REPROGPU: Cross-Adapter Bit-Reproducible WebGPU Kernels and a Conformance Harness for a Declared Integer Subset

DeepForge Research

September 19, 2026 · reprogpu

Technical report · Artifact · Not peer reviewed

Abstract

GPU results are not reproducible across vendors. The WGSL specification leaves rounding direction, reassociation, fusion, and subnormal flushing unspecified (section 15.7), so a floating-point kernel is free to return different bytes on different adapters. REPROGPU is a browser/WebGPU library and cross-adapter conformance harness that pins byte-identical SHA-256 outputs for a declared integer and fixed-point WGSL kernel subset - exact f32 summation by a 320-bit integer superaccumulator (K1), Philox4x32-10 (K2), Q16.16 GEMM with one arithmetic floor shift (K3), and integer SHA-256 (K4) - and declares general floating-point results out of scope, on the specification's own boundary. K5 is an f32 matmul negative control that the browser lab runs and records; it carries no pass criterion and no pinned hash, and no outcome for it is recorded in this artifact, so this paper does not claim an observed float divergence. The CPU-only gate reproduces the pinned reference hashes byte-for-byte in five of five criteria: K1 at count 4,104 with hash c046384e, K2 at 65,536 with f9004cf3, K3 at 65,536 with fbe13a3b, and K4 at 16,385 with eea119f4. It also matches three Random123 Philox known-answer tests and the FIPS SHA-256 vectors, pins all five WGSL source hashes, and scans the integer kernels for float types, atomics, discard, and unmasked runtime shifts. The gate cannot compile WGSL or run a GPU; the browser lab is the demonstration, and no CI artifact is evidence of GPU agreement.

Keywords: webgpu, reproducibility, integer kernels, conformance harness, superaccumulator, philox, sha-256

1. Introduction & motivation

A GPU result is not a portable number. The WGSL CRD of 2026-09-15 specifies floating-point arithmetic only loosely: section 15.7.4.1 gives x+y, x-y, and x*y as correctly rounded to either neighbour with the rounding direction unspecified, gives x/y as 2.5 ULP rather than correctly rounded, permits reassociation and fusion in section 15.7.5, permits subnormal flushing and an indeterminate value for runtime overflow and NaN in section 15.7.2, and says the sign of zero may be ignored. Two drivers on the same machine may therefore return different bits for the same shader, and two vendors need not agree at all. This is not a defect in a particular engine; it is the specification's declared boundary. NVIDIA's own reproducibility team states that bit-exactness is not expected across GPU architectures (framework-reproducibility issue #28), and the WebGPU CTS tests floating-point accuracy within ULP tolerances rather than bit-exact cross-adapter equality.

The consequence is that a browser GPU computation cannot currently be re-run and compared by a third party. Scientific reproduction of a GPU-assisted result, verifiable browser compute, and automated grading of a computational answer all need the same missing property: the same declared program on a different adapter either produces the same bytes or is reported as a

divergence. The smaller observation, and the one this paper builds on, is that a well-defined subset of WGSL already has that property. Integer wrapping arithmetic, bitwise operations, comparisons, bitcast between i32 and u32, and the little-endian host-shared buffer layout are all specified exactly. If every real step of a kernel is reduced to those operations, then the output bytes are a function of the input bytes alone, independent of adapter, workgroup schedule, or compiler reassociation. The hard part stops being the arithmetic and becomes the discipline of staying inside the subset, pinning the source bytes, and building a harness that can tell a genuine divergence from a partial or stale run.

- A declared integer subset with a written boundary: only i32/u32 arithmetic, bitwise operations, comparisons, bitcast between i32/u32, and host-shared little-endian layout are relied upon; all f32 values are transported and decoded as opaque u32 bit patterns, and no float arithmetic, conversion, or bitcast through float occurs in K1 to K4.
- Five kernels: K1 exact f32 summation by a 10-limb (320-bit) integer superaccumulator; K2 Philox4x32-10 counter RNG against published Random123 vectors; K3 Q16.16 256x256x256 GEMM with a 3-limb (96-bit) accumulator and a single arithmetic floor shift; K4 integer SHA-256 over a fixed 1 MiB message; and K5, an f32 matmul negative control that carries no pass criterion, no pinned hash, and no recorded outcome in this artifact.
- A CPU-only permanent gate that reproduces every pinned reference hash and count, runs the known-answer tests, pins the WGSL source bytes, and scans the integer kernels for float types, atomics, discard, and unmasked dynamic shifts.
- A cross-adapter manifest spec: per-record adapter identity, dispatch shape, output bytes and raw-byte SHA-256, status count and flags, compile errors, and outcome, serialized canonically and hashed.
- An explicit negative control and an explicit honesty rule: K5 is an f32 matmul with no pass criterion, no pinned hash, and no recorded outcome in this artifact; the gate checks references and vectors, the browser lab is the demonstration, and no CI artifact may be cited as evidence of GPU agreement.
- A limitations list that names non-finite inputs, out-of-range wrap, the one-machine sample, self-reported adapter identity, machine-checkable exceptions, the absence of any recorded cross-adapter comparison, and the non-claim about general float.

