



FP32 Almost Everywhere, FP64 Where It Counts

Measured quantitative-finance compute on six workstation Blackwell GPUs at a 250 W cap, and the precision

Paper: PT-R-2026-034 v1.0

Author: PureTensor Ltd

Date: 6 October 2026

Status: Published

Abstract

We measured how much quantitative-finance computation six workstation-class Blackwell GPUs and three workstation and server CPUs can carry, and at which numerical precision. The work covered Monte Carlo pricing, full-revaluation market risk, adjoint Greeks, early-exercise pricing, dense linear algebra and tick research. The cards are the RTX PRO 6000 Blackwell in two variants: two Workstation Edition cards on one node, and two Max-Q Workstation Edition cards on each of the other two nodes. All six were capped at 250 W. NVIDIA's architecture whitepaper gives this silicon FP64 (double precision) at 1/64 of the FP32 (single precision) rate, so the precision question decides how useful the hardware is. Every Monte Carlo result below carries a standard error (SE, the sampling uncertainty of the estimate) and a reference value.

FP32 Monte Carlo with FP64 accumulation of the payoff sums ran 11.3–22.7x faster per card than FP64, depending on the workload. Its prices agreed with closed-form references to within 0.04–0.98 SE for a European call. Across all six cards the 252-step Asian option ran at 1.75e12 path-steps per second in FP32. On a deliberately gamma-heavy option book, FP32 and FP64 full-revaluation VaR differed by 7.1e-8 in relative terms. The one place FP32 left a trace in pricing was a Heston model. At four discretisation step counts, FP32 sat above FP64 every time, with a pooled offset of $+0.00072 \pm 0.00030$ ($z = 2.43$, 7.1e-5 of price). We read this as suggestive rather than established.

Longstaff-Schwartz least-squares Monte Carlo is where precision failed in practice. Run entirely in FP32, the Bermudan put price was between 0.055 % and 1.23 % below the FP64 value, depending on how the regression was accumulated and on which library build ran it. Moving only the 4x4 regression to FP64 removed the bias at about 1.5x the cost of the fastest FP32 variant. On a single card, changing the CUDA 12.8 software stack for CUDA 13.0 made the skinny matrix products in that regression 8.6x faster in FP32 and 104x faster in FP64. The change also altered the FP32 answer.

Several smaller results follow. The Zen 5 CPU node reached 2.73 TFLOPS in FP64 matrix multiply, more than any single card's native 1.87 TFLOPS. Ozaki-scheme FP64 emulation on one card reached 7.675 TFLOPS (4.13x native) with error 9.1e-15. At the common 250 W cap the Max-Q cards beat the Workstation Edition cards at FP32 and lost to them at FP64. Adjoint differentiation produced 128 Greeks 101–121x faster than bumping.

All of this was measured on a single day. Most figures are single runs, the six cards ran identical seeds, and the GPU variant comparison is confounded by compiler version. We state below



which claims survive those limits.

1. Question and Scope

We asked a practical question: what quantitative-finance work can this hardware do, and where is FP32 safe? We wanted the answer from measurement rather than from specification sheets. The whitepaper (Appendix A, Table 4) gives FP32 non-tensor throughput of 109.7 TFLOPS for the Max-Q and 126.0 TFLOPS for the Workstation Edition. FP64 at 1/64 of those rates is therefore 1.71 and 1.97 TFLOPS at boost clock. The whitepaper describes the GB202 die as having "384 FP64 Cores (two per SM)" and "a very minimal number of FP64 Tensor Cores ... for program correctness". On this hardware, FP64 is close to an afterthought.

Latency-sensitive trading was out of scope; we have no colocation.

2. Hardware, Software and Method

The three nodes are a 32-core Zen 4 Threadripper 7970X (64 threads, 251 GiB), a 32-core Zen 5 Threadripper PRO 9975WX (64 threads, 503 GiB) and a 64-core Zen 3 EPYC 7713 (128 threads, 503 GiB). The Zen 4 node holds the two Workstation Edition cards, whose board power limit can go to 600 W. The other two nodes hold Max-Q cards rated to 325 W. Every card has 96 GB of GDDR7 with ECC and 188 SMs (streaming multiprocessors). All ran driver 615.71.09 at a 250 W limit over PCIe, with no NVLink and no inter-card communication. For the concurrent run, resident GPU services were paused. The Zen 3 node also hosts a language-model service and a container-orchestration worker, which matters for its CPU results.

