 0.8851. The ceiling is set by the worst network and the score by the mean, and at `-0.297` the eleven points between them are worth about 3%.

The obvious repair is a per-network version of the same model, and it fails loudly. `est_v27` recomputed the cost from each network's pruned widths and exhausted the budget on all one hundred, because `CUT_ASSUMED` had folded the dead-neuron cut into its 0.72 and applying it to the already-pruned width counted that cut twice. Any better model has the same failure mode waiting behind it.

So we stopped modelling. `flops.budget_summary_dict()` returns the live `flops_used` from inside `predict`. flopscope's one refusal, `UnauthorizedControlError`, covers opening, closing and resetting a session; reading the meter of the session you are already inside is not restricted, and reading it is the whole of what this needs. `est_v28` runs two small calibration batches through the real code path, reads what they actually cost, and solves the two-point line for the marginal cost of a point and the fixed cost of a batch:

$$\text{marginal} = \frac{(m_2 - m_1) - (m_1 - m_0)}{n_B - n_A},\qquad \text{fixed} = (m_1 - m_0) - \text{marginal}\cdot n_A$$

Two sizes rather than one because the per-batch cost is not proportional to the points — the whitening is a Cholesky and a triangular solve of the width, both `O(width^3)` and both independent of the sample, and at a 521-point calibration that fixed part is 2% of the batch, which read as marginal cost would shrink the real batch by the same 2%.

The linear extrapolation from 521 points to forty-odd thousand is sound for a reason specific to this contraction: the bucket keep lengths are *quantiles* of the last-nonzero distribution, not functions of the sample size, so cost per row is asymptotically constant in the number of rows. `research/rowscale.py` confirms it — the per-row bill at 257 points is 0.9936 of its value at 40,361, at 521 points 0.9954, at 4,099 points 0.9983.

The result, on eight networks with `TARGET_FRAC` set to 0.97:

	mean	sd	min	max	failures
`v25`, modelled	0.8851	0.0362	0.7585	0.9604	0
`v28`, metered	0.9700	**0.0012**	0.9686	0.9723	0

The spread falls by a factor of thirty and the mean lands on the target to four figures. What the constant was buying was not safety but ignorance: the model could not tell a network it was overcharging from one it was undercharging, so it had to be wrong about all of them by the width of the worst.

One thing had to be given up for it. `v16` handed every network the same `npair`, so the lattice was built once and ninety-nine of the hundred networks got it for free — worth about 0.76% of the budget each. Sizing per network makes `npair` different every time, so that discount is gone whatever we do; the point cache is removed rather than kept, because keeping it would make the calibration batches free to build after the first network while the real batch still paid, and the marginal cost would read low by exactly the generation rate. The calibration batches themselves cost 1.7% of the budget and their answers are discarded — averaging a 521-point lattice into a 44,000-point one, weighted by size, carries more variance than the single larger lattice does.

12. Where the variance is made, and why every correction was capped

Sections 7 and 10 are a list of thirty-four levers that did not work, and the honest reading of a list like that is that we were unlucky or not clever enough. This section is the attempt to find out which, and the answer turned out to be neither: almost every lever we tried belongs to a family whose ceiling can be measured directly, and the ceilings are small. Four measurements do it, and they agree with each other in places where they did not have to.

12.1 Intervene, do not regress

The first version of this measurement was wrong in a way worth recording, because the mistake is easy to make and produces a confident null.

We wanted to know whether the final-layer error is inherited from earlier layers or made near the output, so we regressed the final error at neuron `j` on the layer-`l` error at neuron `j`, for every `l`, using `alm` — which carries the true mean at all thirty-two layers, not only the scored one. Every coefficient came back at the null value of `1/R`, and the natural conclusion was that the error is made fresh at the end.

It was an artefact. Neuron `j` at layer 30 and neuron `j` at layer 31 are not the same neuron and are not connected in any privileged way; the map between the layers is a dense 256x256 mixing matrix, so a neuron-aligned regression is comparing quantities that have nothing to do with each other. The null was guaranteed by the design.

The fix is to stop regressing and start intervening. Run the deployed pipeline; at layer `l`, overwrite the sample's column means with the truth; carry on to the end; measure the final-layer MSE. That is an oracle — no real method can do better at that layer, because no real method knows the answer — so whatever it leaves behind is a floor on every correction that acts at that layer.

Two forms of the intervention were run, because the network is positively homogeneous and it was not obvious which one the depth would prefer:

```
shift    add T_l − m_l to every row       fixes the mean, leaves the shape
scale    multiply column-wise by T_l/m_l  the multiplicative form
```

They agree to within a few per cent everywhere, which is the main evidence that what follows is a property of the network rather than of the correction we chose.

12.2 The error is made evenly across the depth

`research/whereborn.py`, eight networks, four independent shifts, `npair = 24571`, base final-layer MSE `3.5675e-07`:

layer	shift MSE	gain	scale MSE	gain	share made at or before
0	3.2093e-07	1.11x	3.2279e-07	1.11x	9.9%
1	3.2320e-07	1.10x	3.3380e-07	1.07x	9.1%
2	3.0737e-07	1.16x	3.0192e-07	1.18x	13.8%
4	2.9049e-07	1.23x	2.8160e-07	1.27x	18.7%
8	2.0933e-07	1.70x	2.0692e-07	1.72x	41.2%
12	1.6139e-07	2.21x	1.6166e-07	2.21x	54.8%
16	1.2461e-07	2.86x	1.2457e-07	2.86x	65.0%
20	8.7409e-08	4.08x	8.7865e-08	4.06x	75.5%
24	5.5937e-08	6.38x	5.4713e-08	6.52x	84.3%
28	2.1921e-08	16.27x	2.1411e-08	16.66x	93.9%
29	1.4611e-08	24.42x	1.4573e-08	24.48x	95.9%
30	7.3406e-09	48.60x	7.3976e-09	48.23x	97.9%
31	5.2e-15	—	7.6e-11	—	control

Layer 31 is the answer itself and is in the table as a control: correcting the output must return exactly zero, and it does, to fourteen figures.

Differencing the last column gives the share each layer contributes on its own:

layers	share per layer
1-8	3.9%
9-12	3.4%
13-16	2.6%
17-20	2.6%
21-24	2.2%
25-28	2.4%
29, 30, 31	2.0%, 2.0%, 2.1%

Every layer contributes between two and four per cent, and no layer contributes more than about a tenth. If the contributions were exactly equal and independent the gain at layer `l` would be `32/(31-l)`; the observed gains run about 1.1x that at the front and 1.5x at the back, so late layers carry slightly less than their share, and that is the whole of the structure. There is no dominant layer, no crossover, and no place where the error is born.

Layer 0's 9.9% is the one entry that stands out, and it is the exception that confirms the arithmetic: it is a single "layer" that carries the whitened first contraction, the antithetic pair, and the exact first-moment carry all at once, so it does the work of several.

12.3 What that caps

The consequence is immediate and it is the thing we did not know while writing section 7. **A correction that acts at one layer cannot buy more than that layer's share.** Section 6 measured the exact layer-1 closed forms — the half-normal first moment, the arc-cosine Gram — at a few per cent and called the result disappointing. It is not disappointing; it is the ceiling, and the measurement in section 6 is within noise of it. The same bound retires, without further experiment, every variant of "compute something exactly at the input and propagate it": the input is one layer, and one layer is three per cent.

It also explains the shape of section 7 rather than excusing it. The levers that failed were overwhelmingly single-site: fix the first layer, fix the last layer, fix the kink, fix the Gram. The ones that worked — the antithetic pair, the radius projection, the whitening — are the ones that act on the sample itself and therefore at every layer at once.

The remaining question is whether anything acts at every layer *and* is available without knowing the answer. Sections 12.5 and 12.6 are the two candidates we could construct, and both close.

12.4 The point set, measured against the corrections rather than against nothing

Section 5 put a decent rank-1 lattice at about 1.4x over independent sampling and the tent transform at 1.17x on top. Both figures were measured against a plain i.i.d. baseline. Running the same comparison inside the deployed pipeline, with the sphere projection, the whitening and the antithetic pair applied to *both* arms so that only the point set differs, gives something else.

`research/depthgain.py`, eight networks, sixteen independent draws each, `npair = 24571`, mean-square error against `alm` at every layer:

layer	lattice	i.i.d.	gain
0	9.4601e-07	9.0842e-07	0.96x
4	1.4815e-06	1.4213e-06	0.96x
8	1.1691e-06	1.1326e-06	0.97x
12	9.8414e-07	9.5904e-07	0.97x
16	7.5718e-07	7.4135e-07	0.98x
20	5.7175e-07	5.8335e-07	1.02x
24	4.7519e-07	4.7383e-07	1.00x
28	3.7969e-07	4.0741e-07	1.07x
31	3.5252e-07	3.9194e-07	1.11x

Per network the final-layer gains are 1.10, 1.09, 1.27, 1.07, 0.75, 0.98, 1.28, 1.10 — **1.08 +- 0.06 across the eight, which is 1.3 sigma from nothing.**

This does not contradict section 5 and it is not a retraction of it; it is a different comparison. Section 5 asked what a lattice buys over independent sampling. This asks what it buys *given* that the sample's radius has been fixed to `E||x||` and its covariance has been whitened into the first layer — and those two corrections are themselves a form of stratification, supplying exactly the low-order structure a lattice is good at supplying.

Two honest measurements disagreeing is worth taking apart rather than adjudicating, so `research/arms.py` runs all four combinations on the same eight networks with sixteen draws each:

arm	MSE	paired ratio	
A lattice + sphere + whiten	3.5252e-07	A/B	**0.97x +- 0.07**
B i.i.d. + sphere + whiten	3.6242e-07	B/C	**2.00x +- 0.32**
C i.i.d. + sphere	7.0702e-07	C/D	1.06x +- 0.13
D i.i.d.	6.7406e-07	A/D	1.81x +- 0.15

Ratios are per-network means over the sixteen draws, then averaged across the eight networks, so the error bar is across networks.

The whitening is the entire effect, and the lattice is redundant with it. `A/D = 1.81x` is the figure section 5 was quoting and it survives; `B/C = 2.00x` says a Cholesky solve on the sample covariance accounts for all of it and a little more; `A/B = 0.97x` says that once the solve is there, the rank-1 lattice, the tent transform and the generating-vector search together contribute nothing measurable. Two independent measurements — 1.08 +- 0.06 in the table above, 0.97 +- 0.07 here — both contain one.

The equal-cost ladder in section 8 had already said this from the other direction and we did not hear it. It adds the devices to a lattice rather than to an i.i.d. sample, and its rows read `lattice + tent 1.727x`, then `+ whitening 1.717x` — whitening on top of the lattice is worth `0.994x`. Here the lattice on top of the whitening is worth `0.97x`. Two measurements taken two months apart, in opposite orders, both find nothing, which is what redundancy looks like when you only ever add in one direction and never think to ask which of the two is carrying it.

The mechanism is not mysterious once the numbers are in front of you. Both devices do the same job by different means: a good lattice is a point set whose low-order moments are close to exact, and whitening simply forces the sample's second moment to be exact. Having done the second, the first has nothing left to supply. This is the single most useful thing in this section for anyone building one of these: **the cheap linear-algebra correction replaces the entire quasi-Monte-Carlo apparatus**, and we spent a considerable fraction of this project on the apparatus.

The immediate consequence is that section 7.34 was not a surprise. We predicted that a scrambled Sobol' net would beat Korobov at high point counts and it did not; all four arms tied. They tied because at this depth no point set is doing anything for anyone. It is the same reason the budget exponent came out at `alpha = 1.01` in section 11.8: an estimator whose error falls as `1/n` is an estimator running at the Monte Carlo rate, and we have been running at the Monte Carlo rate the whole time with a lattice bolted to the front of it.

12.5 A 7.7x oracle with no reachable target

If nothing acts at one layer and the point set is spent, the remaining family is post-hoc correction: take the estimate we have and the sample statistics we have, and combine them. `research/errstruct.py` measures the ceilings, on twenty-four shifts and eight networks.

First, there is no bias to remove. Averaging over shifts leaves `1.94e-08` of a `3.57e-07` MSE, which is 5.4% where twenty-four independent shifts would leave 4.2% by chance alone. The estimator is unbiased, as section 4 argued it must be, and the whole of the error is variance.

The oracle ceilings, each fitted with the truth in hand:

oracle	MSE	gain
best global scale per shift	2.1621e-07	1.65x
best additive offset per shift	3.1245e-07	1.14x
best rank-1 of the error	1.9902e-07	1.79x
best rank-4	9.7639e-08	3.66x
best rank-16	1.0587e-08	33.74x

The rank-16 figure is not a result — sixteen parameters fitted on twenty-four observations is mostly fitting the observations — but rank-1 at 1.79x is real, and a single global scale at 1.65x is very nearly all of it. That number is worth holding: 1.65x is almost exactly the gap between our raw error and the best entries running our budget.

Then the covariates. The error is a 256-vector per shift; any statistic of the sample that correlates with it across shifts is a control variate, and the price list in section 10.13 says a pointwise pass over the sample costs 1/512 of one layer's contraction, so a covariate that removes even a tenth is free.

covariate	R^2	gain if exploitable
mean of `a^2`, final layer	0.9272	13.73x
firing fraction	0.0579	1.06x
mean of `|z|` at layer 1	0.0428	1.04x

The null at twenty-four shifts with one fitted coefficient is 0.0417. Two of the three covariates are at the null and one is at 0.93.

The two that carry nothing are the two whose true values we know exactly, and that is not a coincidence — it is the same fact as section 12.2 seen from another side. The layer-1 statistics are uninformative *because* we already force them: the whitening makes the sample's first-layer pre-activation second moment exact by construction, and the first-moment carry makes its mean exact, so what is left at layer 1 has already had its information removed.

The one that carries 93% is the sample's own second moment at the output. Writing `m` for the estimate we report and `s` for the sample mean of `a^2` on the same draw, both are integrals of the same empirical measure through the same kink, and

```
m * sqrt(S / s),    S = E[a^2] the true second moment
```

removes the part of `m`'s fluctuation that `s` shares. `research/anchor.py` measures it against a reference computed on thirty-two times the points:

anchor's own relative error	MSE	gain
0	4.9285e-08	7.69x
1e-5	4.9326e-08	7.68x
1e-4	4.9827e-08	7.60x
3e-4	6.4243e-08	5.90x
1e-3	2.3529e-07	1.61x
3e-3	1.6357e-06	0.23x

7.7x, and the tolerance is generous — the anchor's error enters under a square root, so a relative error `eps` costs about `eps/2` of bias while removing the full variance, and anything under `3e-4` is still worth more than 5x. A 7.7x on the variance axis would move us from 65th to inside the top ten.

We cannot have it, for a reason that took two more measurements to make precise.

12.6 The anchor cannot be bought, with points or with a closure

There are exactly two places a value for `S` could come from: more sampling, or a closed form. Both were measured rather than argued about.

Sampling. An anchor built from the sample has to be **more accurate than the estimate it corrects**, which is a strange requirement when you write it down. `research/anchor_econ.py` measures the sampling error of both quantities directly — no reference run is needed, because the spread across independent shifts is the sampling error — and finds a relative error of `1.050e-03` on the mean and `1.583e-03` on `a^2`. Lattice error falls as `1/n`, so reaching a target `eps` costs `1.583e-03 / eps` times the points:

target eps	x points	MSE gain	total cost	net
1e-4	15.8	7.60x	16.8	2.22x worse
3e-4	5.3	5.90x	6.3	1.06x worse
1e-3	1.6	1.61x	2.6	1.60x worse

Worse at every accuracy, and the best case is a break-even that lands on the wrong side of one. This is section 11.8's flat budget axis wearing a different hat: when error falls as `1/n` and cost rises as `n`, no accuracy can be bought at a discount, including the accuracy of a correction to the same estimator.

A closure. The other source is a closed-form second moment, which would be nearly free — a moment recursion is thirty-one 256x256x256 products, a fifth of a per cent of the budget. The measurement already existed in `research/gsmooth.py` and we had filed it under a different question. Feeding the sample's own `mu` and `sigma` to the Gaussian rectifier formula gives a final MSE `4.86x` worse than estimating directly; adding one Edgeworth term in the third cumulant brings that to `1.53x` worse; and the optimal blend of the closure with the direct estimate is `beta = 0`, which is to say do not blend. A family whose error on the first moment is 1.5x to 4.9x ours will not deliver a second moment at `3e-4`, which is a quarter of our *own* relative error. The door is closed by three orders of magnitude, not by a near miss.