> *The claim ceiling — REPROGPU is a browser/WebGPU library and cross-adapter conformance harness that pins byte-identical SHA-256 outputs for a declared integer/fixed-point WGSL kernel subset — exact f32 summation by a 320-bit integer superaccumulator, Philox4x32-10, Q16.16 GEMM, and integer SHA-256 — and declares general floating-point results out of scope, on the specification's own boundary: WGSL §15.7 leaves rounding direction, reassociation, fusion, and subnormal flushing unspecified. K5 is an f32 negative control whose per-adapter hash and run-to-run comparison are recorded by the browser lab and classified as divergent, run-to-run only, or agreeing; it has no pass criterion and no pinned hash, no K5 outcome is recorded in this artifact, and K5 agreement on any hardware set would not be evidence of float reproducibility. This is an artifact and engineering contribution, not a new method: exact integer execution that is hashed and compared on CPU and GPU is already demonstrated live (the Coherence Energy Labs gpu-exact demo), bit-exact browser WebGPU integer runtimes already ship (bitgpu), and the numerical science underneath — reproducible accumulation, counter RNG, exact integer GEMM — is mature. What this paper offers is the narrow package around that idea: a multi-kernel suite with pinned reference hashes, a*

> *cross-adapter (vendor-to-vendor) manifest protocol, and a gate that states what it cannot prove. No cross-adapter comparison is recorded in this artifact either: the word cross-adapter in the title names the protocol the harness implements and the claim a reader can test by running the lab on two machines; the repository holds CPU reference outputs only, and every cross-adapter statement in this paper is an observation about one time-stamped machine, never a population claim. No claim of novelty, priority, or a new capability is made.*

2. Related work, stated honestly

The adversarial prior-art search returned GO-WEAK, and the verdict is taken as given. The headline capability - making a documented subset of GPU compute bit-reproducible across vendors - is a mature, named category. ReproBLAS defines the binned accumulator for order-independent float summation; ExBLAS combines floating-point expansions with a Kulisch superaccumulator and already reports bit-perfect reductions and GEMM on NVIDIA and AMD GPUs; the Ozaki scheme decomposes GEMM into error-free integer and modular pieces and runs on Tensor Cores; and NVIDIA's own CCCL ships a reproducible floating-point accumulator that makes gpu_to_gpu reductions identical across different NVIDIA GPUs. Counter-based RNG is a 2011 design already in cuRAND, rocRAND, and NumPy; deterministic training and inference is covered by RepDL and Hawkeye; reproducible BLAS on CPUs is oneMKL's conditional numerical reproducibility; and the WebGPU CTS is already a cross-vendor, normative test suite. None of that is new here, and no row of T1 is a strawman.

The remaining clause is narrow and is a delivery delta rather than a scientific one: no located project pins cross-adapter (vendor-to-vendor) byte-identical output hashes for a declared integer and fixed-point WGSL kernel set and ships the reusable harness and manifest that produce them. The exact integer-execution idea itself is already demonstrated in a browser: the Coherence Energy Labs gpu-exact demo runs one fixed-point kernel on the CPU and a WebGPU i32 shader on the GPU and shows the same SHA-256, with the same integer-exact/float-drifts scope note this paper draws; bitgpu ships bit-exact browser WebGPU integer execution, though as an application runtime rather than a conformance protocol. What is not located is the multi-kernel, pinned-reference-hash, cross-adapter manifest protocol around that idea. The WebGPU CTS tests accuracy within tolerances, not bit-exactness; the zkML line proves properties of a computation rather than reproducing its exact bytes; and the PAradigm brief that recommends exactly this integer-only-core-plus-conformance-matrix design is a research note with no code and no hashes. T1 states the delta row by row, including the motivating counter-pressure that bit-exactness across GPU architectures is not the norm. Every URL in T1 was read by the prior-art pass on the dates recorded there; a change to a cited URL or a cited WGSL section number invalidates the corresponding statement and must be re-checked.

Work	URL	What it does	Clauses not done

Work	URL	What it does	Clauses not done
ReproBLAS / binned summation (TOMS 2020)	https://doi.org/10.1145/3389360	Bitwise order-independent float sum, dot, and nrm2 via a 6-word binned accumulator; handles over- and underflow reproducibly	GPU/WebGPU port; conformance harness; GEMM absent from the stable release
ExBLAS (NRE 2015; Parallel Computing 49)	https://doi.org/10.1016/j.parco.2015.09.001	Five-level floating-point expansion plus a Kulisch superaccumulator; bit-perfect dot, GEMV, TRSV, and GEMM on CPUs and on NVIDIA and AMD GPUs	Same-vendor GPUs, not WebGPU; no published cross-adapter hashes; no browser harness
Ozaki scheme / OzBLAS (IJHPCA 2024)	https://dl.acm.org/doi/abs/10.1177/10943420241239588	Error-free integer/modular decomposition gives bit-level reproducible FP64 GEMM across CPU and GPU, including a Tensor-Core integer path	WebGPU; browser; float64 is absent from WGSL; harness
Ozaki-II (arXiv 2504.08009, 2025)	https://arxiv.org/abs/2504.08009	Round-correct, tunable reproducible FP64 matrix multiplication	WebGPU; browser; declared-kernel conformance artifact
NVIDIA CCCL / CUB gpu_to_gpu RFA (2026-03-05)	https://developer.nvidia.com/blog/controlling-floating-point-determinism-in-nvidia-cccl/	A reproducible floating-point accumulator makes reductions identical across different NVIDIA GPUs; default run-to-run determinism	NVIDIA-only, not cross-vendor; reductions only; not in a browser
NVIDIA framework-reproducibility issue #28	https://github.com/NVIDIA/framework-reproducibility/issues/28	Documents that bit-exactness is not expected across GPU architectures	Motivating counter-pressure only; no kernel library and no harness
RepDL (arXiv 2510.09180, 2025)	https://arxiv.org/abs/2510.09180	Bitwise-reproducible deep-learning training and inference across CPU and GPU via correct rounding plus order invariance	CUDA-centric; not browser/WebGPU; heavy framework; no cross-adapter hash table