The software stacks differed. The Workstation node ran PyTorch 2.10.0 on CUDA 12.8 (cuBLAS 12.8.4) with CuPy 13.6.0. The Max-Q nodes ran PyTorch 2.11.0 on CUDA 13.0 (cuBLAS 13.1) with CuPy 14.2.0. A CUDA 13.0 environment was later added to the Workstation node for a same-card rerun.

The Monte Carlo kernels are fused CUDA kernels compiled through CuPy and templated on float and double. Random numbers come from an in-register Philox4x32-10 generator (counter-based, so each thread can compute its own stream without stored state), followed by Box-Muller. FP32 draws 4 normals per Philox call from 32-bit uniforms. FP64 draws 2 per call from 53-bit uniforms, so the two precisions see different random numbers. We launched 256 threads per block and 16 blocks per SM, giving 770,048 threads. Each thread simulates its paths serially. Path state and the Asian running average are held in working precision, but per-thread payoff sums and sums of squares are accumulated in FP64 in both cases. Path counts were sized to about 1.5 s per run. Each kernel ran one warm-up and three timed repetitions (seeds 1000 to 1002). We report the best of three for time and the last repetition for price and SE. Best-of-three applies only to these fused kernels; other figures are a single timed pass or a short mean. Board power was sampled by nvidia-smi about 4 times a second.

The workloads were as follows. The European call ($S_0=100$, $K=100$, $r=5\%$, $\sigma=20\%$, $T=1$) is simulated under geometric Brownian motion (GBM) in one exact log-step and has a Black-Scholes reference of 10.4505836. The arithmetic-average Asian call takes 252 daily steps; it has no closed form, so we checked FP32 against FP64. The Heston call ($v_0=0.04$, $\kappa=2$, $\theta=0.04$, $\xi=0.5$, $\rho=-0.7$) uses Euler full truncation, which floors negative variance at zero inside the drift and diffusion terms. Its reference is a semi-analytic value of 10.154627. Its parameters violate the Feller condition ($2\cdot\kappa\cdot\theta = 0.16 < \xi^2 = 0.25$), so the variance process touches zero and the discretisation is under strain.



3. Monte Carlo Throughput and Accuracy

Workload	FP32 per card	FP64 per card	FP64 penalty	Six-card FP32	Six-card FP64
European, 1 step	1.65–2.08e11	1.28–1.59e10	11.3–16.3x	1.18e12	8.26e10
Asian, 252 steps	2.44–3.20e11	1.37–1.71e10	15.6–22.7x	1.75e12	8.86e10
Heston, 252 steps	1.21–1.72e11	0.78–0.97e10	13.7–21.7x	9.28e11	5.01e10

Units are path-steps per second. The FP64 penalty is smaller than the 1/64 hardware ratio suggests, because these kernels are not pure arithmetic: generation, transcendental functions and control flow all take time. The penalty is still an order of magnitude or more. Across six cards the Asian penalty is 19.8x. At FP32 rates, one card prices a 252-step Asian on 1 billion paths in 0.79–1.03 s.

The accuracy figures come from one Workstation card. The European FP32 run on 4.25e11 paths returned 10.4506055 against 10.4505836, 0.97 SE away. All cards fell within 0.04–0.98 SE in both precisions. For the Asian option, FP32 returned 5.782085 and FP64 5.781781, a difference of 0.00030 against a combined SE of 0.00081.

Energy follows throughput. On the Asian option, FP32 delivered 0.97–1.33e9 path-steps per joule of board power, 14.9–17.2x FP64's per-joule rate on the same card. FP32 runs averaged 233–251 W, pressed against the cap. FP64 on the Workstation cards drew 222–246 W. On the Max-Q cards FP64 drew only 148–187 W: the FP64 units cannot use the available power.