One family survives long enough to be worth naming, because its ceiling is derived rather than measured and then agrees with a measurement. A control variate needs a surrogate `g(x)` whose exact mean is known; the only such surrogate available here is the network with the ReLU kept at layer 1 and replaced by the identity everywhere below,

```
g(x) = relu(x W_1) . (W_2 W_3 ... W_32)
```

whose mean is the half-normal first moment times a precomputed 256x256 product, and which costs one extra contraction per row. It captures the randomness of layer 1 and nothing else. Layer 1's share of the error, from the table in section 12.2, is 9.9%; so the ceiling on the entire exactly-integrable-surrogate family is **1.11x** — which is the number the intervention oracle at layer 0 returns, `1.11x`, arrived at independently. We did not build the surrogate.

12.7 What is left, and what the leaders' numbers look like from here

Putting the four measurements together, the variance axis has a shape:

family	ceiling	how it is closed
single-layer corrections	~3%	error is made evenly across depth (12.2)
exactly-integrable surrogates	1.11x	only layer 1 is integrable (12.6)
point-set choice	1.0	redundant with the whitening (12.4)
global calibration	1.65x	oracle, and no estimator for it
second-moment anchor	7.69x	oracle; target unbuyable (12.5, 12.6)

Nothing in that table is both large and reachable, and we think that is the honest summary of the estimator we shipped: not that it is good, but that the things which would make it better are either small or not obtainable without knowing the answer.

That reframes the leaderboard too. The leading entry holds a raw error of `3.600e-09` at a multiplier of `0.095`, against our `2.382e-07` at `0.885`. Their error is 66.2 times smaller than ours. If we tried to reach it by sampling — and section 12.4 says sampling is exactly what we are doing, whatever is written on the front of it — we would need 66.2 times the points, which puts our multiplier at `58.6`, and our score at

```
3.600e-09 x 58.6 = 2.11e-07
```

against the `2.092e-07` we actually scored. **The same number to within one per cent.** No amount of budget spent on our estimator reaches their score, because what separates us is not how many points we can afford but that their error does not fall as `1/n` at all. That is not a tuning gap. It is a different method, and Phase 2 is the place to build one.

13. The deterministic axis, and four measurements that closed it

Section 12 ended by saying that what separates us from the top of the board is not budget but that their error does not fall as `1/n`. That is a specification, and it can be checked. This section is what happened when we went and built the only deterministic estimator the problem admits, measured where its error comes from, and then tested the one structural property that could have made a cheap deterministic method possible.

None of it shipped. All of it is negative. It is here because the negative results have mechanisms, and because two of them correct things we ourselves had written down wrong.

13.1 What the cost ledger of the board above us actually says

Every submission page carries per-network telemetry, unauthenticated. Read on 7 August, the top of the board bills like this:

rank	who	metered FLOP	C/B	F/C	wall	backend	FLOP/s
1	dpskv5	2.433e+10	0.095	0.940	45.3s	45.1s	5.4e+08
2	joe_wanza	4.687e+10	0.243	0.708	52.5s	48.4s	9.7e+08
3	huang_chung_yi	6.640e+08	0.014	0.181	1.4s	0.2s	3.0e+09
5	dstepanov	1.280e+07	0.580	0.000	1.6s	0.0s	4.8e+09
65	us (324409)	2.386e+11	0.885	0.993	8.0s	7.2s	3.3e+10

Three facts come straight off it. Rank 3 reaches a raw error of `2.02e-07`, which is our own, on **1/359** of our arithmetic. Rank 5 is billed almost entirely through residual wall time — `1.578 s` of work that flopscope never saw — and the organisers opened a thread on 5 August (18132) saying prize contenders will be reviewed on exactly this ratio. And ranks 1 and 2 run their metered arithmetic at `5e8–1e9` FLOP/s, two orders below what a traced dense contraction achieves, which means their arithmetic arrives in a very large number of very small operations.

The public honest field, by contrast, is tightly bunched and we are at the top of it: natasha_stewart `1.551e-07` adjusted, us `2.092e-07`, evaaaz `4.10e-07`, radiant-allomancer `4.98e-07` (after the v0.10.0

regrade), jamesrahenry `1.768e-06`. Every one of those five is a sampler, or a sampler with an analytic last step. Nobody in that group is within two orders of magnitude of rank 1, and nobody in that group has published a method that would be.

So the question worth six weeks is narrow: **is there a deterministic estimator of this expectation whose error is small and does not buy its accuracy with points?**

13.2 One step of Gaussian closure obeys a clean 1/width law

The only deterministic estimator the problem admits is moment propagation. Treat the pre-activation `u = a W` as Gaussian — it is a sum of `width` correlated terms, so this is a central-limit assumption and nothing else — and both rectified moments are closed form:

```
E[u_i^+]       = s_i ( alpha_i Phi(alpha_i) + phi(alpha_i) )
E[u_i^+ u_j^+] = s_i s_j F(alpha_i, alpha_j, rho_ij)
```

with `F` the exact Price/Wick expression, `Phi2` by Gauss-Legendre on `Phi2 = Phi(a)Phi(b) + int_0^rho phi2 dt`. There is exactly one approximation in the whole construction, and it is a finite-width effect, so it should die as some power of the width. Which power decides everything: at `1/width` a first finite-width correction exists and would leave `1/width^2`, which at width 256 is a factor of 256 and lands in sight of rank 1.

`research/widthscale.py` builds our own He MLPs at four widths — bias-free, `W ~ N(0, 2/fan_in)`, depth 32, drawn exactly as chal8's are — and measures two errors against a `1.2e6`-row reference. *One-step* is the analytic `E[a_l]` given the **true** mean and covariance at `l-1`: one step of central-limit error with no accumulation. *Chain* is the free-standing propagation from layer 0. Both are quoted relative to the RMS of the truth so the widths are comparable, and the reference's own sampling floor is subtracted, because a measured exponent means nothing if what is being measured is the reference.

The starting moments are themselves sampled, and at width 512 that noise is the same size as the closure error we are chasing. So the covariance is drawn twice from disjoint rows and the bias is read off the cross product `E[(pA - T)(pB - T)] = b^2 + Var(T)`, with each replica carrying its own mean — sharing one mean leaves that mean's noise inside the cross product, where it is indistinguishable from bias.

Twelve networks per width:

```
width      chain    one-step   ref floor
   64  2.3701e-02  4.6888e-03  3.5843e-04
  128  9.6201e-03  2.0634e-03  2.5949e-04
  256  6.0643e-03  1.0114e-03  2.0528e-04
  512  9.3058e-03  5.5449e-04  1.8243e-04

one-step ~ width^-1.027   (max residual 0.056 in log-error)
chain    ~ width^-0.471   (max residual 0.400 -- not a law)
```

One step of closure is `1/width` to the precision the experiment can resolve. That is the cleanest scaling law anywhere in this write-up, and it says the single-step closure error is a central-limit error and nothing else.

It also says the single step is not enough. At width 256 the one-step error is `1.01e-03` of the truth, which on chal8 is a final-layer MSE of `3.24e-06` against the `4.82e-07` a sample of the shipped size gets: **6.7 times worse.** A first finite-width correction would have to work, and it would have to be computable without the true moments it is correcting — the same trap section 12.5 documented for the second-moment anchor.

13.3 The chain does not obey it, and the accumulation is ordinary

The chain has no law. Its exponent fits at `-0.471` with a log residual of `0.400`, and the width-512 networks are *worse* than the width-256 ones. It saturates around one per cent of the truth and stops caring about the width.

`research/chaindepth.py` asks why, by starting the closure at layer `k` from the **true** moments and closing only the remaining `31-k` layers. Width 256, six networks, same cross-product estimator; `k = 31` is the control, no closure at all, and has to come out at zero:

```
layers closed     rel err
            0  2.1339e-04   (control -- this is the estimator's resolution)
            1  8.7512e-04
            2  1.3469e-03
            4  1.8407e-03
            7  2.4073e-03
           11  3.0141e-03
           15  3.7400e-03
           19  4.5471e-03
           23  5.1360e-03
           27  5.5171e-03
           31  7.9181e-03
    full chain  7.4162e-03
```

The growth is `(layers closed)^0.5` to `^0.6`. That is ordinary quadrature addition of independent per-layer errors — no amplification worth the name, and no anticorrelated cancellation either. `sqrt(31)` times the one-layer error is `4.9e-03` against the `7.4e-03` measured, so there is a little amplification and that is all.

Which leaves the width-512 anomaly unexplained: quadrature addition predicts `sqrt(31) x 5.5e-04 = 3.1e-03` there, and we measure `9.3e-03`. We did not isolate it. The most likely candidate is that the propagated covariance becomes strongly degenerate at depth — section 13.5 measures a participation ratio of `2.9` at layer 31 — so a large fraction of the `rho_ij` fed to `F` sit against the `+-0.999999` clamp, where the bivariate expression is numerically delicate and the clamp itself is doing the arithmetic. That is a hypothesis, not a result, and we are flagging it as one.

Either way the conclusion for the estimator is the same and it is not close: a free-standing deterministic chain is a one-per-cent method, our sampler is a `1e-04` method, and no width available in this challenge changes that ordering.

13.4 A correction: our own layer-0 closed form was wrong by 1/width

Layer 0 is the one place where the closure is not an approximation. `z = x W_0` with `x ~ N(0, I)` is exactly Gaussian, so `E[a_0]` and `Cov(a_0)` are the arc-cosine kernel exactly. We had been computing it with

```
tau = E||x||^2_sphere / d = (E||x||)^2 / d = 1 - 1/(2d) + ...
```

because our sampler draws on the sphere of radius `E||x||` — the radius normalisation of section 4, which is exact for the *mean* because a bias-free ReLU net is homogeneous of degree one. It is not exact for the second moment, and using it as the scale of the arc-cosine kernel injects a systematic error of order `1/width` at layer 0, which then feeds all 31 layers above:

width	tau = (E||x||)^2/d	tau = 1	reference floor
64	3.9958e-03	8.2521e-04	1.3283e-03
128	2.2129e-03	1.9043e-04	1.3319e-03
256	9.0827e-04	0.0000e+00	1.3339e-03

With `tau = 1` the layer-0 residual is at or below the reference's own floor at every width; with the sphere scale it is a clean `1/width` systematic. The organisers' bundled covariance-propagation example does not have this bug — it starts from `cov = eye(width)` — and neither does anything we submitted, since none of our shipped estimators call the closed form. It was ours alone, in `research/aprop.py`, and it inflated the chain numbers we quoted internally on 5 August by about a tenth. The direction of that conclusion does not move.

The general shape of the mistake is worth more than the number: a device that is exact for one functional was carried over to a different functional where it is not. Homogeneity buys the mean and only the mean.

13.5 The rank collapse is real, and it does not help

The one structural fact that could have made a cheap deterministic method possible is the depth-induced collapse of the activation covariance. jamesrahenry's write-up (forum 18097) reports the pre-activation participation ratio contracting `128 -> 5.2` over the 32 layers. If the cloud that matters is five-dimensional, then five dimensions can be integrated by quadrature to a precision Monte Carlo cannot buy at any budget, and rank 3's `6.6e08` FLOPs stop being mysterious.

We measure the collapse on chal8 and it is real, and larger than that:

layer	PR	variance share of the top r directions				
		r=1	r=2	r=8	r=32	r=64
0	165.3	0.0123	0.0242	0.0913	0.3109	0.5260
8	21.3	0.1535	0.2277	0.4899	0.8213	0.9228
16	9.1	0.2935	0.3899	0.6742	0.9073	0.9628
24	5.1	0.4321	0.5291	0.7775	0.9432	0.9788
31	2.9	0.5821	0.6656	0.8604	0.9691	0.9894

The participation ratio falls from 165 to 2.9. It is also, read alone, misleading: `PR` is `(sum lambda)^2 / sum lambda^2` and a single dominant eigenvalue drags it down while the spectrum keeps a heavy tail. At layer

31 the leading direction carries 58% of the variance and the top **sixty-four** still leave 1.1% outside.

Rank is a statement about second moments, and `E[ReLU]` is not a second moment. A direction carrying a thousandth of the variance still matters if it is what pushes a neuron across its kink. So the honest test is a projection oracle, and it can be made free of sampling noise entirely: push one sample to layer `l`, replace each row by its projection onto the mean plus the top `r` eigendirections of that layer's own covariance, push the projected cloud the rest of the way, and compare the final-layer mean against what **the same rows** gave unprojected. The difference is exactly what a rank-`r` carrier would commit, paired, with the sampling error common to both sides and cancelled (`research/rankcollapse.py`, 8 networks, 98,304 rows).

Final-layer MSE committed by keeping only rank `r` at layer `l`, as a multiple of the `2.382e-07` our shipped estimator actually scores:

layer	r=1	r=2	r=4	r=8	r=16	r=32	r=64
8	27892.1	19164.5	11490.0	6108.1	2547.2	648.1	106.3
16	19400.1	13153.6	6557.1	2705.6	730.2	169.6	26.3
24	7065.6	4690.9	2310.5	845.6	209.0	42.4	6.2
28	2420.9	1486.4	708.3	232.0	57.4	11.1	1.3
30	893.0	585.2	260.7	87.0	20.5	3.8	0.5

One cell in the table is below one. A rank-64 carrier, introduced at layer 30, is 2.2 times better than our sampler; a rank-64 carrier at layer 28 is already worse than it, and at layer 24 it is six times worse. Everything cheap enough to integrate by quadrature — rank 4, rank 8 — is between two and four orders of magnitude above the sampling error at every depth.

This is the same wall jamesrahenry hit from the other side. They measured that the sampling *noise* concentrates in the leading directions at `lambda_i/N`, so splitting the subspace leaves the analytic side only 1.7% of the error to remove. We measure that the *truncation itself* costs more than the sampling error it would replace. Two different quantities, one conclusion: the collapse is real and there is nothing in it to spend.

13.6 What this leaves for Phase 2

Four measurements, four closures. One step of Gaussian closure is a clean `1/width` central-limit error and is 6.7 times worse than our sampler at the width we are given. Chained, it accumulates in quadrature to a one-per-cent method that stops improving with width. The activation covariance does collapse to a participation ratio under 3, and no truncation cheap enough to integrate deterministically is within two orders of the sampling error. And the exactly integrable device we do have — homogeneity — buys the mean and nothing else, which is what section 13.4 is about.

So we can now say precisely what the top of the leaderboard is not doing. It is not moment propagation; that is a one-per-cent method. It is not low-rank quadrature; the truncation error forbids it. It is not sampling; section 12.7 showed that reaching `3.6e-09` by sampling costs exactly the score we already have. And it is not a control variate or a learned corrector, because radiant-allomancer, evaaaz and we have now each measured that class independently and put it at `1.0x` to `1.2x`.

What is left, from the ledger in section 13.1, is an arithmetic path that costs less than it does. Ranks 1 and 2 run at `5e8`–`1e9` metered FLOP/s. Rank 5 is billed almost entirely for work flopscope never saw.

The forum has four separate metering-bug reports on file — negative charges for empty contractions (18076), batched right-hand sides undercounted in `linalg.solve` (18082), a budget summary whose cost grows with process history (18092), a general accounting bypass (18099) — and the organisers have said prize contenders will be reviewed on instrumented share (18132). We are at `0.993`. We are not making an accusation, and we have not reproduced any of those bugs; we are recording that after eliminating every algorithmic explanation we could construct or measure, the one hypothesis our own data cannot rule out is the metering.

That is also the honest answer to what we would build in Phase 2. Not a faster sampler: section 12 says the sampling axis is closed, and the flat budget axis says a faster one would score the same. The open problem is the one this section failed to solve — **a deterministic estimator of `E[ReLU]` through 32 layers whose error is set by something other than the width** — and the four measurements above are, we think, the most useful thing we can hand the next person who tries it, because they say where not to look.

14. The ceiling was real: a counting argument that closes the sampling axis

Section 8 derived a harmonic ceiling on this estimator, measured that our own deployed code violated it, and withdrew it. The withdrawal was right about the arithmetic and wrong about the measurement. Redone with the weighting the score actually uses, the ceiling holds, we are within nine percent of it, and the reason the remainder is unreachable is not that we were not clever enough — it is that there are not enough points in the sample to reach it, by a factor of two thousand.