Work	URL	What it does	Clauses not done
Hawkeye (arXiv 2603.20421, 2026)	https://arxiv.org/abs/2603.20421	Reverse-engineers NVIDIA Tensor-Core rounding and accumulation and replays it bit-exactly on CPU	One vendor; reproducibility by simulation rather than portable kernels
DAB (MICRO 2020) / GPUDet (ASPLOS 2013)	https://microarch.org/micro53/papers/738300a981.pdf	Deterministic atomic buffering and the first fully deterministic GPU architecture; hardware ordering of floating-point atomics	Requires new silicon; no portability guarantee and no browser path
Intel oneMKL conditional numerical reproducibility	https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-1/obtaining-numerically-reproducible-results.html	Bitwise run-to-run reproducibility under fixed threads and code path; strict mode for GEMM and TRSM	CPU only; conditional on build and thread configuration; not cross-vendor
Counter-based RNG: Philox / Random123 (2011)	https://github.com/DEShawResearch/random123	A fixed key and counter give an identical integer stream on CPU, GPU, and FPGA; shipped in cuRAND, rocRAND, and NumPy	No WGSL implementation; no cross-adapter stream hash and no harness
PyTorch / TensorFlow determinism switches	https://docs.pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html	Same-hardware and same-software determinism; raises an error when only a nondeterministic kernel exists	Explicitly not cross-platform or cross-vendor; not a browser runtime
zkML / GZKP (EuroSys 2024; survey arXiv 2502.18535)	https://dl.acm.org/doi/10.1145/3627703.3650088	Zero-knowledge proofs of machine-learning inference, with GPU-accelerated provers	Proofs, not bit-exact reproduction; not WebGPU

Work	URL	What it does	Clauses not done
SAGE software attestation for GPUs (USENIX ATC 2023)	https://www.usenix.org/system/files/atc23-ivanov.pdf	Verifiable untampered kernel execution on an NVIDIA GPU	Attests code integrity, not numeric reproducibility; no browser path
WebGPU CTS floating-point primer	https://chromium.googlesource.com/external/github.com/gpuweb/cts/+/07f15b8e1e73c5cc52d94ab9916a59c04f9c95ec/docs/fp_primer.md	The normative cross-vendor conformance suite; tests floating-point accuracy within ULP and absolute tolerances	Tests tolerances, not bit-exact cross-adapter output hashes
PAradigm cross-vendor determinism brief	https://github.com/11vated/PAradigm-reference/blob/main/research/001-gpu-determinism-cross-vendor.md	States that WGSL is not bitwise deterministic and recommends an integer-only core plus a conformance matrix	A recommendation, not an implementation; no code and no hashes
Coherence Energy Labs gpu-exact demo	https://demos.coherenceenergylabs.com/gpu-exact/	Runs one fixed-point reaction-diffusion kernel in JavaScript on the CPU and as a WebGPU i32 compute shader on the GPU, SHA-256 hashes both, and shows them bit-identical, with a live tamper test and an explicit integer-exact/float-drifts scope note	A single CPU-versus-GPU pair on the visitor's machine, not cross-adapter vendor-to-vendor; no pinned reference hashes, no multi-kernel suite, no manifest format, no CI gate
bitgpu (GitHub)	https://github.com/stfurkan/bitgpu	Dependency-free WebGPU runtime for 1-bit LLMs in the browser; greedy output and KV-cache reuse are gated bit-exact on real hardware with a headless GPU verification gate	An application runtime, not a declared integer-kernel conformance protocol; no cross-adapter hash manifest, no exactness-boundary kernel suite, no reference-hash pins
WgPy (arXiv 2503.00279, 2025)	https://arxiv.org/abs/2503.00279	A NumPy-like WebGPU/WebGL array library in the browser, with fast matmul	No determinism or reproducibility claim and no harness
hash-shader (GitHub)	https://github.com/RustyBamboo/hash-shader	A SHA-256 compute shader written in integer WebGPU ops	Not a reproducibility harness and publishes no cross-adapter hash table

Work	URL	What it does	Clauses not done
BitExact (GitHub)	https://github.com/aaravkohli1/BitExact	Deterministic CUDA matmul, RMSNorm, and reductions with a PyTorch drop-in	CUDA/NVIDIA only; not a browser and not cross-vendor
WGSL CRD 2026-09-15 (specification)	https://www.w3.org/TR/2026/CRD-WGSL-20260915/	Defines the exactness boundaries this paper relies on: wrapping integer arithmetic, bitwise ops, comparisons, bitcast, little-endian host-shared buffers	Leaves floating-point rounding direction, reassociation, fusion, and subnormal flushing unspecified

T1. Closest work and the exact delta (static; no REPROGPU numbers). Every row states the mechanism that is known elsewhere and the clause that is not done there. No row is a comparison against a strawman, and none of these works was run in our gate. The exact integer-execution idea is not new, and the CPU-versus-GPU hash-equality demonstration already exists: the Coherence Energy Labs gpu-exact demo runs one fixed-point kernel in JS on the CPU and as a WebGPU i32 shader on the GPU, SHA-256s both, and shows them bit-identical. The clause not located there is narrower: a reusable, multi-kernel conformance protocol that pins byte-level reference hashes for a declared integer/fixed-point WGSL subset and collects cross-adapter (vendor-to-vendor, not CPU-versus-GPU-on-one-machine) manifests, with a gate that states what it does not prove. The science of reproducible accumulation is mature; the artifact is a narrow combination, not an unprecedented capability.

Correction to the prior-art record. The first prior-art pass for this wave did not locate the Coherence Energy Labs gpu-exact demo or bitgpu, and an earlier draft of the abstract and of this section described the capability as unmatched without them; both were added after publication and are now rows in T1. With them, the honest description of the contribution is a narrow systems/artifact package - a declared exactness boundary, a multi-kernel suite with pinned reference hashes, a cross-adapter manifest, and a gate that publishes what it cannot prove - on top of an idea (exact integer execution, hashed and compared) that is already demonstrated elsewhere. The correction is recorded in docs/research/wave-48-priorart-reprogpu.md rather than silently patched.