4. The Heston Offset

At 252 steps in the six-card run, FP32 Heston prices sat 3.5–4.1 SE above the semi-analytic reference (+0.00108 to +0.00122), while FP64 sat 0.23–0.57 SE away. Part of any such gap is Euler discretisation bias, which affects both precisions. To separate discretisation from precision, we ran a step-doubling follow-up on one Workstation card. It used 63, 126, 252 and 504 steps, with equal path counts in both precisions at each step count (1.478e9, 7.39e8, 3.70e8 and 1.85e8 paths).

Steps	FP64 bias	FP32 bias	FP32 minus FP64	SE of the difference	z
63	+0.00464	+0.00536	+0.00073	0.00041	1.79
126	+0.00152	+0.00223	+0.00071	0.00057	1.25
252	+0.00031	+0.00117	+0.00086	0.00081	1.07
504	+0.00021	+0.00063	+0.00042	0.00114	0.37

FP64 bias falls as steps double, as Euler full truncation should. FP32 sits above FP64 at all four step counts, but no single step count reaches 2 SE. The draws are independent, so the relevant uncertainty is the SE of the difference, not the per-run SE. Pooling the four by inverse variance gives $+0.00072 \pm 0.00030$, $z = 2.43$, or $7.1e-5$ of price. The evidence for a real offset is therefore the four-of-four sign agreement and that pooled estimate. Resolving an effect this size needs more than $1e8$ paths. We have not identified a mechanism. The cost of the comparison is also visible: FP32 took 0.66–2.22 s per step count and FP64 9.66–9.82 s.

Set against the discretisation error, the offset is smaller than FP64's own bias at 63 and 126 steps, comparable at 252, and larger at 504.



5. Longstaff-Schwartz: The Precision Trap

Least-squares Monte Carlo (LSMC) prices early-exercise options. At each exercise date it regresses discounted future cash flows on functions of the current state, and uses the fitted value to decide whether to exercise. Our case is a Bermudan put ($S_0=36$, $K=40$, $r=6\%$, $\sigma=20\%$, $T=1$) with 50 exercise dates and 4,194,304 paths (2^{21} plus antithetics), written in PyTorch. The basis is a cubic in S/K over in-the-money paths, and the 4×4 normal equations are solved directly. We accumulated those equations in two ways. "GEMM" uses a skinny $M \times 4$ matrix product through cuBLAS. "Moments" uses 7 power moments and 4 cross moments computed by bandwidth-bound reductions. Each variant was one timed run with seed 7, after a warm-up.

Variant	Stack	Price	vs all-FP64 4.475512	vs mixed	Time
All FP32, GEMM	CUDA 12.8 (Workstation)	4.420657	-1.23 %	-1.26 %	0.81-0.88 s
All FP32, GEMM	CUDA 13.0 (all cards)	4.473039	-0.055 %	-0.086 %	0.094-0.100 s
All FP32, moments	CUDA 12.8	4.449454	-0.58 %	-0.61 %	0.048-0.052 s
All FP32, moments	CUDA 13.0	4.443568	-0.71 %	-0.74 %	0.055-0.064 s
FP32 paths + FP64 regression	both	4.476867	+0.030 % (SE 0.0014)	reference	0.080-0.091 s
All FP64, moments	both	4.475512	reference		0.117-0.131 s
All FP64, GEMM	CUDA 13.0	4.475512	0.00 %		0.40-0.48 s
All FP64, GEMM	CUDA 12.8	4.475518	0.00 %		42.15-44.86 s

The mixed variant returned a bit-identical 4.4768673 under both CUDA builds. The FP32 paths are therefore identical across builds, and the FP32 bias comes entirely from how the regression is accumulated: which library path runs and in what reduction order. The bias ranged from 0.06 % to 1.2 %, and nothing in the setup would have predicted its size beforehand. A 4×4 system built from moments of a cubic is badly conditioned, so the result is unsurprising. What these runs show is how large the error is and how much it depends on software that users do not normally inspect. Moving only the regression to FP64 removes the bias for about 1.5x the time of the fastest FP32 variant.