14.1 What was measured wrong

Section 8 estimated the harmonic spectrum "on 20,000 antithetic pairs against the highest-variance neurons of three networks". The scored quantity is the mean over all 256 output neurons of the squared error, so the spectrum that governs it is the one averaged over all 256 neurons, not the one of the loudest few. The two are not close.

We remeasured with the two-point function. For `f` on the sphere,

```
C(t) = mean_j Cov( f_j(u), f_j(v) )       at   u . v = t
     = sum_l sigma^2_l P_l(t)
```

with `P_l` the degree-`l` Gegenbauer normalised to `P_l(1) = 1`. In dimension 256 `P_l(t) -> t^l` to `O(1/d)`, so `C` is essentially a power series and the even and odd parts separate for free:

```
E(t) = (C(t) + C(-t)) / 2      the part antithetic pairing leaves behind
O(t) = (C(t) - C(-t)) / 2      the part it annihilates
```

`E(1)` is the pair-mean variance and `O/E` at `t = 1` is the odd/even split. The truth is the bundled label, so no reference sample is needed and `C` is unbiased. Eight networks, 65,536 pairs at each of twenty-one inner products (`research/harmonic.py`):

	section 8 (3 networks, loudest neurons)	this measurement (8 networks, all 256)
degree 1	32.5%	24.5%
all odd, killed by pairing	59.1%	55.8%
degree 2, killed by whitening	**3.6%**	**18.1%**
even degree >= 4	**37.3%**	**26.1%**

The odd side barely moves. Degree two moves by a factor of five, and it is the one entry the argument turns on.

One trap, because it cost us an hour. Fitting `E(t)` by ordinary least squares in the monomials returns alternating coefficients — `+13.3` at degree 10 against `-9.4` at degree 8 — because the monomials are hopelessly collinear on `[0,1]`. The coefficients are variances and are non-negative by construction, so the fit is a non-negative least squares problem; solved that way it is stable and the residual is `1.1e-3`. The unconstrained numbers are meaningless and we do not report them.

14.2 The ceiling, in the units that matter

Section 8's arithmetic correction stands: `E(1)` is a *per-pair* variance and a pair costs two forward passes, so the equal-cost bound on the whole moment-matching family is `1 / (2 * (E(1) - sigma^2_2))`. With the corrected spectrum:

device	degrees it annihilates	share of the integrand's variance
antithetic pairing	every odd degree	0.5580
input whitening	degree 2	0.1809
nothing in this family	**even degree >= 4**	**0.2611**

`1 / (2 * 0.2611) = 1.915x` over plain i.i.d. sampling on the sphere, at equal cost. Our deployed estimator is `1.755x` over plain i.i.d. *Gaussian* sampling (section 0) and radius normalisation accounts for `1.014x` of that, so against the same baseline the ceiling bounds it is at `1.731x`.

The deployed estimator sits at 90.4% of the bound. Section 8 withdrew the ceiling because `1.34x` was below `1.755x` and a bound one's own code violates is not a bound; that reasoning was correct and the input to it was not. With the spectrum measured the way the score weights it, nothing is violated and the remaining headroom in the entire family is nine percent.

This also resolves a loose end section 8 left. It explained the exact arc-cosine Gram correction being worth only `1.048x` by saying degree two carries 3.6% of the variance and no correction to a 3.6% contribution can pay. Degree two carries 18.1%, so that explanation fails — and the right one is better: **input whitening has already removed degree two**, so a second-moment correction one layer downstream arrives to find nothing left. The direct test agrees. Forcing the layer-1 post-ReLU sample Gram onto its exact arc-cosine value, on top of the deployed pipeline, is worth `1.011x` across eight networks and wins on three of them (`research/actwhiten.py`) — noise.

14.3 Why the remainder is unreachable: count the degrees of freedom

Every device in this note is a moment matcher. It forces the sample to reproduce the population value of some moment and removes exactly the variance living in that degree. So the ceiling is not a question about cleverness. It is a question about how many constraints a sample of `N` points can be made to satisfy.

`N` points on `S^{d-1}` carry `N(d-1)` degrees of freedom. Matching degree `l` exactly imposes `dim(l) = C(d+l-1, l) - C(d+l-3, l-2)` independent conditions. `N(d-1) >= dim(l)` is therefore *necessary*, and for `d = 256` against our shipped `N = 89,506` rows it is already decisive:

degree	`dim(l)`	`N(d-1)/dim(l)`	points needed	vs our `N`
2	32,895	694	129	0.001x
3	2,828,800	8.07	11,093	0.12x
4	**183,148,480**	**0.125**	**718,229**	**8.02x**
5	9,522,602,496	0.002	37,343,539	417x
6	414,173,091,136	0.0002	1,624,208,201	18,146x

Degrees 1 through 3 are cheap — which is exactly why the two devices that work, antithetic pairing and whitening, work: degree 2 needs 129 points and whitening spends a `256 x 256` linear map on it, degree 3 needs 11,093 and the pairing gets every odd degree for free. Degree 4 is where it stops, and degree 6 and up are four orders of magnitude outside the problem. That is the whole of it: sections 5, 7 and 10 are not thirty-four independent disappointments. Korobov moduli, CBC construction, scrambled Sobol', interlaced higher-order nets, the baker transform, stratification and per-network selection all run into the same wall, and the wall is the shape of the space rather than a property of any of them.

Degree 4 is the one honest maybe, and it is closed three times over. It deserves the arithmetic because it is the only step that is not absurd.

The budget does not forbid it. Since `score = k c / B` with `k` the variance per row, the point count cancels; buying `8x` more rows costs `8x` more and scores the same, so `N` is not budget-limited at all.

The wall clock does. Our submission spends `8.0 s` of the `60 s` allowed per MLP, so `N` can grow by at most about `7.5x`. Degree 4 needs `8.02x`. We miss the *necessary* condition by a factor of `1.07` — close enough to be irritating and not close enough to matter.

Nothing constructs it anyway. The condition above is necessary, not sufficient: one still has to exhibit a spherical 5-design on `S^255` with about `7 x 10^5` points, and none is known. The obvious cheap candidate does not work and fails for an instructive reason. Take the `+-1/sqrt(d)` images of a binary code whose dual distance is at least 6 — an orthogonal array of strength 5, for instance the `[256, 16]` dual of the extended double-error-correcting BCH code, giving `2^16 = 65,536` points at no construction cost. It is an orthogonal array of strength 5 and it is still only a *spherical 3*-design, because the hypercube itself already fails at degree four: every vertex has `x_1^2 = 1/d` exactly, so `E[x_1^4] = 1/d^2 = 1.526e-5`, against `3/(d(d+2)) = 4.542e-5` on the sphere — a factor of `3d/(d+2) = 2.98` off, with no code able to repair it. Strength in the cube's own scheme is not strength on the sphere.

And the prize is small. Grant all three. Antithetic pairing and whitening already take every degree up to 3, so a 5-design buys degree 4 and nothing else: `0.0580` of the variance, out of the `0.2611` that survives. The per-evaluation variance falls from `2 x 0.2611 = 0.5222 V` to `2 x 0.2031 = 0.4062 V`, a gain of **1.286x**.

The equal-cost gap to the three entries above us is `1.73x` to `2.61x`. Even the impossible version of this does not close it.

14.4 The two escapes, and why neither is open

The counting argument has two honest loopholes. Both were checked and both are closed; section 8 reported the first and we sharpen the second.

High degree is not the same as high dimension. A function can be of very high degree and still depend on a handful of directions, and degree-4 matching *inside* a `k`-dimensional active subspace costs `C(k+3,4)`, which is affordable for small `k`. The active-subspace matrix `C = E[J^T J]`, `J` the Jacobian of the final activations, decides it: 90% of the Jacobian energy needs **117 of 256** input directions and 99% needs 190, against 230 for a perfectly flat spectrum — a concentration factor of two, not of thirty. Degree four inside the 117 directions is still `8.2e6`, ninety times our sample. The 35 directions that carry half the energy would fit, at `7.3e4` — but they carry half the *first order* sensitivity, which is not the same as half the degree-four variance, and half of a `1.22x` correction is not a lever.

Cost, not variance. The score is `k * c / B` and if `k` is at its floor the only remaining term is the billed cost per row. A layer costs `2 W^2` because each point is carried as a full 256-vector; if the sample's fluctuation about its own mean lived in `r << 256` directions the layer could be carried as a mean plus a rank-`r` factor at about `3 W r`, which is `5.3x` cheaper at `r = 16` and `10.7x` at `r = 8`. The participation ratio of the centred activations does collapse — 106 at layer 1 to 4.4 at layer 24 — and that collapse is what makes the idea look plausible. It is the wrong statistic. The rank needed to hold `1 - 1e-6` of the centred energy, which is what a `1e-7` target requires, is **256 at layer 1, 231 at layer 8, and still 196 at layer 24** (`research/actrank.py`): the spectrum concentrates but its tail does not thin. This agrees, from the other side, with the paired projection oracle of section 13 — truncating at rank 64 from layer 24 costs `6.2x` our deployed error. A rank carry cannot be bought.

14.5 What this says about the front of the field

Put together with section 12's finding that our error is 98.7% variance and 1.5% bias, and with the equal-cost gap measured on the public telemetry — `1.73x` to `2.61x` at matched budget against three entries immediately above us, and, as section 14.6 reads off the front's own ledger, a factor of six hundred against the top of the board — the conclusion is forced and it is not comfortable:

our estimator is at the floor of its class, and the class is the wrong one.

Not a faster sampler; not a better point set; not another exact identity. A sampler priced by this meter cannot get there, and we can now say that with a count rather than with a list of failures. Whatever the entries above us are doing, it does not have a `1/n` error, and section 13 is the record of our failing to build one.

14.6 The front's own ledger says it is not a sampler

Section 15.1 discloses how we read other submissions. The per-network telemetry under each graded submission carries `flops_used`, `effective_compute` and the flopscope timing breakdown, and the median row of it is a complete cost statement. Three facts fall straight out of it, and together they close the question of whether the entries above us are doing a better version of what we are doing.

Instrumented time is free. The charged cost is `C = flops_used + lambda * residual_wall_time` with `residual = wall - backend - overhead`, so every second spent *inside* a flopscope-metered operation costs

nothing beyond the FLOPs it declares. The leading entry spends 45.3 seconds of wall clock and is charged for 0.015 seconds of it. We optimised our estimator for throughput and get 33.1 GFLOP/s out of the meter; the top three run at 0.13, 0.54 and 0.89 GFLOP/s. That is not a handicap they are overcoming. It is a dimension of the problem that does not exist: the budget is a FLOP budget, and being slow is not spending anything.

They cannot be sampling. A row of our estimator costs `32 * 2 W^2 = 4.19e06` billed FLOPs. Dividing each leader's billed FLOPs per network by that number gives the largest sample they could possibly have drawn:

rank	who	raw error	billed/net	row-equivalents	GFLOP/s
1	dpskv5	3.600e-09	2.434e10	5,802	0.537
2	huang_chung_yi	9.000e-09	6.187e09	1,475	0.132
3	joe_wanza	4.000e-09	4.687e10	11,174	0.893
	us (324409)	2.382e-07	2.390e11	56,968	33.052

Plain i.i.d. sampling on the sphere at 11,174 rows sits at `0.0353 / 11174 = 3.2e-06`, and section 14.2's ceiling says no point set and no moment matcher improves on that by more than `1.915x`. The best a sampler can do with joe_wanza's budget is `1.6e-06`; they report `4.0e-09`, four hundred times lower. At rank 2 the gap is worse still — 1,475 rows bound a sampler at `1.2e-05`, and the reported error is `9.0e-09`. Whatever the front is running, it is not an average over draws, and no refinement of ours becomes it.

The shape of the spend points at pairs. Divide the billed FLOPs by the 32 layers and again by the `W^2 = 65,536` ordered neuron pairs in a layer:

huang_chung_yi	2,950 FLOPs per neuron pair per layer
dpskv5	11,600
joe_wanza	22,400

Those are the sizes of a small two-dimensional quadrature — a 30x30 to 90x90 node grid with a handful of operations at each node — evaluated once for every pair of neurons at every layer. They are far too large for a closed-form pairwise formula (the arc-cosine kernel that a Gaussian closure needs is a dozen operations per pair) and far too small for anything that touches triples. The rate corroborates it: 0.13 to 0.89 GFLOP/s is what numpy delivers on memory-bound elementwise work over arrays of a few times `10^7` elements, which is exactly `W^2` pairs times a few thousand nodes.

Subtracting a backbone sharpens all three numbers. Any closure that carries a covariance pays for the transitions `C_{l+1} = W^T S_l W` before it pays for anything else: two dense `256^3` products at each of 31 transitions is `2.08e09` billed FLOPs, and section 14.2's price list gives contractions no symmetry discount, so that figure is a floor rather than an estimate. What is left over, spread across the `31 * W^2 = 2,031,616` pair evaluations, is

huang_chung_yi	6.187e09 - 2.08e09	=	4.11e09	2,021 FLOPs per pair
dpskv5	2.434e10 - 2.08e09	=	2.23e10	10,957
joe_wanza	4.687e10 - 2.08e09	=	4.48e10	22,046

At five operations a node that is a bivariate quadrature on a 20x20, a 47x47 and a 66x66 grid. The three of them are running the same shape of calculation at three fidelities.

The same accounting corrects something we asserted earlier in this section. `huang_chung_yi` has two graded submissions, at `6.596e08` and `6.187e09` billed FLOPs, for raw errors of `2.016e-07` and `9.000e-09`, and we read that pair as one method scaled up, giving a cost-error exponent of `-1.4` against a sampler's `-1.0`. It cannot be one method: `6.596e08` is *below* the `2.08e09` backbone, so the cheaper submission is not propagating a covariance at all. The exponent was fitted across a change of algorithm and we withdraw it. What survives is a sharper observation about the cheap run itself — it reaches `2.016e-07`, which is our own raw error, on `1/362` of our billed cost, without carrying a covariance.

14.7 Two more routes closed: the layer skip and the moment ladder

Section 14.4 closed the cost axis for a low-rank carry. Two other ways to make a row cheaper, or to replace the row entirely, were still open. Both are now measured and both are dead, and the way they die is more useful than the fact.

Restarting from a matched Gaussian. A row costs 32 layers because it is carried from the input. The pre-activation of layer `k` is a sum of 256 terms, so replacing its law by the Gaussian with the same mean and covariance should be nearly free, and it would let a row start at layer `k` for the price of one matvec — cost `(32-k)/32`, which is a straight `2x` at `k = 16` and `4x` at `k = 24` on a score that is linear in cost. `research/gaussrestart.py` hands the oracle the exact `m_k` and `C_k` from a two-million-row reference pass and measures the bias at the output. Eight networks, four replicates of 524,288 rows each; `k = 0` is a control, because `p_0` is exactly Gaussian and must come back at zero:

k	cost/row	bias^2	vs control	score if it worked
0	1.000	2.65e-08	1.0x	(control)
1	0.969	2.92e-06	110x	1.03x
8	0.750	7.76e-06	293x	1.33x
16	0.500	7.35e-06	277x	2.00x
24	0.250	6.08e-06	229x	4.00x
28	0.125	3.22e-06	121x	8.00x

Substituting a matched Gaussian at a **single** layer costs between `2.9e-06` and `8.1e-06` at the output — twelve to thirty-four times our entire deployed error of `2.382e-07`, against a control that reads `2.65e-08`. There is no `k` at which the trade is close. The curve is also nearly flat in `k`: the first hidden layer is as unforgiving as the twenty-eighth, so the non-Gaussianity is not born anywhere in particular and there is no shallow prefix to give away.

This is the same statement as the `5.76e-05` that `research/kscale.py` measures for `K = 2` cumulant propagation, which substitutes at all 32 layers, seen one layer at a time. It also explains why that number is not 32 times ours: a later substitution partly absorbs the error an earlier one introduced, because each layer's Gaussian is fitted to the moments it is handed, wrong or not.