3. The WGSL exactness audit and the declared boundary

The subset is chosen from the specification, not from benchmarking. The following are guaranteed portable and exact, and the kernels are built only from them. Expressions on concrete integer types that overflow produce a result modulo 2^bitwidth (section 6.2.3), and i32 is two's complement, so i32 and u32 addition, subtraction, multiplication, and unary negation all wrap exactly. Bitwise and, or, xor, and complement are exact and component-wise (section 8.10). Comparisons and select are exact (section 8.9). Shifts are specified as the shift value modulo the bit width of the left operand, so u32 right shift is logical and i32 right shift is arithmetic (section 8.10). Bitcast between i32 and u32, and between their vectors, is an exact bit reinterpretation (section 17.2.1 with layout in section 14.4.4). Numeric values in host-shared buffers are stored little-endian (section 14.4.4), so readback can be byte-addressed.

The following are excluded because the specification leaves them implementation-defined. All f32 and f16 arithmetic is out, because section 15.7.4.1 gives the rounding direction as either neighbour for add, subtract, and multiply, because division is 2.5 ULP rather than correctly rounded, and because section 15.7.5 explicitly permits reassociation and fusion, so a*b+c may or may not contract

to fma and fma itself may expand. Subnormal inputs and outputs may be flushed to zero, the sign of zero may be ignored, and runtime overflow, infinity, and NaN may become an indeterminate value (section 15.7.2, the Finite Math Assumption). Transcendentals, determinant, smoothstep edges, and out-of-bounds indexing are likewise excluded. Mapping a floating-point value to an integer is also not exact (section 15.7.6), so K1 never converts an f32 value; it only decodes its bit pattern as u32.

Declared boundary. Only integer (i32/u32) and bit operations, bitcast between i32/u32, comparisons, and host-shared buffer layout are relied upon. In K1–K4 all f32 values are transported and decoded as opaque u32 bit patterns; no floating-point arithmetic, conversion, or bitcast-through-float occurs. All floating-point results are out of scope for the reproducibility guarantee.

> *No i64 in WGSL — WGSL has no i64 or u64 integer type, so every accumulator is multi-limb u32 with carries propagated in a fixed limb order. K1 uses 10 limbs (320 bits) because a single term spans 277 bits and up to 2^24 terms must be summed without wrap; K3 uses 3 limbs (96 bits) because the seeded row sums stay below 2^70. The exact error-free products of K2 and K3 are built from 16-bit halves, never from a wider integer type. A per-limb atomicAdd cannot carry between limbs, so atomics are banned for every accumulator and reduction uses private accumulators plus a fixed tree. Any larger shape requires re-deriving the bound before the pin is accepted.*

4. Kernels K1-K5

Every kernel follows the same discipline. It reads only u32 or i32 words, performs only exact operations, and writes a status word first. Multi-limb integer addition is associative and exact on bounded values, so any workgroup schedule produces identical limbs; this is a proof, not an empirical result, and the fixed reduction tree is pinned anyway so that the byte serialization is recomputable without depending on a schedule. No kernel uses discard, an unmasked dynamic shift count, a floating-point operation, or atomic. Every input buffer is pre-filled with the sentinel 0xDEADBEEF and every output begins with a count canary, so a partial run or a stale buffer is detectable rather than silently hashed. The five kernels and their pinned reference outputs are below.

Kernel	Inputs	Algorithm	Output layout	Count	Pinned SHA-256
K1 exact f32 sum	4,096 LCG u32 binary32 patterns (seed 0x2545F491, exponent 0xFF skipped) plus 8 specials = 4,104 words	Decode sign, exponent, mantissa; subnormals at shift 0; non-finites set sticky flags and are skipped; sum in a 10-limb (320-bit) two's-complement integer superaccumulator	12 u32 little-endian: A[0..9], flags, n_seen; 48 bytes	4,104	c046384ed6580b8111b8ef34f5d0932289209f4920666477bc76ffe724dfa27a
K1-flag	The K1 word list plus 0x7F800000 (+Inf) and 0x7FC00000 (NaN)	Same accumulator; non-finites contribute zero and set flags bit 0; the gate treats this vector as an out-of-domain flag test, never as an exact-sum oracle	12 u32 little-endian: A[0..9], flags, n_seen; 48 bytes; flags must read 1	4,106	cc7dca6d69fec7162cf610b0170076aec74d3d5bec47b2ea80d0b74afb3c9ecb

Kernel	Inputs	Algorithm	Output layout	Count	Pinned SHA-256
K2 Philox4x32-10 stream	Fixed key (0x12345678, 0x9ABCDEF0); block i uses counter (i, 0, 0, 0), 65,536 blocks	Random123 Philox4x32-10; 32x32 to 64 mulhilo by 16-bit halves; exactly 10 rounds; key bumps W0/W1 after rounds 1 to 9	u32 little-endian: [n_blocks=65536, then 65,536 x 4 RNG words]; 1,048,580 bytes	65,536	f9004cf3ed50db5cdb6351a3eb090a498586ade2e3f7499744c951bf77f86d01
K3 Q16.16 GEMM 256x256x256	A and B are 256x256 i32 row-major, Q16.16, seeded from LCG 0x1234ABCD; each raw value in [-2^20, 2^20)	Exact 64-bit products from 16-bit halves accumulated in a 3-limb (96-bit) two's-complement accumulator over k = 0..255; one arithmetic right shift by 16 at the end; low 32 bits wrap	256x256 i32 row-major plus status [256,256,256, n_ok]; 262,160 bytes	65,536	fbe13a3b4e2fc7bf3a4b38c838912651151c03460bd9d1d7f1ea81190af3932e
K4 integer SHA-256	1 MiB message (2^20 bytes; byte i = LCG state_{i+1} >> 24, seed 0x2545F491), padded to 16,385 blocks	FIPS 180-4 SHA-256 in u32; 64 constants; rotr message schedule; big-endian block load; final H words byte-swapped so the 32 bytes after the count equal the standard big-endian digest	9 u32 little-endian: [n_blocks=16385, H0..H7]; 36 bytes	16,385	eea119f4059566875ceaf784400d519fd8ff2148d9f221e0004fd04944a21797
K5 f32 negative control	256x256 f32 matrices with a wide exponent range	acc = acc + A[i,k]*B[k,j] for k = 0..255; no reduction-tree and no fusion promise	256x256 f32 row-major; 262,144 bytes	65,536	(none: no pass criterion)