External checks support the FP64 figure. A 20,000-step CRR binomial American put gives 4.48668. Because it allows continuous exercise, a 50-date Bermudan must sit below it, and the all-FP64 price is 0.249 % lower. Longstaff and Schwartz (2001), Table 1, report 4.472 (s.e. 0.010) by simulation and 4.478 by finite differences for this case; our value is within that simulated range. Belletti et al. (arXiv 1906.02818) report that their GPU Longstaff-Schwartz baseline "was often failing to do the Cholesky decomposition" and needed regularisation. The problem is not specific to our setup.

6. Same Card, Different Library Build

One Workstation card	CUDA 12.8 stack	CUDA 13.0 stack
FP32 LSMC, GEMM regression	0.812 s, 4.420657	0.0941 s, 4.473039
FP64 LSMC, GEMM regression	42.15 s	0.404 s
BF16 / FP16 GEMM 8192	377 / 302 TFLOPS	431 / 409 TFLOPS



One Workstation card	CUDA 12.8 stack	CUDA 13.0 stack
FP8 GEMM	490 TFLOPS	fails (CUBLAS_STATUS_NOT_INITIALIZED)

On the same physical card, the CUDA 12.8 stack ran the skinny GEMM 8.6x slower in FP32 and 104x slower in FP64. Under CUDA 13.0 the Workstation card's FP32 LSMC price matched the Max-Q cards exactly. The earlier cross-card gap (8.0–9.4x in FP32 and 88–96x in FP64 against the Max-Q cards) was therefore a library effect, not a hardware one. CUDA 13.0 also raised BF16 and FP16 GEMM by 14 % and 35 %. It broke FP8, which failed with a cuBLASLt heuristic error, and the Max-Q cards failed in the same way.

7. Market Risk

Book 1 tests throughput. It holds 10,000 Black-Scholes options on 100 underlyings with a 5-factor correlation structure. Each scenario applies a one-day correlated normal spot shock, an independent lognormal vol shock and one day of decay. We repriced every option in each of 200,000 scenarios, giving 2e9 valuations. VaR99 is minus the 1 % quantile of portfolio P&L. ES97.5 (expected shortfall) is minus the mean P&L at or below the 2.5 % quantile. Scenarios were generated once in FP64 on the host and cast, so both precisions see identical inputs. Timing excludes generation, transfer and the quantile step.

Book 1 (2e9 valuations)	Time per card	Valuations/s per card	Six cards
FP32	0.19–0.28 s	7.2–10.5e9	5.67e10
FP64	0.86–0.99 s	2.0–2.3e9	1.35e10

FP32 and FP64 differed by 3.5e-6 in VaR99 and 2.2e-6 in ES97.5. The worst single-scenario P&L difference was 0.41 on an RMS P&L of 17,115. Book 1 is nearly Gaussian, however (VaR99 = 2.33 x RMS, ES97.5/VaR99 = 1.006), so it says little about precision when the book is nonlinear.

Book 2 is built to be nonlinear. It holds 10,000 short-dated options near the money on 50 underlyings, with 15 to 30 trading days at inception and 5 to 20 remaining after the horizon. Sixty per cent of positions carry an extra short 3,000, so the book is net short gamma and vega. The horizon is 10 days. Spot shocks are correlated Student-t with 4 degrees of freedom. The vol multiplier is $\exp(0.15 \cdot W)$, with W normal plus $0.5 \cdot \max(-Z, 0)$, so vol rises when spot falls. This gives 100,000 scenarios and 1e9 revaluations on one Workstation card under CUDA 13.0.

Book 2 (1e9 revaluations)	Full revaluation	Delta-gamma	Delta only
VaR99 (P&L units)	62.09 million	110.33 million (1.78x over)	5.41 million (11.5x under)
ES97.5 / VaR99	1.136		1.063
P&L skew	-28.9		

FP32 and FP64 differed by 7.1e-8 in VaR99 and 6.5e-8 in ES97.5. The worst single-scenario difference was 139 on an RMS P&L of 19.5 million. FP32 took 0.113 s and FP64 0.503 s. The approximations are evaluated on the same simulated scenarios, using Black-Scholes delta and gamma from automatic differentiation. The delta-only figure is not a parametric variance-covariance VaR. Both approximations also omit the vol and time shocks that full revaluation includes, so their errors combine missing curvature with missing risk factors. Even so, on a book of this shape the approximations miss in opposite directions by large factors, and precision is not the main risk.