Reconstructing the final marginal from its moments. `E[relu(p_{31,j})]` depends on nothing but the marginal law of `p_{31,j}`, so a method that knew that marginal exactly would be exact. The question is how much of it a moment summary carries. Given `2n` moments, the `n`-point Gauss rule is the canonical reconstruction — the unique `n`-atom measure matching all of them. `research/momentfloor.py` builds it per neuron from a 134-million-row reference pass, whose own noise on the answer is `6.9e-10`:

moments	mse	vs leader (3.6e-09)	vs ours (2.382e-07)
2	2.107e-04	58,514x	884x
4	2.685e-05	7,459x	113x
6	1.220e-05	3,389x	51x
8	7.959e-06	2,211x	33x
10	4.512e-06	1,253x	19x
12	2.894e-06	804x	12x

The ladder falls as `n^-2`, so reaching the leader from twelve moments needs about 170 atoms, i.e. **340 moments per neuron**. That is not a summary.

The interesting part is the comparison across families rather than along the ladder. Two moments read as a smooth Gaussian density score `8.6e-07` (section 10.11's Gaussian floor); twelve moments matched exactly but read as atoms score `2.9e-06`, three times worse. A kinked integrand is badly served by a discrete representation no matter how many moments it honours. What buys accuracy here is the assumed *shape*, not the moment count — which is why the tabulated `h(u)` with a per-neuron skewness reaches `4.7e-08` on two and a half moments, and why the Edgeworth correction, which perturbs the shape in the wrong family, makes things four times worse.

14.8 The missing information is joint, not marginal

Both of the closed routes above leave one construction untested, and it is the one the front's per-pair spending points at. A matched Gaussian gets the marginals wrong *and* the dependence wrong. Which of the two costs the `2.9e-06`?

`research/copularestart.py` separates them. It restarts at layer `k` from three laws built on the same reference sample: the matched Gaussian; a distribution with the **exact per-neuron marginals** — 4,096 empirical quantile knots each — tied together by a Gaussian copula fitted on normal scores; and no substitution at all, as a control. Six networks, four replicates of 524,288 rows:

k	control	matched gaussian	exact marginals	marginals' share
1	-1.03e-08	2.83e-06	4.22e-06	-49.1%
8	4.79e-09	8.09e-06	6.65e-06	17.9%
16	-1.33e-08	7.17e-06	5.95e-06	17.1%
24	-1.67e-08	4.75e-06	4.16e-06	12.5%

Making every marginal exact removes **12 to 18 percent** of the error, and at layer 1 it removes none at all — forcing exact marginals through a Gaussian copula there is worse than being consistently Gaussian. Four fifths of what a Gaussian substitution destroys is not in the marginals. It is in the dependence, and specifically in the part of the dependence that a Gaussian copula does not reproduce, since the copula arm matches the correlation structure by construction.

That closes the cheapest attractive family outright. Propagating a few per-neuron cumulants — diagonal third and fourth cumulants cost one matvec each against the elementwise cube and fourth power of the weight matrix, so the whole scheme is four matvecs a layer — cannot work, because the object it summarises carries at most a fifth of the information. It also explains, from our side, the number we read off the front's ledger in 14.6: they spend `10^3` to `10^4` FLOPs per **neuron pair** per layer because

the pair is where the information is, and a per-neuron summary is not merely worse, it is looking in the wrong place.

For Phase 2 that is the whole brief in one line. The object to propagate is the joint law beyond its second moment, the cheapest representation that can carry it appears to be pairwise, and the budget is a FLOP budget in which wall time is free.

14.9 Three quarters of the neurons are already solved

Every number in this write-up before this point treats the 256 output neurons as one population. They are not. Write `c_j` for the mean of the final pre-activation divided by its standard deviation. Over four networks and 1,024 neurons, measured from 134 million reference rows,

quantile	5%	25%	50%	75%	95%
|c|	0.36	1.93	3.49	4.32	5.04

A neuron at `|c| = 4` is not near a kink. Its pre-activation is positive on all but one draw in thirty thousand, so the rectifier is the identity there and the answer is the mean; at `c = -4` the rectifier is zero on all but those draws and the answer is a tail integral of order `7e-06 sigma`. Only a minority of neurons live where the shape of the law matters at all, and the Gaussian read-out with exact `mu` and `sigma` — `9.33e-07` overall, section 10.11's `8.6062e-07` Gaussian floor reproduced independently here — divides accordingly:

|c| band	neurons	mean squared error	share of the total
[0.0, 0.5)	71	1.867e-06	13.9%
[0.5, 1.0)	61	3.749e-06	24.0%
[1.0, 2.0)	133	4.280e-06	59.6%
[2.0, 3.0)	159	1.490e-07	2.5%
[3.0, 5.0)	541	1.127e-09	0.1%
[5.0, inf)	59	2.854e-09	0.0%

The 265 neurons with `|c| < 2` are a quarter of the population and carry 97.5% of the error. The 600 neurons above `|c| = 3` are three fifths of it and carry one part in a thousand — and at `1.1e-09` the plain Gaussian read is already three times better *there* than the leaderboard's best raw error is anywhere. The problem is not 256 neurons per network. It is about 66 of them, and a deterministic method's entire budget belongs to those.

This is not a lever for us, and the reason is worth stating. It would seem to follow that we could replace our sampled estimate by the closed form wherever `|c|` is large and pick up two orders of magnitude on three fifths of the output. We cannot, and the argument that says so is one line: where the neuron is saturated on, `relu(p) = p` almost surely, so the sample mean of `relu(p)` and the sample mean of `p` are the same number, and the closed form evaluated at our own `m-hat` and `sigma-hat` returns that same number back. The `1.1e-09` above is what the closed form achieves *given the exact moments*, which cost 134 million rows to obtain. A correction has to be more accurate than the estimate it corrects; here it is the estimate it corrects. Section 12.6 records the same shape of failure for the second-moment anchor, whose 7.7x oracle turned into a net loss at every precision once the target had to be bought.

The depth profile says where the structure comes from. Sweeping 33 million rows through all 32 layers and comparing, at each layer, the Gaussian read-out against the sample mean of the rectifier on the same draws:

```
layer    |c| median   frac |c| < 2    gaussian read-out, relative rms
    0         0.000         100.0%                4.53e-05
    1         0.455          99.8%                2.20e-03
    3         0.828          91.1%                2.70e-03
    8         1.592          60.0%                2.14e-03
   16         2.085          48.3%                2.25e-03
   24         2.615          37.0%                1.63e-03
   31         3.493          25.9%                1.19e-03
```

Layer zero is `x W_0` with `x` standard normal, so its pre-activation is exactly Gaussian; the `4.5e-05` there is this run's own sampling noise and the closed form is exact. One rectifier later the error is `2.2e-03`, and it never gets materially worse. **The non-Gaussianity that costs anything is created by the first ReLU and by nothing after it.** What changes with depth is only the saturation, which climbs monotonically and slowly drags the cost of a Gaussian read back down. Two effects, opposite signs, nearly cancelling — which is why 32 iterated substitutions cost only seven to twenty times one substitution (14.7) instead of thirty-two.

`natasha_stewart` reached the same structure from the other side, and published it: the write-up behind their `1.551e-07` classifies neurons as dead, on, or kinked and linearises the last two layers. That is this table used as an algorithm, and it is the best published honest score on the board.

14.10 The moments do not determine the answer. The assumed shape does.

Section 14.7 measured a reconstruction: exact `2n` moments of the final pre-activation, read through the `n`-point Gauss rule, reaching `2.9e-06` at twelve moments and falling as `n^-2`. That is a statement about one reconstruction. The sharper question is what the moments *determine*, and it is a linear program rather than an experiment. Over measures on a grid,

```
min / max   int max(u + c, 0) dmu(u)     subject to   int u^k dmu = mu_k,  k <= K
```

and the width of the resulting interval is an impossibility statement: two admissible worlds sit at its two ends, the `K` moments do not distinguish them, so no method whose only input is those moments can be more accurate than half the width. Restricting the grid to a finite window can only narrow the interval, so the numbers below are lower bounds — the honest direction for a bound of this kind. `research/momentbracket.py` solves it per neuron on 2,401 grid points spanning nine standard deviations, in a shifted Legendre basis because the monomial rows at `K = 12` span eleven orders of magnitude and HiGHS declares them infeasible. At `K = 2` the answer is known in closed form — `max E[X^+] = (c + sqrt(c^2+1))/2` and `min E[X^+] = max(c, 0)` for unit variance — and the program reproduces both to `1e-12`, apart from the lower end at `c = 0`, which needs an atom at minus infinity and which the finite window therefore misses by `0.056`. Expressed as the mean squared error a minimax method would have to accept:

K	half-width^2	vs leader 3.6e-09	vs our 2.09e-07
2	2.287e-04	63,520x	1,094x
4	4.390e-05	12,190x	210x
6	2.221e-05	6,168x	106x
8	1.389e-05	3,859x	66x
10	9.727e-06	2,702x	47x
12	7.272e-06	2,020x	35x

The interval falls roughly as `K^-1.3`. Reaching the leader's `3.6e-09` from this direction would take moments in the hundreds, which is not a summary of anything.

Split the `K = 12` bracket by saturation and it lands on top of 14.9's table:

|c| band	neurons	bracket half-width^2	share
[0.0, 0.5)	71	5.660e-05	54.0%
[0.5, 1.0)	61	3.668e-05	30.0%
[1.0, 2.0)	133	8.820e-06	15.8%
[2.0, 3.0)	159	1.080e-07	0.2%
[3.0, 5.0)	541	3.060e-11	0.0%
[5.0, inf)	59	2.096e-15	0.0%

For the saturated three fifths of the population, twelve moments pin the answer to `3e-11` and better — the indeterminacy is not spread across the neurons, it is *entirely* at the kink. Which is the same 66 neurons that carry 97.5% of the Gaussian read-out's error and the same ones a deterministic method's whole budget belongs to. Two different arguments, an impossibility bound and a measured error decomposition, localising to the same quarter of the output.

Now put the two facts side by side. Two moments *determine* the answer only to `2.29e-04`. Two moments *plus the assumption that the law is Gaussian* deliver `9.33e-07`. The assumption is worth a factor of 245 and the moments are worth none of it. Section 10.11's tabulated shape sharpens the same point twice over: `h(u)` with one skewness parameter — call it two and a half moments — reaches `4.69e-08`, which is 4,875 times inside what two moments determine and 936 times inside what *four* moments determine. And in that same table, adding excess kurtosis to `h(u)` instead of skewness bought `1.84x` where skewness bought `18.4x`. A fourth moment is not worth a tenth of a shape parameter.

That is also why twelve exact moments read as atoms (`2.9e-06`) lose to two moments read as a smooth Gaussian (`8.6e-07`): the atoms honour more constraints and assume less, and assuming less is exactly the wrong move here.

Which raises the obvious question of what the *least-committal smooth* density would do, and the answer is that for these laws it does not exist. The maximum entropy density with `K` moment constraints is `exp(-sum_k lam_k u^k)`, and for `K = 4` normalisability requires `lam_4 > 0`, which forces tails lighter than Gaussian. These laws are heavier: the median excess kurtosis is `+0.356` and 99.2% of the 1,024 neurons are leptokurtic. The Newton iteration is driven to `lam_4 -> 0+`, the density stops being normalisable, and the solver — walking the target moments in from the Gaussian in eight steps, precisely so that it cannot fail silently — declines. This is a property of the exponential family, not of the

arithmetic. Maximum entropy is the wrong smooth prior here, and at `K = 2`, where it reduces to the Gaussian and reproduces `9.3252e-07` exactly, it is not a new prior at all.

The family is what fails, not the information. Replace the exponential density by a half-half mixture of two normals with free means *and* free variances — four parameters for four moments, leptokurtic wherever it needs to be, and with a closed form for `E[relu]` rather than a quadrature. The variances have to be free: a two-normal mixture with a *shared* variance is flat-topped or bimodal, hence platykurtic whenever it is symmetric, and can only reach positive excess kurtosis by leaning on a skewness these laws do not have. With that repair the arm solves for every neuron:

```
two-normal mixture, four moments, closed-form read-out
  mse 3.7701e-08     24.73x better than the Gaussian read (9.3252e-07)
                     76.8x  better than the twelve-moment Gauss rule
                     no fit: 0 / 1024
```

Set that beside the impossibility bounds. Four moments *determine* the answer only to `4.39e-05`; four moments read through this mixture reach `3.77e-08`, which is 1,164 times inside the interval those same four moments leave open, and 193 times inside what *twelve* moments leave open. It is also better than section 10.11's best tabulated shape — `h(u)` plus a skewness parameter reached `18.4x` over its own Gaussian baseline where this reaches `24.7x`.

So the two arms of this section are the cleanest statement of the whole subsection: **maximum entropy and this mixture consume exactly the same four numbers, and one of them has no solution for 99.2% of the neurons while the other beats the Gaussian read by a factor of twenty-five.** The moments are not where the accuracy is. The family is.

Where the residual error sits has also moved. Under the Gaussian read the `[1,2)` band carried 59.6% of the error; under the mixture the mass has slid down to the kink itself —

|c| band	neurons	mean squared error	share
[0.0, 0.5)	71	2.586e-07	47.6%
[0.5, 1.0)	61	1.674e-07	26.4%
[1.0, 2.0)	133	5.144e-08	17.7%
[2.0, inf)	759		8.2%

— and in the saturated band the mixture and the Gaussian agree to four figures (`2.8533e-09` against `2.8535e-09`), which is the reference sample's own noise rather than a difference between methods. Where the rectifier is the identity, no shape assumption can be right or wrong.

One caution carries over from 14.9 and applies to this number with full force: `3.77e-08` is what the mixture achieves *given exact moments*, and a sampling method does not have those. Section 14.12 measures what happens when it has to buy them.

14.11 The closure is wrong in its pair block, by a factor of forty

Section 14.8 located the missing information by oracle: give a layer its exact per-neuron marginals through a Gaussian copula and only 12 to 18% of the Gaussian-substitution error goes away, so four fifths of it is joint. That measurement runs the corrupted law all the way to the output. The

complementary measurement stays local, and it is sharper, because every closure in this family is one map applied 32 times:

```
(m_l, C_l)  ->  mu_l = E[relu(p_l)],  S_l = Cov[relu(p_l)]
            ->  m_{l+1} = W^T mu_l,   C_{l+1} = W^T S_l W
```

The second arrow is exact linear algebra — note in passing that it makes the scored quantity the last term of a recursion in which each mean is an exact linear image of the previous layer's *answer*. The first arrow is the entire approximation, and entry `(i,j)` of `S_l` depends only on the joint law of two scalars. So take the true law of `p_l`, fit the Gaussian with its exact mean and covariance, and ask which part of the map the fit gets wrong. `research/paircov.py` does this with 4.2 million rows per arm and, critically, with a control: two independent real samples of the same layer compared to each other by the same formula, so that a small bias cannot be mistaken for a large one. Relative errors, Gaussian-versus-true beside control-versus-true:

k	rectified mean		rectified variance		pair covariance		ratio
	gauss	ctrl	gauss	ctrl	gauss	ctrl	pair
1	2.253e-03	6.855e-04	1.023e-02	1.159e-03	1.024e-01	1.098e-02	9.3
4	2.711e-03	3.875e-04	1.521e-02	1.039e-03	1.126e-01	5.868e-03	19.2
16	2.274e-03	2.735e-04	1.847e-02	1.014e-03	8.052e-02	2.287e-03	35.2
30	1.052e-03	1.294e-04	7.482e-03	7.510e-04	3.240e-02	9.935e-04	32.6

The rectified mean is right to a quarter of a percent. The rectified variances are wrong by one to two percent. The off-diagonal block — the pairs — is wrong by three to eleven percent, nine to thirty-five times the control, and forty times worse than the mean it sits beside. The error is largest in the shallow layers, exactly where 14.9 says the non-Gaussianity is born, and decays with depth as saturation takes over.

Three independent readings now point at the same object: an oracle run inside our own code (14.8), a cost structure read off other participants' telemetry (14.6), and a local decomposition of the one map every closure applies (here). The pair block is where the information is and where the front's `2,021` to `22,046` FLOPs per pair per layer are going.