T2. The five kernels as shipped, with pinned reference output hashes. Counts are the status canary the harness compares: for K1 the full consumed word count including the eight appended specials (4,104), for K2/K3/K5 the block or element count (65,536), and for K4 the block count (16,385). K5 is the float negative control: it has no pass criterion, so its hash column is empty, and no K5 outcome is recorded anywhere in this artifact. The pinned hashes are the CPU reference outputs; the gate recomputes them and the browser lab compares live GPU outputs against them.

```
                K1 limb placement (no 64-bit intermediate)
term = (-1)^s * M * 2^shift, with M = m for a subnormal (shift = 0) and
M = 0x800000 | m for a normal (shift = e - 1); let b = shift mod 32 and
 L = floor(shift / 32); if b == 0 add M at limb L, otherwise add (M <<
    b) & 0xFFFFFFFF at limb L and M >> (32 - b) at limb L + 1; the
 accumulator is A[0..9] with value sum over i of A[i] * 2^(32i), and a
negative term is negated by inverting all ten limbs and adding one with
  a carry from limb 0. Terms and partials are then added with carries
                  propagated from limb 0 to limb 9.
```

Bound: a term is below 2^277 and N is at most 2^24, so the magnitude stays below 2^301, inside the 320-bit two's-complement range. The reference is computed in BigInt as A mod 2^320 and

reduced to ten u32 limbs; the permanent gate pins its serialization and recomputes it on every run.

```
            K3 exact accumulation and the single floor shift
   S = sum over k of u32(A[i,k]) * u32(B[k,j]) mod 2^96, accumulated
       exactly as a 3-limb two's-complement value; result32 = (S
 arithmetic-right-shift 16) & 0xFFFFFFFF; the stored i32 is the low 32
bits and wraps modulo 2^32 when the floored value leaves the i32 range
(section 6.2.3). The k loop is fixed at k = 0..255 and the accumulator
          is 96 bits because |product| < 2^62 and |S| < 2^70.
```

The seeded vectors draw each raw value from [-2^20, 2^20), so |product| < 2^40 and each 256-term row sum is below 2^48, comfortably inside the 96-bit bound; the wider spec bound is the design ceiling, not the pinned vector range. Wrap is documented, never presented as a correct result.

> *Why floor once — K3 accumulates every product exactly and rounds exactly once, at the end, by an arithmetic right shift of 16 bits, which is floor toward negative infinity. Truncation is preferred over round-half-up because it is a single unambiguous bit operation that matches BigInt right shift, because round-half-up would need a tie rule with no basis in the specification, and because rounding each term is unnecessary once the accumulator is exact. The low 32 bits then wrap per section 6.2.3 whenever the floored value leaves the i32 range. The seeded pinned vectors do not reach that case — their row sums stay below 2^48 inside a 96-bit accumulator — so the shipped K3 result contains no wrap; wrap is a property of the design ceiling for larger shapes and is stated here rather than left implicit.*

5. The cross-adapter conformance harness

The harness runs each kernel on each available adapter, reads the output buffer back, hashes the raw bytes, and emits a manifest. Its per-record fields are fixed: kernel, kernelVersion, wgslSha256, adapter identity (vendor, architecture, device, description, features, and a subset of limits), dispatch shape, output byte length, output SHA-256, status count and flags, duration, timestamp, user agent, compile errors, and an outcome of pass, fail, exception, or skip. The source hash is shared: all adapters run the same module bytes, the manifest records wgslSha256 once per kernel, and the gate pins it.

The manifest record (frozen shape)

```
export interface ManifestRecord {
  kernel: KernelId;                // K1 | K2 | K3 | K4 | K5
  kernelVersion: number;
  wgslSha256: string;              // SHA-256 of the module bytes, CRLF -> LF
  adapter: AdapterInfo;            // vendor, architecture, device, description,
features, limits
  dispatch: { workgroups: number[]; workgroupSize: number[] };
  outputBytes: number;
  outputSha256: string;            // SHA-256 of the raw readback bytes
  status: { count: number; flags: number };
  durationMs: number;
  timestampISO: string;
  userAgent: string;
  compileErrors: string[];
  outcome: "pass" | "fail" | "exception" | "skip";
}
```

```
// Canonical manifest JSON: UTF-8, keys sorted, arrays sorted by kernel then
// adapterKey (vendor|architecture|device|description), no insignificant
// whitespace; manifestSha256 = SHA-256(serialized manifest) is included in the
// artifact and re-derived by the gate.
```

Readback is byte-exact by construction: copyBufferToBuffer, then mapAsync in read mode, then copy to a fresh Uint8Array and unmap, and hash the raw bytes rather than a typed-array numeric view, because a Uint32Array view would make the hash depend on host endianness. The sentinel and canary rules close the silent-no-run hole: every input buffer is pre-filled with 0xDEADBEEF, every output begins with a count word, the harness rejects a kernel whose output equals the sentinel or whose count differs from the pinned expected count, createShaderModule is followed by a compilation-info check that must report zero errors, validation and out-of-memory error scopes are pushed, and onSubmittedWorkDone is awaited. A mismatch that matches a pre-declared entry of exceptions.json becomes an exception with the adapter identity, kernel, observed and expected hash, reason, and an evidence URL; anything else is a fail. A compile or limit failure is a skip with the reason recorded, and if every GPU skips, the demonstration is empty and the paper must say so.