8. Adjoint Greeks



Adjoint algorithmic differentiation (AAD) obtains all first-order sensitivities from one backward pass through the computation. We applied it to a 64-asset correlated basket call, simulated in one step on 2,097,152 paths. PyTorch reverse-mode autodiff gives 128 sensitivities: 64 deltas and 64 vegas. Bump-and-revalue uses central differences, which costs 256 prices on common random numbers.

Six cards	Price + 128 Greeks (AAD)	AAD cost / one price	Bump-and-revalue	AAD speed-up	Max delta difference vs bump
FP32	8.6–10.3 ms	2.61–2.82x	0.88–1.04 s	101–109x	4.4e-7
FP64	19.9–22.7 ms	2.23–2.36x	2.36–2.64 s	115–121x	3.4e-8

Only deltas were cross-checked against bumping. A one-step terminal payoff is the easiest case for AAD, because the tape (the stored record of operations replayed backwards) stays small. Path-dependent payoffs are untested.

9. Dense Linear Algebra and FP64 Emulation

Per card, 250 W	Workstation Edition	Max-Q
FP32 FMA (TFLOPS)	77.9–85.7 (62–68 % of 126.0)	89.3–90.2 (81–82 % of 109.7)
FP64 FMA	1.67–1.81	1.45–1.49
FP64 DGEMM 8192	1.72–1.87	1.50–1.53
SGEMM / TF32	32.2–35.4 / 69.1–84.6	48.9–49.7 / 100.6–106.9
BF16 / FP16	311–377 / 276–305	264–271 / 258–261
Cholesky 16384, FP32 / FP64	17.2–20.7 / 1.68–1.84	12.8–28.1 / 1.48–1.52
Symmetric eigenvalues 8192, FP32 / FP64	0.59–0.66 s / 1.55–1.75 s	0.54–0.56 s / 1.50–1.61 s

The FP32/FP64 FMA ratio measured 46.8–61.5x. Native DGEMM tracks the whitepaper's FP64 figures closely. The BF16/FP16 figures were taken under CUDA 12.8 on the Workstation cards and CUDA 13.0 on the Max-Q cards. FP32 Cholesky varied from 12.8 to 28.1 TFLOPS between identical Max-Q cards. A single-card Workstation run on a card shared with a resident service gave 1.5 TFLOPS, so we treat FP32 Cholesky as sensitive to contention.

The Ozaki scheme emulates FP64 matrix multiply by splitting operands into fixed-point slices and multiplying them on faster low-precision units. On one Workstation card, built with the CUDA 13.2 toolkit (linked cuBLAS 13.4.1), native DGEMM ran at 1.860 TFLOPS. Emulation reached 7.675 TFLOPS, 4.13x faster, with maximum error 9.1e-15 relative to the largest entry of the native result. NVIDIA's own paper (Schwarz et al., arXiv 2511.13778) reports up to 13.2x on the Server Edition of the card. Emulation helps only GEMM-shaped work; Monte Carlo paths gain nothing from it. In the same run, BF16x9 emulation of FP32 gave 35.64 TFLOPS against 35.71 native with identical error (2.4e-6). It probably did not engage, and we cannot yet explain why.

10. Card Variants at the Same Power Cap

At 250 W the Max-Q cards beat the Workstation Edition on FP32 Monte Carlo. Over card pairs the margin was 13–26 % on the European, 13–31 % on the Asian and 22–43 % on Heston (mean ratios 1.19, 1.21, 1.32). They were also faster on SGEMM and FP32 FMA. The Workstation Edition was faster at FP64, by a mean ratio of 1.17–1.18 across FMA, DGEMM, Cholesky and the three FP64 Monte Carlo kernels (pairwise 1.11–1.24).