That closes the deterministic side of the accounting. Stacking the measurements in the order a method would meet them:

```
propagate (m, C) with a Gaussian pair closure, read out Gaussian   4.89e-05
read out Gaussian    from the exact final mean and sigma          9.33e-07
read out a mixture   from the exact final four moments            3.77e-08
read out from the exact final marginal                            ~1e-09  (noise floor)
leaderboard best                                                  3.60e-09
our deployed sampler                                              2.38e-07
```

Both halves are needed, but they stopped being equally binding somewhere in the middle of this section. A Gaussian read-out on exact final moments halts at `9.33e-07`, 260 times short of the leader, which is why the read-out looked like half of the problem for as long as the Gaussian was the only read-

out on the table. The four-moment mixture halts at `3.77e-08` — ten times short. **The read-out is very nearly solved, conditional on the moments being exact.** What is left binding is the propagation, and the condition attached to it is not decoration: 14.12 shows the moments cannot be bought by sampling at any budget, so the only way to collect the mixture's factor of twenty-five is to compute them, which is the deterministic path and no other. That, and not the read-out, is where the four orders of magnitude are — and this section says they have to be spent on pairs. Sections 14.13 and 14.14 measure what a pair correction can actually buy and what state it has to carry to buy it, and section 14.16 measures what that is worth once it is run through the chain — which is where this line of attack ends.

14.12 A shape prior has no variance to give, so it can only spend bias

The read-out floors of 14.10 are ceilings for a method that knows the moments. Ours does not know them; it estimates them from the same rows it would otherwise average. So the question a sampling method actually faces is not which read-out is most accurate given exact moments, but what to compute from `N` rows — and that is settled by giving five estimators the identical rows and looking. `research/momentnoise.py` runs

```
  mean     sample mean of relu(p)                                  -- deployed
  gauss    sample (m, sigma), Gaussian read-out
  mix4     sample (m, sigma, skew, exkurt), mixture read-out
  mixX     sample (m, sigma); skew and exkurt exact, per neuron
  mixU     sample (m, sigma); skew and exkurt fixed at the field medians
```

paired on the same draws, over four networks and six repetitions, at budgets from a thousand rows to a million. Mean squared error against the bundle labels:

rows	mean	gauss	mix4	mixX	mixU
1,024	6.716e-05	6.793e-05	6.718e-05	6.710e-05	6.808e-05
4,096	1.297e-05	1.376e-05	1.299e-05	1.297e-05	1.386e-05
16,384	2.158e-06	3.061e-06	2.203e-06	2.197e-06	3.183e-06
65,536	7.324e-07	1.680e-06	7.784e-07	7.759e-07	1.810e-06
262,144	1.645e-07	1.110e-06	2.037e-07	2.012e-07	1.236e-06
1,048,576	4.802e-08	9.821e-07	8.472e-08	8.457e-08	1.108e-06

The plain average wins at every budget, and it is not close above sixteen thousand rows. But the interesting thing is not the ranking, it is that every cell of this table is one subtraction away from being predicted in advance. Take the differences against the winning column:

rows	gauss − mean	mixX − mean
1,024	+7.70e-07	-5.70e-08
4,096	+7.90e-07	+5.00e-09
16,384	+9.03e-07	+3.90e-08
65,536	+9.47e-07	+4.36e-08
262,144	+9.46e-07	+3.66e-08
1,048,576	+9.34e-07	+3.66e-08

Those columns converge to `9.34e-07` and `3.66e-08`. The exact-moment floors measured independently in 14.10, on 134 million reference rows and by a different program, are `9.3252e-07` and `3.7701e-08`. **The plug-in read-outs have the same variance as the direct average, to three figures, and their entire excess is their own bias.** That is not an approximation of the table, it is the table: `MSE(read-out) = MSE(mean) + floor`, at every budget where the estimator has settled.

The reason is homogeneity again. `relu(sigma*u + m) = sigma*relu(u + c)`, so the plug-in functional `m*Phi(c) + sigma*phi(c)` is, to first order in the sampling error, the same linear statistic of the same rows as the direct average — `dF/dm = Phi(c)` and `dF/dsigma = phi(c)`, and at the median `|c| = 3.49` the first is one and the second is `9e-04`. The estimator is the sample mean wearing a different hat. It cannot be quieter, and so a shape assumption has nothing to trade: whatever bias it introduces is added, undiscounted, to an error the direct average was already paying in full.

Three smaller readings sit inside the same table.

`mix4` and `mixX` agree to two parts in a thousand at every budget — the skewness and kurtosis estimated from the very same rows are as good as the ones measured on 134 million. Estimating the shape is free. It is *having* a shape that costs.

`mixU`, which fixes the shape at the field-wide medians (`skew +0.0357`, `excess kurtosis +0.3559`) and so is the one arm here that would cost a deployed method nothing at all, is worse than `gauss` at every budget above sixteen thousand rows. Its implied floor is `1.06e-06`, above the Gaussian's `9.33e-07`. **Assuming the median shape is a worse assumption than assuming no shape.** Section 10.11 saw the same thing from the other direction, where a tabulated universal `h(u)` bought `1.745x` and the same table with a per-neuron skewness bought `18.4x`; taken together the two measurements say the shape varies across neurons by more than it differs from the normal on average, which is exactly the condition under which a universal correction is worse than none. The mixture fit itself declined on 0.3% of neuron-budget pairs at a thousand rows and on none above that, so the fallback is not doing any of this work.

And the direct average falls as `4.26x` per `4x` of budget over the five steps, which is `1/N` inside what twenty-four paired replicates can resolve. There is no regime here in which it is doing anything other than averaging.

This closes, with measurements rather than argument, the family of levers that section 14.9 could only rule out by inspection. It also sharpens what 14.10's `3.77e-08` is worth and to whom. A twenty-five-fold better read-out exists, it is closed-form, it costs two exponentials, two error functions and a handful of multiplications per neuron, and **no sampling method can reach it**, because the moments it consumes are exactly as expensive as the answer it produces. It is available only to a method that obtains its moments by computing them. That is the sharpest statement this write-up can make about why the front of the field is deterministic: not that sampling is inefficient — 14.1 through 14.4 put our sampler at 90.4% of the ceiling for all samplers, `1.731x` against a bound of `1.915x`, so at most `1.11x` remains on that axis — but that sampling forecloses the read-out. The read-out that sampling cannot have is worth twenty-five times what the entire remaining margin of the sampler is worth.

14.13 Two thirds of the pair error is a table, and the table is the same for every network

An error can be large and still be irreducible, so 14.11's finding — the pair block wrong by three to eleven percent against a control of one — does not by itself justify spending a Phase-2 budget there. What justifies it is knowing the error is *predictable from things the closure already holds*.

Homogeneity says exactly which things those could be. Since `relu(sigma*u + m) = sigma*relu(u + c)`,

```
Cov[relu(p_i), relu(p_j)]  =  sigma_i sigma_j F(law of (u_i,u_j); c_i, c_j)
```

with `u` the pair standardised to zero mean and unit variance. The Gaussian closure evaluates that same functional at the normal law of correlation `rho`, so its answer is a function of `(c_i, c_j, rho)` and of nothing else, and the dimensionless residual

```
t_ij  =  (S_ij - S^gauss_ij) / (sigma_i sigma_j)
```

measures how far the true standardised pair law is from the normal one. If pairs that agree on `(c_i, c_j, rho)` have the same residual, then `t` is a function of three scalars, a grid closes the pair block, and the front's `2,021` to `22,046` FLOPs per pair per layer is that grid being read.

`research/pairtable.py` tests it as a variance decomposition against the same control as 14.11: bin the pairs, and credit a lookup only with what it removes beyond what the identical lookup removes from pure sampling noise. Four networks, five depths, `32,640` pairs a layer, `652,800` pooled. The residual itself has rms `2.4e-03` at layer 1 rising to `1.7e-02` at layer 30, against a control of `2.6e-04` to `8.7e-04`. Refining the grid:

```
bins/axis    cells    removed     of which noise      net
    2            8     26.61%          0.34%       +26.27%
    3           27     35.83%          0.56%       +35.27%
    4           60     43.18%          0.71%       +42.47%
    6          196     54.48%          1.05%       +53.43%
    8          455     61.41%          1.50%       +59.91%
   12        1,452     67.31%          2.36%       +64.95%
   16        3,294     68.77%          3.46%       +65.31%
   20        6,248     69.68%          4.02%       +65.66%
   28       16,042     71.30%          8.68%       +62.63%
```

It plateaus. Going from twelve bins an axis to twenty buys seven tenths of a percentage point and going past that buys nothing the control does not eat. So the answer is not "yes" and not "no": **about 65% of the pair-block error is a function of `(c_i, c_j, rho)` and about 35% is not.** A perfect three-argument table shrinks the residual's rms by `1 / sqrt(0.35) = 1.7x`.

Two further keys settle what kind of table it is.

```
extra key            cells    removed    of which noise     net      change
(none, 12 bins)      1,452     67.31%         2.36%       +64.95%     ---
+ layer index        5,934     73.50%         4.20%       +69.30%    +4.35 pt
+ network identity   5,638     69.00%        22.39%       +46.61%     none
```

Adding the layer index buys four points, so the map drifts slowly with depth — consistent with 14.9, where the non-Gaussianity is created at the first rectifier and then only the saturation moves. Adding the network identity buys nothing: its apparent six-point gain is entirely the noise column, which

quadruples because each cell now holds a quarter as many pairs. **The pair map is a property of this architecture, not of any particular draw of its weights** — which is the condition a Phase-2 submission has to satisfy, since it will be scored on networks it has never seen.

Layer 1 is the one place the table underperforms, at `+24.43%` against `+57%` to `+66%` for layers 4 through 30. That is the layer immediately after the first rectifier, where 14.9 measured the non-Gaussianity being born; the shallowest map is the least like the others and would want its own cells.

So the first `1.7x` of a Phase-2 pair correction is a three-dimensional lookup that costs a handful of FLOPs per pair, transfers across networks, and needs about twelve cells an axis — which is a smaller object than the `20x20` to `66x66` grids the front's ledger implies, and therefore not the whole of what they are doing. The remaining 35% is the interesting part, and the next section asks what it is made of.

14.14 The missing third is one joint third moment, and it is nearly rank one

Section 14.13 left 35% of the pair-block error outside a three-argument table. That remainder is information the Gaussian state does not hold, so the question is which statistic holds it. `research/pairfeat.py` collects the natural candidates alongside the residual on the same `652,800` pairs and tests each as an additional binning key, against two controls: the sampling-noise residual, and a *random* key of identical cardinality, which prices the cell-count inflation that any extra axis buys for free.

extra key	cells	removed	noise	net	vs base	vs random
(none, 12 bins)	1,452	64.14%	2.04%	+62.10%		
skew of the low-c neuron	4,688	71.21%	5.91%	+65.30%	+3.20p	+3.26p
skew of the high-c neuron	4,543	71.05%	5.58%	+65.47%	+3.37p	+3.43p
both skews	11,760	74.52%	6.73%	+67.79%	+5.69p	+6.29p
both excess kurtoses	17,302	73.74%	8.00%	+65.74%	+3.64p	+4.23p
coskew E[u_lo^2 u_hi]	5,023	84.06%	5.62%	+78.44%	+16.34p	+16.40p
coskew E[u_hi^2 u_lo]	5,083	71.99%	5.54%	+66.46%	+4.36p	+4.42p
both coskews	13,544	85.65%	6.47%	+79.18%	+17.08p	+17.67p
both coskews + a skew	26,673	84.94%	8.15%	+76.78%	+14.68p	+21.41p

(The base figure here is `+62.10%` where 14.13's independent run gave `+64.95%` on the same binning with different seeds; the effects below are five times that discrepancy.)

One number does the work. `E[u_lo^2 u_hi]`**, the genuinely joint third moment with the square on the *less saturated* of the two neurons, moves the table from 62% to 78% on its own.** Every marginal candidate — either skewness, either excess kurtosis, all of them together — is worth three to six points. And the marginal ones are not additive with it: adding a skewness on top of both coskews *lowers* the net figure, because the extra cells cost more in noise than the axis returns. The joint statistic subsumes the marginal ones.

A linear correlation against the post-table residual says the same thing without any binning at all, and therefore without any cell-count artefact:

```
statistic   corr with residual   corr with the noise control
coskew E[u_lo^2 u_hi]   +0.5594          +0.0119
skew, high-c neuron     +0.2431          +0.0093
exkurt, high-c neuron   +0.2170          +0.0088
exkurt, low-c neuron    +0.1387          -0.0028
coskew E[u_hi^2 u_lo]   +0.0607          +0.0082
skew, low-c neuron      -0.0220          -0.0072
```

The asymmetry between the two co-skewnesses — `+0.56` against `+0.06` — is the physical content. Squaring the neuron *nearer its kink* and taking it against the other one's level is what predicts the error; the reverse ordering predicts almost nothing. The closure fails where one coordinate is being rectified and the other is merely riding along, and it is the rectified one whose second power matters.

That names the state a Phase-2 closure has to carry: not `(m, C)` but `(m, C, K)` with `K_ij = E[u_i^2 u_j]`. The obvious objection is cost — a third order object is naively `W^3`, which at `W = 256` is `1.7e07` numbers per layer and dwarfs everything in the ledger. But `K` is not the full third cumulant. It is the slice with the first two indices tied, so it is a `W x W` matrix, the same size as the covariance. And it is very far from a dense one:

```
network / layer   effective rank   rank for 90% / 99% of ||K||_F^2   ||K||_F / ||corr C||_F
   m0  layer  4         6.5              19 /  78                        0.308
   m0  layer 16         2.4               3 /  22                        0.566
   m0  layer 30         1.6               1 /   8                        0.587
   m1  layer  4         5.3              16 /  71                        0.336
   m1  layer 16         3.6               6 /  32                        0.522
   m1  layer 30         1.8               2 /  11                        0.493
```

By the depths that matter `K` is nearly rank one. At layer 30 a single singular vector carries 90% of its Frobenius energy and eight carry 99%. Held in factored form the extra state is `W x r` numbers with `r` in the single digits for most of the network and around twenty at its shallowest, and transforming it through a layer costs `O(W^2 r)` against the `2W^3` per transition the covariance backbone already pays. **The object that carries the missing third of the pair-block error is cheaper to move than the covariance that carries the part we already have.**

Its size is not negligible either — `||K||` is a third of `||C||` in correlation units at layer 4 and rises to nearly six tenths by layer 30 — so this is a first class piece of the state rather than a perturbative correction, and it grows exactly where 14.11 measured the closure getting worse.

Two things this does *not* establish, stated plainly because a Phase-2 plan built on them would fail. First, everything above is a property of the closure *map* evaluated on the true law at each layer; whether `K` can itself be propagated — whether `K_{l+1}` is predictable from `(m_l, C_l, K_l)` with comparable fidelity — is a separate measurement that has not been made. A state that must be re-measured each layer is not a closure. Second, `79%` of the residual variance removed is a `1 / sqrt(0.21) = 2.2x` reduction in the pair block's rms error, and 14.11 puts that block at three to eleven percent against a one percent control; two point two is a real factor and it is not four orders of magnitude. What this section establishes is where the next order of magnitude lives and that it is affordable, not that the ladder has been climbed.

Both caveats were subsequently measured, and both went the wrong way. Section 14.16 converts the `79%` into an answer through the chain and finds it worth `0.12` times our own score, against the `f = 0.99` the chain turns out to need; it also finds that `79%` is a ceiling of the variables rather than of the table, since refining the bins does not move it. The Phase 2 architecture proposed here is withdrawn there, on our own evidence. The `K`-propagation question was never resolved because the oracle made it moot.

14.15 Instrumented share does not see everything, and a second axis is free

On 7 August the organisers stated in thread 18105 that *"submissions that intentionally route computation around flopscope's accounting to obtain unmetered compute … will be disqualified based on final review by ARC"*, which is the disposition of the pattern section 10.12 measured: entries whose instrumented share `F/C` is at or near zero, charging almost their whole run to residual wall time. That screen works on exactly those entries. This section is about a submission it does not see, and it is offered as a note on the metric rather than as an allegation about anybody.

Overnight on 8 August the board moved and submission `325657` took second place at raw `9.0e-09`. Its median per-network ledger, read from the same public telemetry as everything else in section 14.6:

```
metered FLOPs          F  =  6.187e09
effective compute      C  =  7.992e09        C/B = 0.0294  (under the 0.1 clamp)
instrumented share   F/C  =  0.774
wall time                   46.960 s
flopscope backend            0.0023 s
flopscope overhead          46.929 s
residual wall time           0.0179 s
```

Two things follow arithmetically. First, `F/C = 0.774` is comfortably above the `0.60` line, so this entry sits in the "metered" class of our scan of 120 leaderboard rows and is indistinguishable there from the fifty-eight ordinary entries above us. Second, since the charge is `C = F + lambda * (wall - backend - overhead)`, **46.929 of the 46.960 seconds are free by construction** — not through any evasion, but because that is what the formula says. The residual charged is eighteen milliseconds.