The pass rule for K1 to K4 is exact: every non-skipped record passes if and only if outputSha256 equals the pinned expected hash for that kernel, the status count equals the pinned expected count, and compileErrors is empty. K5, the float negative control, has no pass criterion at all. Its per-adapter hash is recorded, a second dispatch gives a run-to-run measurement, and the run is classified as divergent, run-to-run only, or agreeing. If every adapter agrees, that is recorded as an observation about that hardware set and this one time, with no generalization to floating point. The gate checks none of this, because it has no GPU, and no K5 record is committed to this repository: a reader who runs the lab is the first to observe it, which is why the abstract and the claim ceiling describe K5 as a control rather than as a demonstration.

> *The honesty rule — The gate verifies the reference implementations and the vectors; the browser lab is the demonstration of cross-adapter reproducibility. No CI artifact may be cited as evidence of GPU agreement.*

6. Experiments and known-answer tests

The experiments here are deterministic and CPU-only. They establish the reference side of the claim: that the shipped TypeScript references reproduce the pinned hashes and counts, that the integer implementations match external known-answer tests, that the WGSL source bytes are pinned, and that the integer kernels stay inside the declared subset. The cross-adapter side is not an experiment in this paper; it is the browser lab of section 8, and the only honest statement about it is that a local manifest is a small, time-stamped, self-reported sample. The gate runs in about one second on the authoring machine, and every number in T3, T4, and T5 is re-derived on each run with no auto-update path.

Test	Vector	Expected	Oracle

Test	Vector	Expected	Oracle
Philox KAT 1	ctr 00000000 00000000 00000000 00000000; key 00000000 00000000	6627e8d5 e169c58d bc57ac4c 9b00dbd8	Random123 kat_vectors
Philox KAT 2	ctr ffffffff ffffffff ffffffff ffffffff; key ffffffff ffffffff	408f276d 41c83b0e a20bc7c6 6d5451fd	Random123 kat_vectors
Philox KAT 3	ctr 243f6a88 85a308d3 13198a2e 03707344; key a4093822 299f31d0	d16cfe09 94fdcceb 5001e420 24126ea1	Random123 kat_vectors
SHA-256 KAT, empty message	"" (0 bytes)	e3b0c44298fc1c149afbf4c8996f b92427ae41e4649b934ca49599 1b7852b855	FIPS 180-4
SHA-256 KAT, abc	"abc" (3 bytes)	ba7816bf8f01cfea414140de5da e2223b00361a396177a9cb410ff 61f20015ad	FIPS 180-4
K4 oracle digest	1 MiB LCG message, seed 0x2545F491	79bfb41c6346bc10d842265e62 eb4e35ce77ac3bc2d3c21ebe06 8d808acc1361	Node/Web Crypto SHA-256 oracle, pinned in tests/reprogpu.test.ts

T3. Known-answer tests as the gate runs them. The Philox lines are the published Random123 kat_vectors values, word-compared in order (c0, c1, c2, c3). The SHA-256 lines are FIPS 180-4 vectors. The K4 digest is the SHA-256 of the exact 1 MiB LCG message; it is pinned from an independent Node/Web Crypto oracle in the test suite (tests/reprogpu.test.ts), while the permanent gate recomputes the shipped reference and compares it to the frozen digest. The frozen artifact carries these two FIPS vectors; the longer NIST 448-bit and 896-bit vectors named in the amended spec were not added to the shipped test list.

#	Criterion	Status	Detail as printed
1	References reproduce pins	PASS	K1/K1-flag/K2/K3/K4 all match
2	Known-answer tests	PASS	3 Philox KATs + FIPS SHA-256 + K4 oracle match
3	WGSL source pins	PASS	all five sources match the pins
4	Integer kernel purity	PASS	no float/atomics/discard; shifts masked; canaries present
5	Summary	PASS	all criteria passed

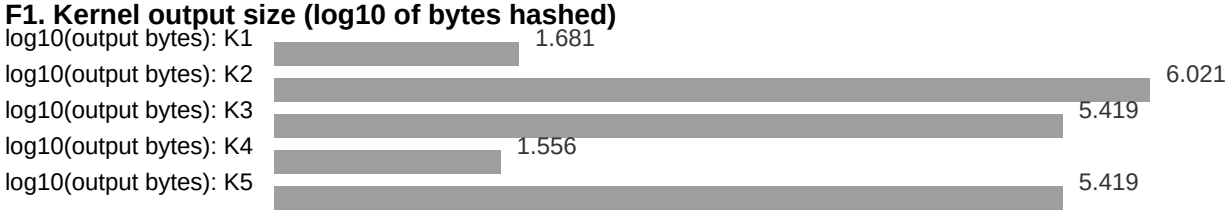
T4. Permanent gate criteria (scripts/verify-reprogpu.ts). The gate is pure deterministic TypeScript on the CPU. It exits 1 on any failure and prints the final line REPROGPU_GATE {"passed":5,"failed":0,"runtimeMs":1049}. Runtime is wall time on the authoring machine and will vary; the pinned content is the pass/fail result, not the millisecond count.

Kernel	Pinned count	Output bytes	Reference output SHA-256
K1	4,104	48	c046384ed6580b8111b8ef34f5d0932289209f4920666477bc76ffe724dfa27a
K1-flag	4,106	48	cc7dca6d69fec7162cf610b0170076aec74d3d5bec47b2ea80d0b74afb3c9ecb
K2	65,536	1,048,580	f9004cf3ed50db5cdb6351a3eb090a498586ade2e3f7499744c951bf77f86d01
K3	65,536	262,160	fbe13a3b4e2fc7bf3a4b38c838912651151c03460bd9d1d7f1ea81190af3932e

Kernel	Pinned count	Output bytes	Reference output SHA-256
K4	16,385	36	eea119f4059566875ceaf784400d519fd8ff2148d9f221e0004fd04944a21797

T5. Pinned reference outputs: counts, sizes, and hashes. These are the values in docs/research/reprogpu/expected-hashes.json, re-derived byte-for-byte by the gate. Output bytes is the length of the byte serialization that is hashed. K1-flag has no separate count pin in the artifact; its count is 4,104 plus the two appended non-finite words, and only its hash is pinned. The WGSL source pins are listed in section 9.