The Max-Q SM clock averaged 1,556–1,934 MHz during FP32 Monte Carlo and 2,260–2,351 MHz during FP64, when the card was well under its power cap. The Workstation clock readings were unusable: one card reported a constant 2,422 MHz and the other a constant 555 MHz. Our working hypothesis, which is not confirmed, is that silicon rated to 600 W gives up more clock at 250 W than silicon rated to 325 W. The Monte Carlo comparison is confounded by compiler version (Section 15). The SGEMM gap persisted on the Workstation card under CUDA 13.0, at 34.6 TFLOPS, which supports a real hardware effect.

11. CPUs

Measure	Zen 5	Zen 4	Zen 3
Single-core NumPy FP64 Asian, path-steps/s	88.4 million	71.5 million	39.7 million
Per task, 240 concurrent (median task time)	18.8 million (20.1 s), 21 %	13.7 million (27.6 s), 19 %	2.9 million (128.9 s), 7 %
DGEMM 8192 FP64, fenced	2.73 TFLOPS (32 threads)	1.15 (32)	0.78 (64, contended)
SGEMM 8192	5.23 TFLOPS	2.02	1.74

The CPU Monte Carlo ran as 240 single-threaded tasks (64, 64 and 112 per node) of 1.5 million paths each, 9.07e10 path-steps in total, under a static split. All 240 tasks completed and exited cleanly. The run took 138.7 s, or 6.5e8 path-steps per second. Perfect balancing would have given 2.41e9; the shared Zen 3 node was the straggler. Per-task throughput fell to about 20 % of single-core throughput at full concurrency. This is consistent with memory-bandwidth limits, since each chunk materialises 4,096 x 252 FP64 arrays of about 8 MB, but we did not read bandwidth counters. A Numba-compiled scalar version was slower than vectorised NumPy on one core (31.5 million on Zen 4, 41.0 million on Zen 5).

On FP64 dense algebra the CPUs are competitive. The Zen 5 node alone exceeds any card's native DGEMM. The three CPUs together reach 4.65 TFLOPS, 48 % of the six cards' 9.67. For Monte Carlo, six-card FP32 beat CPU NumPy wall-clock by 2,678x and the balanced figure by 727x. Six-card FP64 beat balanced CPU NumPy by 37x. The CPU baseline was idiomatic NumPy, not a tuned AVX-512 kernel, and these ratios overstate the gap to such a kernel by an amount we did not measure.

12. Tick Storage

We wrote a synthetic quote history of 20 days at 25 million quotes a day, 500 million rows over 2,000 symbols. It was stored as zstd Parquet in 1 million-row row groups, 4,572,361,900 bytes in total (about 9.1 bytes per row). We queried it with DuckDB 1.5.2 on 32 threads on the Zen 3 node. Symbols and timestamps are deterministic, so the data compresses unusually well.

Query	Local NVMe cold	HDD erasure-coded CephFS cold	NVMe warm	CephFS warm
Full scan, count + mean spread	1.078 s	5.573 s	0.814 s	0.828 s
Size-weighted mid by symbol	1.319 s	4.449 s	1.034 s	1.054 s
One symbol over 20 days	2.640 s	6.130 s	2.321 s	2.364 s
Minute bars, all symbols (15.6 million bars)	14.266 s	17.865 s	14.295 s	14.263 s



The cold full scan was 5.2x faster on local NVMe (464 million rows/s, against 89.7 million rows/s or 0.82 GB/s from the networked tier). Warm times sit close to NVMe cold, so 464 million rows/s reflects DuckDB's Parquet decode rate rather than the drive. Minute-bar construction is CPU-bound at about 35 million rows/s. Single-stream copy into the networked tier ran at 213 MB/s, and generating the data took 218.7 s.

13. A Production Port

We ported an existing pure-Python production Monte Carlo model to a vectorised GPU implementation. Validated draw by draw against the original on 2,000 random parameter sets, the headline output agreed to a maximum absolute difference of $8.9e-16$, and a secondary output to $1.3e-13$ relative. That is agreement up to summation order. In FP64 on one Workstation card the port ran $1e8$ simulations in 11.2 s, 8.9 million per second, against 2,109 per second for the original on one CPU core. On identical draws, FP32 gave a mean absolute error of $4.7e-8$ and a worst case of $1.9e-4$ in the headline output.