The axis that does separate this row is metered FLOPs per second of backend time: `6.187e09 / 0.0023 = 2.73e12`.

We first compared that against a single-core peak, and that was our error. The 3 August release note states the split explicitly — participant code gets one physical core, *the flopscope backend gets seven* — so the right denominator is about `7.2e11`, not the `1.028e11` we measured for one core. Charged FLOPs are also not executed FLOPs: since v0.9.0 the 64-bit class bills at twice the 32-bit rate, so a float64 hot path is charged about double what it computes. Correcting both, this row runs at roughly **380% of a seven-core float32 peak**, or about **190%** if it is float64. Not the 2,659% we first wrote.

Both corrections apply to the whole column, so the ordering is unchanged: every other entry in the scan falls to between 0.07% and 4.7%, the lone outlier at 179% becomes 26%, and the same participant's earlier submission `325324` sits at 0.4%. This row is still an order of magnitude clear of the field, and the change is still between two of their own submissions rather than between them and everyone else. But a factor of four above a plausible hardware ceiling is a much weaker statement than a factor of twenty-

seven, it sits comfortably inside what a shape-formula FLOP count can over-declare, and we would rather say so than leave the larger number standing.

We want to be careful about what that ratio does and does not mean, because flopscope's own source says it can be produced innocently. Reading `_budget.py`, the backend timer accumulates only the duration of the wrapped numerical call, while `deduct()`'s bookkeeping time and any in-block time that is not the backend call are accumulated into overhead. A program that issues a very large number of *small* metered operations therefore lands exactly here: per-call Python overhead in the tens of microseconds, summed over millions of calls, is tens of seconds of overhead, while the kernels themselves are fast and the declared FLOP totals are computed from shapes rather than measured. Charged FLOPs can exceed executed ones for the same reason — the price list of section 14.2 is a formula on shapes and dtypes — and over-declaring is normally a penalty. It is not a penalty here only because `C/B = 0.0294` sits beneath the `max(0.1, C/B)` floor, so three and a half times this cost would score the same.

So we do not claim this row is anything in particular; we cannot see the code and the benign reading is available. What we do claim is narrower and, we think, useful to a review that is about to run: **instrumented share is one number and it is not sufficient.** An entry can charge 77% of its cost to metered FLOPs, be classed as clean, and still have 99.93% of its wall clock outside the charge. The ratio `F / backend_time` is already in the public telemetry, costs nothing to compute, and orders the field very differently from `F/C`. If the private re-evaluation is going to distinguish algorithms from accounting, it will want both.

For our own part the correction applies equally: we are at `F/C = 0.9928`, second highest of the eighty ledgers we could read, and at 4.6% of a seven-core peak rather than the 32% of one core we first reported. That is what a program which actually does its arithmetic looks like, and it sits well under the ceiling rather than near it.

14.16 The pair integral is free, and the pair map still cannot be bought

Section 14.11 ended by pointing at the propagation and saying the four orders of magnitude have to be spent on pairs. Sections 14.13 and 14.14 measured what a pair correction explains. This section measures what a pair correction is *worth*, which is a different number, and it closes the route.

The pair integral has a closed form, and it costs nothing.

Our closure computes `E[relu(X) relu(Y)]` for a correlated Gaussian pair by the one-dimensional quadrature of section 6, 512 nodes per row. That is accurate and it is the wrong algorithm. Write `X = s_i (U + c_i)` and `Y = s_j (V + c_j)` with `(U, V)` standard bivariate normal of correlation `rho`, expand `relu(u + c)` in probabilists' Hermite polynomials, and apply Mehler's formula `E[He_m(U) He_n(V)] = delta_mn n! rho^n`:

```
E[relu(X) relu(Y)]  =  s_i s_j  sum_k  alpha_k(c_i) alpha_k(c_j) rho^k / k!

alpha_0(c) = c Phi(c) + phi(c)
alpha_1(c) = Phi(c)
alpha_k(c) = phi(c) He_{k-2}(-c)              k >= 2
```

Every term is a rank-one matrix in `c` times an elementwise power of the correlation matrix. The coefficients cost one `Phi` and one `phi` per neuron per layer — `W` special-function evaluations where the

quadrature needed `W * NODES * W`. At flopscope's prices (`norm.cdf` 48 FLOPs per element, `norm.pdf` 27):

```
31 layers of quadrature       8.32e10   =  30.6%  of B
31 layers of series, K = 8    3.31e07   =   0.012% of B
31 layers of series, K = 16   6.57e07   =   0.024% of B
```

These are 32-bit-class counts. Our chain runs in float64 for the covariance recursion, and since v0.9.0 the 64-bit class bills at twice the rate, so as written it pays double: the quadrature would be 61% of `B` and the `K = 8` series 0.024%. The ratio between them — the only thing this paragraph is about — is unaffected, and the series is cheap enough that the dtype question never becomes interesting.

a factor of 2,511 per layer at K = 8. This is not an approximation traded for speed. Run to the end of the chain on all eight bundled networks, the series and the quadrature give final-layer errors of `5.8606e-05` and `5.8605e-05` — the same number to five figures — and the series is 35 times faster in wall clock too. Against the quadrature pointwise the series agrees to `6.8e-04` relative at layer 0 and *improves* with depth, to `4.3e-07` at layer 30, because section 14.9's saturation makes `relu` nearly linear where `|c|` is large and a nearly linear function needs only `He_0` and `He_1`.

The consequence for the accounting is that the deterministic chain is no longer anywhere near the budget. With the covariance backbone at `2.08e09`, the whole chain costs `2.15e09`, which is 0.79% of `B` in the 32-bit class and 1.6% as we actually run it, in float64. The multiplier is the `max(0.1, .)` clamp, and there is six times more compute available at exactly zero cost in score — twelve, if the hot path is moved to float32. Cost has stopped being the constraint on this route. Only accuracy is left, and the rest of this section is about accuracy.

How good does the pair map have to be?

Sections 14.13 and 14.14 report fractions of residual *variance* explained. That is a statement about the residual, not about the answer. To convert one into the other, run the chain with the true correction injected at reduced amplitude: a map that explains a fraction `f` of the residual variance leaves amplitude `sqrt(1 - f)`, so inject `1 - sqrt(1 - f)` of the measured non-Gaussianity at every layer, in standardised units, which is the form a table emits. Two anchors say whether the rig is measuring anything. At `f = 0` it must reproduce the pure closure, and at `f = 1` it must reproduce a Gaussian read-out of the exact final moments, which section 14.9 measured independently at `9.33e-07` through completely different code. It gives `9.1099e-07`, a 2.4% agreement. The rig's own noise floor, with the correction sampled from 1,048,576 rows on four networks, is `1.09e-08`, far below everything it is being asked to resolve.

```
pair map quality        final mse    score at the 0.1 clamp    vs our sampler
pure closure            7.9656e-05         7.97e-06                0.03x
f = 0.65                2.8798e-05         2.88e-06                0.07x
f = 0.79                1.7728e-05         1.77e-06                0.12x
f = 0.95                4.9780e-06         4.98e-07                0.42x
f = 0.99                1.7503e-06         1.75e-07                1.20x
perfect propagation     9.1099e-07         9.11e-08                2.30x
```

The pair map has to be right to about 99% of its residual variance before the deterministic chain matches the sampler we already have. Section 14.13's 65% lands at 0.07

times our own score. Section 14.14's 79% lands at 0.12. The curve is brutally nonlinear because the error compounds through thirty-one layers, and the interesting region is entirely in the last percent.

A real table does worse than the idealisation.

Both arms of that oracle are guesses about what a table leaves behind, so we built one. Bake a `12 x 12 x 12` table on `(c_lo, c_hi, rho)` — `c`-ordered so it is a function — from four networks, with an empty-cell backoff to the `(c_lo, c_hi)` marginal, and run the chain on the other four, looking the correction up with the *chain's own* keys because that is the deployment condition:

```
pure closure                  3.7555e-05     1.00x
table, pair correction only   5.5629e-05     0.68x    worse than nothing
table, mean correction only   2.0054e-05     1.87x
table, both                   2.0882e-05     1.80x
```

The pair table, which is the object sections 14.13 and 14.14 were about, is *harmful* in the chain. The cheap one-argument mean table carries what gain there is, and 1.87 times is against the 18 times needed for parity. Explaining variance and helping a recursion are not the same thing: what survives a conditional-mean lookup is 35% of the residual with all of its structure intact, and a binned lookup is a full-rank perturbation of a covariance whose true residual is low-rank, so it damages the spectrum it is trying to repair. Our first synthetic control failed for the same reason in a cruder way — scrambling the residual drove the second-moment matrix `3.6e-04` outside the positive semidefinite cone and measured the violation rather than the lever.

And 79% is the ceiling of the state, not of the table.

The obvious rescue is a better estimator on the same variables. It does not exist. Pooling all 652,800 pairs the way section 14.14 did and holding out 35% of them:

```
keys                        in-sample   held out   net of control
(c_lo, c_hi, rho)  12^3        0.6724     0.6667       0.6152
+ both co-skews    4 x 4       0.8506     0.8258       0.7886
+ both co-skews    8 x 8       0.8711     0.8067       0.7856
finer, 16^3 + 6 x 6            0.8505     0.7607       0.7379
+ layer + co-skews             0.8521     0.7927       0.7486
```

Section 14.14's 79% reproduces exactly and survives the held-out split, so that measurement stands. What is new is that refining it does not help: eight bins of co-skew instead of four is no better, and sixteen bins of `c` is worse. The estimator is not what is limiting. **The variables `(c, rho, K)` determine about 79% of the pair residual and that is a property of the state, against a requirement of 99%.**

So the state has to grow, and the one candidate does not come for free.

Section 14.14 proposed carrying `K` as state. Whether that is even necessary has an answer now. Fit a Gaussian to a layer's true `(m, C)`, push it through the same rectifier and the same weight matrix, and compare the third moment it implies at the next layer against the truth, with an independent real sample of the same layer as the control:

```
 layer       zero      Gaussian closure      control      captured
 1 ->  2   1.0000          0.5853             0.0567        41.5%
 4 ->  5   1.0000          0.8197             0.0264        18.0%
 8 ->  9   1.0000          0.8797             0.0154        12.0%
16 -> 17   1.0000          0.9314             0.0108         6.9%
24 -> 25   1.0000          0.9579             0.0081         4.2%
30 -> 31   1.0000          0.9862             0.0065         1.4%
```

A covariance-only closure recovers 1.4% of the third moment at the last layer, against a control eight times smaller than the gap, so the measurement resolves. `K` is not derivable from `(m, C)`; it would have to be carried. We did not go on to measure whether `K` propagates from `(m_l, C_l, K_l)`, because the oracle above had already made the answer irrelevant: a state that delivers 79% delivers 0.12 times our own score, and the question of how to transport it does not arise.

The verdict, and what it says about the front.

Three independent measurements agree. The oracle says the pair map needs `f = 0.99`; the held-out fit says the state tops out at `f = 0.79`; the built table says the realised gain is 1.87 times against 18. The deterministic route with state `(m, C)` or `(m, C, K)` cannot reach our own sampler, let alone the front. Section 14.14's Phase 2 architecture is withdrawn on our own evidence.

That also sharpens section 14.6. Perfect propagation with the four-moment mixture read-out of section 14.10 would score `3.77e-09`, and the leader scores `4.00e-10`. **Even a flawless moment-propagation method, with the best read-out we have measured, is a factor of nine short of the front.** Whatever the leaders are doing, it is not this. The arithmetic points at the one line in section 14.11's stack we never reached — reading out from the exact final *marginal*, at `~1e-09` — and the marginal that matters is one-dimensional: the last layer only ever asks for `E[relu(w_j . a_30)]` along 256 fixed directions, not for a joint law over 256 coordinates. Whether that reduction is the front's lever we do not know. It is the only door in this section we did not measure shut.

One loose end we are reporting rather than resolving. Running the Hermite series *without* the exact closed-form diagonal — that is, letting the truncation quietly shrink the propagated variance — beats the exact chain on all eight networks by a consistent factor near 2.3. A uniform variance-shrink knob does not reproduce it: the best shared value gives 1.49 times, the per-network optima disagree, and the chain diverges to NaN past a 3% shrink. So the effect is shaped in `c` rather than uniform, which is consistent with section 14.9's finding that the error lives at the kink, and we have not identified it. We are not claiming a 2.3-times method; we are recording an unexplained factor of 2.3 that a reader with more time should look at.

14.17 The front cannot be running a forward method

Every section above measures our own estimator. This one measures everybody else's, because the grading ledger is public and we had not read it. What it says is sharper than anything we could have got from the leaderboard: the four submissions ahead of the field are not doing a better version of what we do, and the reason is structural rather than numerical.

The ledger. The submission detail page carries, unauthenticated, the whole `evaluation.results` object — not just the aggregate. Inside it, `results.public_scores` holds, for each of the fifty public MLPs, the `final_layer_mse` and a **thirty-two entry** `per_layer_mse` **vector**, and `results.per_mlp` holds the

matching per-MLP telemetry (FLOPs, backend seconds, residual seconds). Scoring's own comment fixes the semantics — `per_layer_mse[-1] == final_layer_mse` and `mean(per_layer_mse) == all_layers_mse` — so the vector is exactly what each participant emitted at each depth, scored against the same ground truth we are scored against. Section 15.1 discloses the endpoint; this section is what is in it. We pulled forty entries.

The multiplier is not what separates the front. `mean_effective_compute` makes the accounting explicit: the budget `B = 2.72e11` is *per MLP*, and the leader spends `2.6e10` of it, a utilisation of `0.0959`, which is below the `max(0.1, C/B)` clamp. That sounds like an advantage and it is not one. For any estimator whose error falls as `k/F`, the score `k/F * max(0.1, F/B)` equals `k/B` for every `F` at or above `0.1B` and *rises* below it, so the multiplier floor is flat and sitting on it buys nothing (section 14.6). Their raw final-layer MSE is `3.63e-09` against our `2.38e-07`, at a tenth of our FLOPs. The gap is accuracy, not accounting.

Their accuracy does not depend on their budget. Walking the submission ids around the front recovers earlier gradings of the same participants, and the same method appears at two very different budgets:

participant	raw final MSE	FLOPs	what changed
dpskv5	3.555e-09	1.765e11	
dpskv5	3.628e-09	2.430e10	7.3x less compute, 2% worse
joe_wanza	4.492e-09	1.765e11	
joe_wanza	3.954e-09	6.658e10	2.6x less compute, 12% BETTER
huang_chung_yi	1.051e-07	1.522e11	
huang_chung_yi	9.022e-09	8.233e09	18x less compute, 12x better

A `1/n` sampler loses a factor of 7.3 when it gives up a factor of 7.3 of compute. These lose two percent. Whatever produces `3.6e-09` is at a **bias floor**, not a variance floor, and the trailing budget was being spent on something that was contributing almost nothing. Note also that two different participants submit at *exactly* `1.765e11` FLOPs, to four figures, and that each participant's raw MSE repeats to four figures across resubmissions: the computation is deterministic and at least two of them share a configuration.