F1. Kernel output size (log10 of bytes hashed)



Output buffer sizes span five orders of magnitude, so the bars are log10 of the byte length and the transform is labelled here. K1 is 48 bytes (12 u32), K2 is 1,048,580 bytes (the count word plus 65,536 Philox blocks of four u32), K3 is 262,160 bytes (65,536 i32 plus a four-word status), K4 is 36 bytes (the block count plus eight digest words), and K5 is 262,144 bytes (65,536 f32). Size is an implementation property, not a measure of importance: the hardest pin to reproduce is the 36-byte K4 digest.

What the known-answer tests do and do not buy. The three Philox lines and the two FIPS SHA-256 lines are external oracles: they were not produced by this codebase, so agreement is evidence that the integer algorithms are the standard ones rather than that the pins are self-consistent. The K4 message digest is checked against a Node/Web Crypto SHA-256 oracle on the exact 1 MiB LCG message in the test suite. The K1 reference is an exact BigInt reconstruction; the K3 reference is compared against a naive O(n^3) BigInt implementation on small seeded sizes in the test suite. The pins themselves are regression anchors: the same reference computes both the expected constant and the observed value, so a pin can only catch drift, and correctness rests on the external oracles and on the exactness argument, not on the pin.

> *What was not run in CI — No WGSL compilation, no GPU execution, no dispatch, no readback, and no cross-adapter equality check ran in CI. The gate is pure TypeScript on the CPU and cannot open a GPU device. The browser lab at /reprogpu is the only place a cross-adapter manifest is produced; it runs on the reader's machine, records adapter identity that the adapter self-reports, and is not cryptographically attested. A CI green therefore means the references, the known-answer tests, the source pins, and the purity scan agree, and it means nothing about any GPU.*

7. Failure cases & limitations

The limitations below are the amended spec's, quoted verbatim. Each names a real envelope of the shipped system or of the experiment, not a caveat added after the fact.

> *Four limitations that apply to every number in this paper — (1) No human-participant data, and none could be relevant here: the object is a GPU kernel and its output bytes, and no experiment in Sections 3 to 8 involves a person, a learner, or a graded run. (2) No external replication. The cross-adapter manifest is produced on a reader's machine and is self-reported and unattested; no committed artifact records a second machine, a second vendor, or a second browser, so no result here has been shown to hold on hardware other than the ones a reader supplies. (3) Recomputed against replayed. The*

> *gate recomputes the five reference outputs, the three Philox known-answer tests, the two FIPS vectors, and the five WGSL source hashes, and compares each with a stored constant. As Section 6 states, the pins are regression anchors: the same reference computes both the expected constant and the observed value, so a pin can only catch drift. Correctness for K1–K4 rests on the external oracles (Random123's published Philox vectors, FIPS 180-4, Node's Web Crypto digest for the K4 message) and on the exactness argument of Section 3, not on the pins. The browser manifest digest is time-dependent by construction and is a run-identity token, not a reproducibility token; only the per-kernel outputSha256 is verifiable offline. (4) No experiment here measures a generative-AI tutor. A browser GPU kernel is not a tutoring interaction; the Bastani et al. generative-tutor field experiment that motivates the wider research agenda is other people's evidence about another system, and nothing in this paper bears on how a generative tutor behaves, teaches, or is graded.*

1. The guarantee covers only K1–K4 and only the shapes/domains declared above; float is out of scope (WGSL CRD 2026-09-15 §15.7.2/§15.7.4/§15.7.5).
2. The gate verifies the references, KATs, pins, and source integrity; it cannot execute WGSL or test cross-adapter equality. The browser lab is the demonstration, not the CI gate.
3. Cross-adapter results are a small, time-stamped sample on one machine; adapter.info is self-reported and not cryptographically attested; engine support is limited and may change.
4. K1 excludes non-finite inputs (flagged and rejected); K3 documents mod-2^32 wrap on out-of-range output, which is a design ceiling rather than something the shipped seeded vector exercises; none of K1–K4 covers general float.
5. Any adapter mismatch is a fail unless it is a pre-declared, reproducible exceptions.json driver bug; exceptions are reported, not hidden.
6. K5 divergence is permitted and K5 agreement is possible; neither outcome is recorded in this artifact, and K5 agreement on any hardware set is not evidence of general float reproducibility.

8. Product: the /reprogpu lab

The product is a single browser page at /reprogpu. It requests a WebGPU adapter, renders an adapter card with the self-reported vendor, architecture, device, and description, and lets the reader run the five kernels. For K1 to K4 the page shows the pinned expected hash next to the live output hash and the status count next to the expected count, so a reader sees pass or fail directly rather than a summary. For K5 the page shows the per-adapter hash and the second-dispatch comparison, classified as divergent, run-to-run only, or agreeing, with copy that says agreement is an observation and not a reproducibility claim. The run produces the full manifest of section 5, which the reader can copy or download; the page re-derives manifestSha256 so a third party can compare it against the committed artifact. The run is local: inputs never leave the device and only output hashes are publishable.

- Ships: the adapter card, the per-kernel expected-versus-observed hash table with counts, the K5 negative-control panel, and a copy/download manifest action with a re-derivable manifestSha256.
- Ships: an always-visible statement of the declared boundary and of what the gate cannot do, placed next to the run control rather than hidden in a footnote.
- Ships: a link to the committed expected-hashes.json and to the pinned WGSL source hashes, so a reader can compare a downloaded manifest without trusting the page.
- Cut: any claim that a matching hash proves the adapter or the driver is correct; any claim that

adapter info is an attestation; any extension of the integer result to float; any server call, storage, account, or telemetry.