14. Precision Doctrine, as Measured

Our measurements support FP32 with FP64 accumulation for three uses: Monte Carlo path generation for pricing and risk, full-revaluation VaR and ES P&L vectors including nonlinear books, and AAD pathwise Greeks. They support FP64 for four others: LSMC and similar regressions, model validation and independent price verification where a $7e-5$ offset becomes visible at $1e8$ paths, covariance and ill-conditioned linear algebra, and anything reconciled to an FP64 golden source. Where the FP64 work is GEMM-shaped, GPU emulation is an option. Our FP64 reference machine is the Zen 5 CPU node. These findings agree with Giles (arXiv 1212.1377): single-precision error is far smaller than Monte Carlo, discretisation and model error, but payoff summation and bumped Greeks need care.

15. What These Numbers Will Not Carry

Everything was measured on one day. The Monte Carlo kernels report the best of three repetitions; every other figure is a single timed pass or a short mean. The six cards are not independent replicates. They ran identical seeds, so prices and risk figures agree across cards by construction, and two Max-Q cards returned bit-identical FP64 prices. Throughput ranges across cards are real. Statistical agreement is, in effect, one observation.

The comparison between card variants is confounded. The Workstation cards ran CUDA 12.8 PyTorch and CuPy 13.6.0, and the Max-Q cards ran CUDA 13.0 and CuPy 14.2.0, so the CuPy-compiled Monte Carlo kernels were built by different compilers. Only the SGEMM gap has been checked under a common stack. The single-card follow-ups (emulation, step-doubling, gamma book, CUDA 13 rerun) ran on one Workstation card shared with a resident service.

The LSMC "same seed" FP64 comparison is not guaranteed to share draws with FP32, because PyTorch generates normals per data type. The mixed variant, which shares the FP32 paths exactly, is the clean reference. LSMC prices are in-sample. The books and tick data are synthetic, and the tick data compresses better than real quotes would. The "cold" networked-storage runs dropped only the client page cache, not storage-side caches, and the data had just been copied in. The CPU baseline is untuned. CPU calibration used 20,000 paths in one run, and the CPU single-core and DGEMM figures survive only in the session log. Power is board power from nvidia-smi, and the short FMA probes had too few samples to measure it.



Three claims survive these limits. FP32 Monte Carlo is an order of magnitude or more faster than FP64 on this hardware and agrees with references within sampling error for the European and Asian cases. All-FP32 LSMC carries a bias of variable size that FP64 regression removes. The library build changed both the speed and the FP32 answer on the same card. Three claims are weaker. The Heston offset rests on a pooled z of 2.43 and is suggestive only. The variant ranking is plausible but partly confounded. The CPU-to-GPU ratios hold only against idiomatic NumPy.

16. What We Changed and What Comes Next

We have recorded the precision rule in Section 14 as our working doctrine for quantitative work, and committed the harness. Moving the Workstation node's default GPU stack to CUDA 13 is the cheapest performance fix available to us. A separate CUDA 13 environment exists, but as of 3 October the default was still CUDA 12.8, and the FP8 failure on CUDA 13.0 builds remains unfixed.

The next experiments follow from the open questions. A power-cap sweep at 250, 350, 450 and 600 W should test the variant hypothesis. A Heston rerun with FP64 accumulation of the log-price should test whether that removes the offset. We also plan QR regression for LSMC and FP64 emulation for Cholesky and QR, neither yet tested. Beyond those, we intend to build a harness modelled on the shape of STAC-A2 (Heston, 5 to 10 correlated assets, LSMC American exercise, a full Greek set). STAC-A2 is STAC's trademarked benchmark; we have not run it and will not attach its name to unofficial results. Path-dependent AAD and multi-GPU scaling are also on the list.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-034. All measurements were taken on 2 October 2026 (UTC), and raw artefacts are retained, with the CPU single-core and DGEMM figures held in the session log rather than the results directory.