Nobody on the board can produce an intermediate layer to the accuracy the front produces the last one. This is the finding. Across all forty entries, the best `per_layer_mse` achieved at any depth `1..30` by anyone is `1.71e-08`, and that submission's final layer is `1.70e-07` — a ratio of one, which is what a forward method looks like. Ours is the same shape (best intermediate `6.38e-08`, final `4.68e-08`). The front is not:

sub	score	final MSE	layer 30	layer 31	L30/L31	
324969	3.63e-10	3.63e-09	2.278e-01	2.06e-09	1.1e+08	(zero-filled)
325657	9.02e-10	9.02e-09	2.195e-05	6.46e-09	3.4e+03	
325329	9.65e-10	3.95e-09	9.236e-07	2.69e-09	3.4e+02	
325006	1.17e-07	2.09e-07	1.738e-07	1.26e-07	1.4	(forward)
324409	2.09e-07	2.38e-07	6.38e-08	4.68e-08	~1	(ours, forward)

An injection experiment fixes what that has to mean. Feed a known error into `E[a_30]` on the eight bundled networks and read the error it produces at layer 31: over injected MSEs from `9.7e-07` down to `1.0e-12`, the amplification is `1.01`, `0.99`, `1.02`, `1.05` across four decades of injected magnitude, because He initialisation doubles the norm (`||W^T v||^2 = 2||v||^2`) and the last layer's gate rate `E[Phi(c)] = 0.495` halves it again. Concretely, 35% of layer-31 neurons sit at `c > 2`, where the rectifier is linear with probability `0.977` and `E[relu(p_j)]` is `w_j . E[a_30]` to that accuracy, so their error is the layer-30 error. A method whose layer-30 mean is good to `2.2e-05` cannot deliver a layer-31 mean at `6.5e-09` by pushing that mean through one more layer. **The front's last layer is not computed from the front's layer thirty.**

Two readings survive. Either they hold an internal state at depth 30 that is accurate to `1e-09` and choose to emit something cheap for the unscored intermediate layers — possible, since `all_layers_mse` is only the secondary score, though emitting an object you already hold is free — or they compute `E[relu(w_j . a_30)]` by a route that never forms `E[a_30]`. We cannot distinguish these from outside. What the ledger does settle is that **no forward chain on the board, ours included, gets within two orders of magnitude of the front at any depth**, and every method whose write-up is public is a forward chain.

Every published method is in the field, not at the front. Four participants have posted methods with graded submission ids, so their positions are checkable rather than claimed:

method	raw final MSE	who / where
cumulant propagation, trajectory-calibrated	5.89e-06	jamesrahenry, 18097
randomly-shifted lattice + exact layer 1	~9.8e-07	evaaaz, 18053
our Gaussian closure (section 14.16)	5.86e-05	this document
kprop k_max = 3 at (256, 32)	4.74e-06	our run of arXiv 2605.05179
structure-aware hybrid, Sobol + sparsity	2.18e-07	natasha_stewart, 18106
ours, whitened lattice sampler	2.38e-07	324409
---		
the front	3.63e-09	undisclosed

The organisers' own reference algorithm is in that table: we ran `mlp_cumulant_propagation` at the grading shape and `k_max = 3` costs `3.61e11` FLOPs — above the whole budget — to reach `4.74e-06`, which is a thousand times short of the front at fifteen times its cost. The companion paper's width exponent collapses at depth 32 and our measurement is consistent with it. Cumulant propagation is not what is happening at the top.

The two independent negative results in that list are worth stating separately, because they close doors we would otherwise have spent Phase 2 on. evaaaz measured linearisation and exact-mean control variates at `1.0-1.2x`, scrambled Sobol' tied with a lattice per row, and a learned bias-corrector on a deterministic block-covariance estimate at `2.3x` against a required `17x` — concluding that what remains is irreducible kink variance. jamesrahenry measured the analytic close at the last layer (radiant-allomancer's method, 18085) *inside another participant's sampler* and found it **1.35 times worse** than the plain sample mean, with third- and fourth-cumulant Edgeworth terms clawing back only to parity. Our own delta-method arithmetic agrees: the rectified-Gaussian plug-in has variance `sigma^2/n * [Phi(c)^2 + phi(c)^2/2]` against the sample mean's `Var(relu(p))/n`, a ratio of `1.30` at `c = 0` and `1.003` at `c = 3`. The analytic close is not a lever, and three independent measurements now say so.

A correction to our own first reading. We initially took the layer-30 to layer-31 drop as the front's signature. It is not, by itself: of the forty entries, twenty-seven fill layers 0-30 with a constant whose MSE equals `E[alm_l^2]` exactly — they are emitting zeros for the unscored layers — and participants ranked below us do this too, so the drop alone predicts nothing about score. The finding above survives that correction because it is not the drop: it is the *census*, that the best intermediate accuracy anyone has ever produced is `1.71e-08` while four participants produce final layers between `2.1e-09` and `9.9e-09`.

What this leaves for Phase 2. Section 14.16 closed with one door: reading out from the exact final marginal, at about `1e-09`, and the observation that the marginal that matters is one-dimensional — the last layer only ever asks for `E[relu(w_j . a_30)]` along 256 fixed directions, never for a joint law over 256 coordinates. The ledger now says the front is behind that door and not behind any of the others, and it adds a constraint we did not have: whatever is behind it produces the answer for layer 31 without producing the answer for layer 30 at the same quality, and it plateaus at `3.6e-09` rather than improving with compute. We do not know what it is. We now know a great deal about what it is not.

14.18 The read-out is capped at 1.5e-08, and the cap is the moments

Section 14.16's oracle ran the chain with a Gaussian read-out at the end, so its "perfect propagation" arm (`9.11e-07`) conflates two different errors: what the chain got wrong about `(m_30, C_30)`, and what a Gaussian assumption gets wrong about the law of `p_31`. Section 14.17 showed the front is `3.63e-09`. This section separates the two errors and finds that the second one alone already rules out every moment-based method, ours included.

Hand the read-out the truth. Expand the rectifier in probabilists' Hermite polynomials about each neuron's own location and scale,

```
relu(s (U + c))  =  s  sum_k  alpha_k(c) He_k(U) / k!
E[relu]          =  s  sum_k  alpha_k(c) E[He_k(U)] / k!
```

with the same `alpha_k` as section 14.16, and measure `E[He_k(U)]` for `k <= 8` from `2^26` forward passes per network, in the ground truth's own arithmetic (float32 forward, float64 accumulation). `U` is standardised, so `E[He_1] = E[He_2] = 0` and truncating at `k = 0` *is* the Gaussian read-out. There is no propagation error anywhere in this rig: every moment is the true one. Eight networks:

```
truncation      final-layer MSE     vs Gaussian
k <= 0            9.0063e-07           1.00x      (Gaussian read-out)
k <= 2            8.6235e-07           1.04x
k <= 3            1.2253e-07           7.35x
k <= 4            1.4674e-08          61.38x      <- floor
k <= 5            2.0972e-08          42.94x
k <= 6            2.5867e-08          34.82x
k <= 8            3.6805e-08          24.47x
plain sample mean on the same draws   3.5328e-10  (the rig's noise floor)
```

Two anchors say the rig is sound. The `k = 0` row reproduces section 14.9's `9.33e-07`, measured independently through different code, to 3.5%. And the noise floor is `3.5e-10`, forty-two times below the number being reported, so the floor at `k = 4` is a property of the expansion and not of the sample.

The series stops helping at fourth order and then hurts. That is the signature of an asymptotic rather than convergent expansion, which an Edgeworth series is; adding true information makes the answer worse. The best read-out we have ever measured is therefore `1.47e-08`, which is 2.6 times better than section 14.10's four-moment mixture (`3.77e-08`) and 61 times better than the Gaussian close, and it is obtained by *giving the estimator the exact answer to every moment it asks for*.

The front is four times below that. `dpskv5` grades at `3.63e-09` raw. A method that propagated `(m, C)` with zero error and then read out with the best moment-based rule in this document would score `1.47e-08` and lose. So the front's read-out is not moment-based. Section 14.11 put "read out from the exact final marginal" at about `1e-09` and section 14.16 called it the one door it had not measured shut; that estimate stands for the *marginal*, but the moment route to the marginal is now measured shut at `1.47e-08`. The distinction matters because it names the state: what has to be carried and refined is the one-dimensional law of each `p_l,i` — a density, or its characteristic function — and not any finite list of its moments.

This also finishes off the Phase-1 version of the idea, for the record. Using the ladder with moments estimated from the *same* `n` samples that the plain mean uses buys nothing, because the moment estimates carry the same `1/n` noise: jamesrahenry measured `1.007x` for exactly this substitution inside another participant's sampler (section 14.17), and our own delta-method arithmetic gives `1.30x` at `c = 0` falling to `1.003x` at `c = 3`. The ladder is a statement about what an *exact* state is worth, not a sampler improvement.

14.19 The dependence that matters is not low-rank, and it is half the spectrum

Section 14.18 said the read-out cannot be bought with moments. This section asks what it *can* be bought with, and the answer is a hard lower bound on the state any deterministic method has to carry.

A permutation oracle. Rotate `a_30` into the eigenbasis of its own covariance and shuffle each eigencoordinate independently across samples. That preserves every one-dimensional marginal *exactly* and destroys every dependence the covariance does not already record, so

```
E_shuffled[relu(w_j . a_30)]  -  E_true[relu(w_j . a_30)]
```

is precisely the error an estimator makes if it carries those marginals and assumes independence. Two independent shuffles give the control. Extending it, leave the top `r` eigencoordinates jointly intact and shuffle the remaining `256 - r`: that is the error of carrying an exact `r`-dimensional joint plus marginals for the rest. Three networks, `2^20` draws:

joint dims kept	independence bias	control
0	9.2643e-07	1.78e-10
2	7.8196e-07	1.70e-10
4	6.4735e-07	1.69e-10
8	3.5702e-07	1.27e-10
16	1.9498e-07	7.59e-11
32	5.7415e-08	2.88e-11
64	8.3778e-09	1.06e-11
128	4.4273e-10	8.47e-13

The control sits five hundred times below the bias at every row, so the whole curve resolves.

Marginals are worth nothing. At `r = 0` — all 256 marginals exactly known, independence assumed — the answer is `1.01e-06`, which is the Gaussian read-out of section 14.18 (`9.01e-07`) to within the difference in sample size. The reason is immediate once seen: under independence `p_31,j = sum_k beta_jk z_k` is a sum of 256 independent terms and the central limit theorem erases the marginals. **Every bit of the 61-fold gain that fourth-order Hermite bought in section 14.18 comes from dependence between coordinates, not from the shape of any coordinate.** A method that propagates one-dimensional laws — densities, characteristic functions, however exactly — is propagating something the last layer cannot use.

And the dependence is not concentrated. The pre-activation covariance's participation ratio contracts from 128 to about 5 over the thirty-two layers; we measured a rank-5 backbone in section 7.2 and jamesrahenry reports the same contraction independently. It is tempting, and wrong, to read that as "the dependence lives in five directions". Keeping the top 8 jointly still leaves `3.6e-07`. Keeping the top 64 leaves `8.4e-09`, which is still above the front's `3.63e-09`. **You have to keep about 128 of the 256 directions — half the spectrum — before the residual falls to `4.4e-10`.** The variance is low-rank; the dependence that determines a rectified expectation is not, because the kink is a threshold and a threshold is sensitive to directions carrying almost no variance.

What this bounds. Put the three oracles together. A deterministic method that wants the front's `3.63e-09` needs (i) propagation of `(m, C)` good enough that the layer-30 term is below that, since amplification is `1.01` (section 14.17), (ii) a read-out that is not moment-based, since the exact-moment ceiling is `1.47e-08` (section 14.18), and (iii) a joint object of dimension at least about 64, since marginals plus a small block are not enough (this section). Our `(m, C)` state fails (i), our four-moment mixture and Hermite ladder fail (ii), and the low-rank hybrid we had planned for Phase 2 fails (iii). Section 14.16 withdrew that architecture on the strength of a binned-table measurement; this section withdraws it again, from the other side and without any estimator in the loop.

We do not know what satisfies all three at `2.43e10` FLOPs. We now know the shape of the thing that does.

14.20 Four more doors, all closed: maximum entropy, the moment problem, a learned read-out, and the target's own noise

Section 14.18 left one question open. Its ladder is an Edgeworth truncation, which is asymptotic, so its 1.47e-08 floor could be the *information* in four moments or merely the *shape* of that particular series. Four experiments settle it, and a fifth checks whether the target itself imposes a floor.

Maximum entropy is not available. The natural family-change is the maximum-entropy density `exp(l0 + l1 u + l2 u^2 + l3 u^3 + l4 u^4)` with `l4 < 0`, which is a genuine positive density everywhere — unlike the Edgeworth expansion, which goes negative in exactly the tails where the rectifier's kink sits for the `c < 0` neurons. A vectorised Newton solve over all 256 neurons refuses to converge, and the reason is structural rather than numerical. A quartic exponential with a negative leading coefficient has *lighter* tails than a Gaussian, so it cannot represent excess kurtosis above zero, and

statistic	0%	5%	25%	50%	75%	95%	100%
skew `g1`	-0.887	-0.558	-0.440	-0.017	0.440	0.549	0.767
excess kurtosis `g2`	-0.056	0.126	0.261	0.331	0.434	0.685	1.759
`c = m/s`	-5.718	-4.973	-3.781	-0.178	3.588	4.826	5.654

99.1% of layer-31 neurons are leptokurtic. The four-moment maximum-entropy problem is infeasible for essentially every neuron in the network. (The number the solver prints under those conditions, 8.23e-07, is a non-converged residual and means nothing; we report it only so a reader who reruns `research/maxent.py` is not misled by it.)

The distribution-free bound is vacuous. If families are going to be tried one at a time this becomes a war of attrition, so we asked the family-free question instead. Over *all* probability measures matching `mu_0..mu_K` exactly,

```
L = min E[relu(s(u + c))],    U = max E[relu(s(u + c))]
```

is a linear program in the measure, and `(U - L)/2` is the irreducible uncertainty of any method whose state is `K` moments. Discretising `u` on `|u| <= 20` restricts the feasible set, so the computed bracket is a *lower* bound on the true one. On 96 neurons across two networks:

K	worst-case MSE	rms half-width	median half-width
4	3.87e-05	6.22e-03	5.57e-04
6	1.91e-05	4.37e-03	7.08e-05

The four-moment bracket is 6.2e-03 wide — forty times *worse* than the Gaussian read-out that uses only two moments. The worst case is dominated by adversarial measures that hide a vanishing mass far out in the tail where the rectifier is large, and it says nothing about the smooth, near-Gaussian laws this network actually produces. Worst-case moment analysis cannot answer the question. We record it because the negative is worth knowing: on this problem the classical moment-problem bound is not a usable ceiling, and a reader who reaches for it first will conclude nothing. (`research/momentlp.py`.)

A learned read-out on exact moments finds nothing. What is left is the empirical version. The Edgeworth truncation error is not noise: it is a smooth deterministic functional of the standardised law, so it should be learnable. We computed exact standardised moments to order eight, formed the normalised truncation residual `y = t/s - edgeworth/s`, and regressed it on `(c, g3, g4, g5, g6, g7, g8, Phi(c), phi(c))` with polynomial features of degree 2 to 4 and ridge across four regularisation strengths. Held out **by network**, never by neuron — neurons inside one MLP share a layer-30 law and would leak. Eight networks, leave-one-network-out, 2,048 rows.

features	best held-out MSE	vs `edge4`	held-out `R^2`	in-sample
degree 2	1.749e-08	1.05x	+0.045	1.30e-08
degree 3	1.747e-08	1.05x	+0.047	1.30e-08
degree 4	1.782e-08	1.03x	+0.028	1.35e-08

Baselines on the same eight networks: Gaussian read-out 8.66e-07, `edge4` 1.83e-08. The learned correction buys **5%**, and — the decisive line — even *in sample* it buys only 1.4x. Eight exact moments do not explain the Edgeworth residual on the training set, let alone off it. This is a different question from 7.21, which regressed the closure's *propagation* residual on features the closure computes and also found nothing; here the features are the exact moments of the true final-layer law and the target is a pure read-out error.

So the read-out cap is the information, not the shape, and the moment route is shut from three independent directions: the series bottoms out at order 4 (14.18), no proper four-moment density family beats it (mixtures at 3.77e-08 in 14.10, maximum entropy infeasible here), and a learned map on eight exact moments adds 5%.

The target is not the floor. One more possibility had to be eliminated before concluding that lower scores exist at all: the ground truth is itself a Monte Carlo estimate, and an estimator's error against it is `|e|^2 + |g|^2` where `g` is the ground truth's own sampling noise — the two are independent, so `|g|^2` is a floor nobody can go under. The contest CLI's `contest_spec` sets `ground_truth_samples = 100 * 100 * 256 = 2,560,000`, and our measured `avg_variance` over four validation networks is 0.0505 (spread 0.0155 to 0.0976), which would put that floor at **2.0e-08** — *above* the front's 3.63e-09, and above our own 2.09e-07 in a way that is arithmetically impossible. Something had to give, and it is the assumption that the contest's baked dataset uses the CLI default. Run the inference the other way: the front's `best_mlp_adjusted_final_layer_score` is 1.28e-10, which at the floored multiplier is an MSE of **1.28e-09** on its best network, so

```
|g|^2 <= 1.28e-09    =>    n_ground_truth >= 3.3e7
```

The contest ground truth is drawn with at least thirty-three million samples, an order of magnitude beyond the CLI default. Two things follow. The target imposes no binding floor at 3.63e-09 — **there is still room below the front**, and any claim that the leaderboard has saturated against ground-truth noise is false. And the challenge-level constant `sampling_mse = 6.46947e-07`, identical across all forty ledger entries, is not the ground truth's noise either: it is the MSE a plain Monte Carlo estimator achieves spending the entire budget. Dividing it into `avg_variance` gives 64,715 samples, and `2.72e11 / 4.203e6 = 64,715` — an exact match, and an implied `avg_variance` of 0.0419 sitting inside our measured spread. It is the organisers' baseline, not a floor, which is why every serious submission is below it.

For completeness on the seed: `seeds.py` derives the estimator's seed as `int(SeedSequence(input_seed).spawn(3)[2].generate_state(1)[0])` — a bare `uint32`, not the `SeedSequence` object, so the parent entropy is not readable through `rng.bit_generator.seed_seq`. And `input_seed` is `secrets.randbits(63)`, so inverting the 32-bit output to recover the 63-bit parent and then spawning the ground-truth substream is not a search anyone can run. There is no seed route to reproducing `g`, which is the only mechanism that could put an estimator under the target's own noise.

14.21 Multilevel Monte Carlo is priced out, and the network is not chaotic

Sections 14.18 to 14.20 leave exactly one class of method standing. The front is 1,870x better in MSE than plain Monte Carlo at its own billed cost (`avg_variance / (2.61e10 / 4.203e6) = 6.8e-06` against `3.63e-09`), it is not moment-based, and 14.19 says whatever it carries has to be a joint object of dimension at least sixty-four. Samples are the only such object we know of, so the question becomes how sampling can be that much cheaper per unit of variance, and the textbook answer is multilevel Monte Carlo:

estimate `E[f_coarse]` with many cheap draws and `E[f - f_coarse]` with few expensive ones, coupling the two on the same input. MLMC fits every observed signature — it carries the joint law, it saturates at the finest level's bias so it looks budget-independent, its cheap levels are small matmuls with low arithmetic intensity and long wall clock, and it breaks the `MSE x FLOPs` invariant, which assumes a single `1/n` level.

The whole method rests on the coupling surviving. With per-level variances `V_l` and costs `c_l`, optimal allocation gives a speedup over plain MC of `V_f c_f / (sum_l sqrt(V_l c_l))^2`, so a two-level scheme with coarse cost `c` (in units of a full forward pass) and residual variance fraction `delta = Var(f - g) / Var(f)` returns `1 / (sqrt(c) + sqrt(delta))^2`. To close our 415x gap from a `delta` of essentially zero the coarse level would have to be about a thousand times cheaper than a full pass.

Rank truncation destroys the coupling completely. Replacing every `W_l` by its rank-`r` truncated SVD costs `r/128` of a full layer. Three networks, `2^18` draws, same input inside every difference:

rank	cost/full	`Var(f-g)/Var(f)`	variance reduction
1	0.039	1.000	1.00x
8	0.092	1.000	1.00x
32	0.273	1.000	1.00x
64	0.516	1.000	1.00x
128	1.000	0.926	1.08x

At **half the full rank** the surrogate still explains 7% of the variance, and the squared bias of its mean is 1.12 against a `Var(f)` of 0.062 — the coarse output is eighteen times further off in mean than the quantity's own spread. The optimal ladder comes out at **0.01x**, i.e. worse than not doing it. Width subsampling and depth truncation produce the same low-rank intermediate and inherit the same fate.

Precision is not priced. The other way to build a cheap level is arithmetic precision, and flopscope's billing table settles it in one line — `data/default_weights.json`, key `dtype_rates`:

```
bool 1.0   int8 1.0   float16 1.0   float32 1.0   float64 2.0   float128 4.0
```

`float16` is billed at exactly the same rate as `float32`. There is no discount below single precision, so a half-precision coarse level costs what the fine level costs and buys nothing. (The only dtype lever is the one we already use: stay off `float64`, which is billed double. See 7.20 for the one place the estimator cannot.)

Both routes closed, MLMC is out. That was the last hypothesis we had.

But the sensitivity measurement inverts a claim this document has been making. Running the same coupling against a *dense, small* relative weight perturbation `W_l -> W_l * (1 + eps * Z)` with `Z` standard normal, on the same three networks:

relative `eps`	`Var(f-g)/Var(f)`	**variance reduction**	`bias^2`
1e-7	3.9e-11	2.6e+10x	1.79e-13
1e-5	3.7e-08	2.7e+07x	9.62e-10
1e-4	3.6e-06	2.8e+05x	1.15e-07
1e-3	3.35e-04	2,984x	8.96e-06
1e-2	2.81e-02	35.6x	1.05e-03

`Var(f-g)/Var(f)` is proportional to `eps^2` across five decades, and the constant is small: the output's relative standard deviation moves by **18.3 times** the relative weight perturbation. Spread over thirty-two layers that is an amplification of **1.096 per layer**. A chaotic map would compound — even 1.5 per layer would give `4e5` — and this does not.

Section 7.21 read its own negative result as evidence that the target is "a chaotic function of 2.1 million numbers" mixed so thoroughly by thirty-two layers of rectification that no summary retains usable correlation, and used that reading to bound the learning route as well. The first half stands, the second does not. The measurement above says the weights-to-output map is *smooth and mildly conditioned*; 7.21's twenty features found nothing because they are twenty summary statistics, not because the underlying map is rough. Rank truncation kills the coupling not through chaos but through the sheer size of the perturbation — discarding half the spectrum is not a small change, and the `eps` table shows what "small" has to mean here.

So the raw-weight learning route that 7.21 declared bounded was never actually tested, and the property it was assumed to lack is measurably present. That is the one lever this document ends with open rather than closed, and the material for it already exists: the public `aicrowd/arc-whestbench-public-2026` dataset carries weights and `final_means` for eight hundred networks against the eight we have been validating on. It is a Phase 2 lever, not a Phase 1 one — but it is the only one left that our own measurements do not already forbid.

`research/mlmc.py`.

14.22 The target's own noise, measured rather than bounded

Section 14.20 bounded the ground truth's self-noise from above. If the labels are themselves Monte Carlo estimates, then any independent estimator's error against them is the sum of two independent pieces, so the leader's best per-network score is an upper bound on the label noise: at 1.28e-09 the labels must have been built from at least thirty-three million draws. That is a bound, and it was derived from somebody else's score. There is a sharper instrument sitting inside the published ledger, and it gives an equality.

Layer 0 is the one layer of this problem that is not an approximation. The input is exactly `N(0, I)`, so `a_0 = x W_0` is exactly Gaussian with variance `||W_0[:,j]||^2`, and `E[relu(a_0)]` has a closed form. An estimator that uses that closed form is not estimating layer 0 at all — it is computing it. Its reported MSE against the label therefore *is* the label's variance, with nothing of ours mixed in.

Both our own entry and the leading entry report the same number there:

entry	per-layer MSE at layer 0
dpskv5 (rank 1)	6.72e-10
ours (324409)	6.72e-10

The agreement is the check. Two independent implementations that both compute the layer exactly must agree, and if either were sampling the layer instead it would sit five orders of magnitude higher — our own working sample size of 5.8e4 draws would put a sampled layer 0 at `Var(relu(a_0))/n = 1.2e-05`. So 6.72e-10 is not an estimator error on either side. Inverting it,

```
Var(relu(a_0)) = 2 (1/2 - 1/2pi) = 0.6817
N_gt = 0.6817 / 6.72e-10 = 1.01e9 draws
```

which is thirty times above the floor that 14.20 could bound. Carrying the same draw count to the scored layer, where `Var(relu(a_31)) = 6.21e-02` on our eight validation networks, the absolute floor of this competition is

```
6.21e-02 / 1.01e9 = 6.1e-11
```

Against that floor the leader sits at 48x and we sit at 3,240x. Neither is close. Whatever the front is doing, it is not exploiting label noise and it is not near the information limit of the task — the observation in 14.20 that there is room below 3.63e-09 is not merely consistent with the data, it is quantified: nearly two orders of magnitude of room. We record this mostly because it removes an excuse. It would have been comfortable to conclude that our own 1.98e-07 was approaching something irreducible; it is not, by three and a half orders of magnitude.

How much of the odd part survives depth. Section 3 records that pairing `x` with `-x` makes layer 1 exact, which it does: `relu(a) + relu(-a) = |a|`, so the paired mean of the first layer has no variance at all. That is a statement about degree-one harmonics, and the harmonic accounting of 14.x says antithetic pairing removes exactly the odd degrees, no more. The natural question is what share of the scored layer's variance is still odd after thirty-two rectifications.

We first tried to answer it by comparing MSEs of a paired and an unpaired run. That was the wrong instrument twice over, and both attempts produced impossible numbers -- at equal evaluation count the ratio is exactly `1 + rho` with `rho = Corr(f(x), f(-x))`, so nothing below `0.5x` is admissible, and we measured `0.62x` and then `0.45x`. The fault is resolution, not the method: per-network MSE spans a factor of ten across the suite, so a ratio of means over four networks and four seeds is dominated by which seeds landed where. The primitive quantity is `rho` itself, and it is estimated from `65,536` draws times `256` outputs rather than from sixteen scalar MSEs:

network	mean rho	median	range
0	-0.0483	-0.0368	[-0.146, +0.004]
1	-0.0312	-0.0158	[-0.104, +0.001]
2	-0.0489	-0.0326	[-0.153, +0.007]

The correlation is negative, so pairing helps, and the equal-cost gain is `1/(1 + rho) = 1.03x` to `1.05x`. Read as a share, the odd degrees carry about `4.8%` of the scored layer's variance, down from `100%` at layer 1. That is the whole of what antithetic pairing can ever be worth here, and the deployed estimator already takes it.

The methodological point is worth more than the number. A variance-reduction device verified at one depth carries no entitlement at another, and a ratio of noisy second moments is not the way to find out -- measure the correlation the device acts on, and the answer arrives with three orders of magnitude more statistical power and a sign check that catches a broken harness. Ours was broken in two different ways before the sign check caught it. `research/.anti2.py`.

The plug-in read-out is worse than the sample it is built from. One route we had not closed was to stop averaging `relu` per output and instead average the shared 256-dimensional summary of `h_30`, reading `E[relu(a_31,j)]` off the projected mean and variance in closed form. The appeal is that every output then reuses all `256 n` numbers rather than `n` of them. Measured at `n = 2^20`, and again with near-exact moments at `n = 2^24` to separate the bias floor from the sampling error:

read-out	MSE	vs direct MC
direct Monte Carlo	9.47e-08	1.0x
plug-in Gaussian	9.86e-07	0.1x
Edgeworth, 4 cumulants	1.15e-07	0.8x
plug-in Gaussian, exact moments	9.32e-07	0.1x
Edgeworth, exact moments	2.18e-08	4.3x

The averaging gain is real but small, and it is swallowed whole by the bias of the read-out: even handed exact moments the four-cumulant expansion bottoms out at 2.18e-08, seven times above the front. This is the fifth independent direction from which the moment state has now closed (14.18, 14.19, 14.20, and the two rows above), and we regard the question as settled: whatever carries the front's accuracy, it is not a low-order moment summary of the scored layer. `research/raolast.py`.

The pair closure is limited by its model, not its quadrature. For completeness we ran the deterministic pair-propagation chain at three quadrature resolutions:

K = 8	K = 16	K = 24	dense quadrature
5.8606e-05	5.8606e-05	5.8606e-05	5.8605e-05

Tripling the resolution moves nothing in the fifth significant figure, which locates the error entirely in the Gaussian closure rather than in the integration. Rectifying the diagonal in closed form takes the family to 2.35e-05 and no further. As a by-product, the Mehler series at `K = 8` reproduces the dense quadrature to five figures at a twentieth of the wall clock — free, and worth nothing here, since metered time is not billed. `research/mehlerchain.py`.

15. Disclosure

15.1 The submission endpoint returns other participants' scores

`GET https://www.aicrowd.com/api/v1/submissions/<id>`, authenticated with an ordinary participant API key, returns `id`, `challenge_id`, `challenge_round_id`, `created_at`, `grading_status_cd`, `grading_message`, `score` and `score_secondary` for **any** submission id, not only the caller's. There is no participant identity in the response and no listing endpoint, so ids have to be enumerated; that is the only barrier, and it is not one.

We used it. Everything this note says about "the front of the field" — the 4.870822e-09 leading raw error, the multiplier near 1.0 that shows the leading *scorer* spends the whole budget, and the raw error agreeing to sixteen digits across two of that participant's submissions, which is how we know their estimator is deterministic — was read through this endpoint rather than inferred from the leaderboard. The figures we quote come from a scan of 1405 graded submissions on this challenge: best score `5.589914e-09` at a multiplier of `0.973`, best raw error `4.870822e-09` at a multiplier of `2.132`, 864 submissions scoring better than ours, and 150 sitting at the multiplier floor with a median raw error of `3.44e-06`. An earlier scan a quarter that size put the leading raw error two orders too high and the floor population at a twentieth of its real size; both numbers appeared in a draft of this document and both were wrong, which is worth saying because the corrected floor figures are what redirected the last day of work. It changed what we did: it is why we stopped refining the sampler and measured the ceiling in section 8 instead, and then the floor in section 10.

We report it rather than sit on it, and we would rather it were closed than that we keep the advantage. Two things follow for the organisers. A competition whose per-submission scores are readable by all participants is a different game from one whose scores are not — in particular `score_secondary` is the raw error before the budget multiplier, so the endpoint discloses each entrant's accuracy-versus-cost operating point, which is most of a method's design. And for Phase 2, if the private split is graded through the same API, the same enumeration reads private scores.

We hold no data beyond the numeric fields above, which carry no participant identity, and we have not attempted to deanonymise any id.

15.2 Method

This work was done with Claude (Anthropic) in an agentic setting: the model proposed the hypotheses, wrote the experiment harnesses and the estimator, ran the measurements, and drafted this document; the human set direction and reviewed. Every quantitative claim above is a measurement made by that harness against the public `mini` split, not a model assertion — the numbers in sections 1, 5 and 6 are reproducible from the scripts. Where we offer an *interpretation* of a measurement (why CBC fails in 5, why rotation fails in 6.2, why the diagonal third cumulant fails in 6.12), that is our reading and we have not proved it.

Two specific hedges, since the call for write-ups asks for them. First, the Gram-Charlier corrections in 6.10 and 6.12 were derived by the model and checked numerically — against the closed-form arccos kernel at `a = 0` (agreement to 1e-15) and against 8e6 Monte Carlo draws at `a != 0` — but not derived independently by the human; the sign error we describe is exactly the kind of mistake that survives a plausible-looking derivation, and it survived ours for one full experiment before the numerics caught it. Second, the FLOP figures we quote for `K = 3` cumulant propagation are read off Table 1 of

arXiv:2605.05179 and evaluated at `n = 256, L = 32`; we did not implement the full algorithm, so "it does not fit in the budget" is an arithmetic claim about the paper's stated cost, not a measurement of ours. What we *did* implement, in 10.5, is the `K = 2` member of the same family — full covariance propagation with an exact rectified bivariate closure — and there the budget claim is a measurement: 2.86 s per MLP for 12 of 32 layers, 280% of `B` in wall time. That does not verify the paper's `K = 3` arithmetic, but it does mean the conclusion no longer rests only on a table we read.

Prior work by other participants that we build on or duplicate is credited inline: `jamesrahenry` (#18097) for the error-compensation account of deep moment propagation, `pscamillo` (#18063) for the stable scalar bias of the Gaussian closure, `radiant-allomancer` (#18085) for the measured performance of the full analytic hybrid, `evaaaz` (#18053) for independently finding that control variates give 1.0–1.2x, and `arianvassili` (#18105) for the machine-checked lower-bound argument that framed how we read the leaderboard. The tent transform is Hickernell's; the CBC construction is Sloan and Reztsov's; the estimator itself contains nothing that is not in a QMC textbook.