> *What the lab must never claim — Never say a hash match proves a GPU is correct, only that this kernel returned the pinned bytes under the recorded rules. Never call adapter.info an attestation; it is a self-reported string that identifies a run. Never say integer determinism extends to general float, and never present K5 agreement as evidence of float reproducibility. Never say the CI gate executed WGSL or compared adapters. Never hide a fail, a skip, or an exception behind a summary, and never treat an exceptions.json entry as a pass.*

9. Reproducibility

Everything reported here is offline, deterministic, and dependency-free. The evidence script and the gate are pure TypeScript and use no clocks, no randomness, no network, and no GPU. The evidence script computes the reference outputs, hashes their byte serializations and the WGSL sources, and writes docs/research/reprogpu/expected-hashes.json; it never writes expected.ts, so there is no auto-update path. The gate recomputes every value and compares against the committed constants; drift fails the gate until the constant and this paper are changed together in one reviewed commit. Scratch paths are not recorded and no absolute path enters the artifact.

Reproducing the reported results

```
# evidence: reference outputs, KATs, WGSL source hashes;
# writes docs/research/reprogpu/expected-hashes.json
bun run scripts/reprogpu-evidence.ts

# permanent gate: 5 criteria, pinned hashes, KATs, source pins, purity scan
bun run verify:reprogpu

# paper and reference checks
bun test tests/reprogpu*.test.ts
```

The pinned WGSL source hashes are the values the gate compares after normalizing CRLF to LF and stripping any byte-order mark. They are the same for every adapter in a run, and they change whenever a kernel source changes, which is the point: a kernel edit must be accompanied by a deliberate pin update. The five pins, in K1 to K5 order, are:

- K1: ea7c46670ea2a3258b7522db6a8b09af1f2e7895021870a32669ca7ec9b2df3f
- K2: 010e4d4db8cc43cf95f9d4329da09d5bd3a4d3d764c20d63490df81c6c0d4dbe
- K3: 1013c964ff0a25721558503d391bd6630e96f848973c7198668dfcd14988132d
- K4: d9d8db073c560de60fead78f846afbb4feec2b2626d2d74c072ed6a1c6b23489
- K5: 3f003ca3d1bba17b2052b5cba08a42211eba3fae742177ad52d9f109541e3cbe

No result in this paper depends on a cited URL resolving to the same content it held on 2026-09-19. The WGSL section numbers, the Random123 vectors, and the FIPS vectors are quoted with their sources; if any cited specification section or URL changes, the corresponding kernel, vector, or claim must be re-checked and the pins re-frozen before the paper is treated as current. The evidence script's runtime was 1.08 seconds in the recorded run, with the 256^3 BigInt K3 reference taking 833 milliseconds of it; these are machine-local timings and are not part of any pin.

10. Future work

The subset is deliberately small, and each of the following extensions keeps the same discipline of declaring the boundary first and pinning the evidence. None of them changes the claim of section 1.

- WASI and WebGPU compute reuse: run the same integer kernels under a WASI runtime backed by WebGPU so a non-browser host can reproduce a browser result without a second implementation.
- More kernels inside the declared subset: integer sorting by deterministic sample and bucket passes, exact integer dot products over binned lanes, and additional counter-RNG streams such as Threefry-2x32 against their published vectors.
- A cross-language harness: one manifest schema implemented in TypeScript, Rust, and Python so three independent hosts emit the same canonical JSON and the same manifestSha256.
- An attestation envelope in the DEVSIG direction: bind a kernel source hash, an adapter identity, and an output hash into one offline-checkable artifact, while keeping adapter identity clearly distinct from a cryptographically attested driver.
- A wider adapter sample and a committed exceptions register: record reproducible, pre-declared driver divergences with evidence so that a future mismatch is classified rather than silently broadened.
- A formal statement of the accumulator bounds for shapes larger than the fixed ones, so a new shape is admitted only with a re-derived proof rather than by scaling the existing constants.

References

1. Demmel & Nguyen 2020, Parallel reproducible summation, ACM TOMS 46(3). https://doi.org/10.1145/3389360
2. Iakymchuk, Collange, Defour & Graillat 2015, Numerical reproducibility for the parallel reduction, NRE / Parallel Computing 49. https://doi.org/10.1016/j.parco.2015.09.001
3. Ozaki et al. 2024, Error-free transformations for reproducible GEMM on tensor cores, IJHPCA. https://dl.acm.org/doi/abs/10.1177/10943420241239588
4. Ozaki-II: round-correct reproducible matrix multiplication (arXiv 2504.08009). https://arxiv.org/abs/2504.08009
5. NVIDIA CCCL 3.1, Controlling floating-point determinism with a reproducible accumulator (2026-03-05). https://developer.nvidia.com/blog/controlling-floating-point-determinism-in-nvidia-cccl/
6. NVIDIA framework-reproducibility issue #28, bit-exactness across GPU architectures. https://github.com/NVIDIA/framework-reproducibility/issues/28
7. RepDL: bitwise-reproducible deep-learning training and inference (arXiv 2510.09180). https://arxiv.org/abs/2510.09180
8. Hawkeye: reverse-engineering Tensor-Core numerics for bit-exact replay (arXiv 2603.20421). https://arxiv.org/abs/2603.20421
9. Deterministic Atomic Buffering, MICRO 2020. https://microarch.org/micro53/papers/738300a981.pdf
10. Jooybar et al. 2013, GPUDet: a deterministic GPU architecture, ASPLOS. https://people.ece.ubc.ca/aamodt/publications/papers/jooybar.asplos2013.pdf
11. Intel oneMKL developer guide, obtaining numerically reproducible results (CNR). https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-1/obtaining-numerically-reproducible-results.html

12. Salmon, Moraes, Dror & Shaw, Random123 / Philox counter-based RNG. https://github.com/DEShawResearch/random123
13. PyTorch torch.use_deterministic_algorithms documentation. https://docs.pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html
14. TensorFlow tf.config.experimental.enable_op_determinism documentation. https://www.tensorflow.org/api_docs/python/tf/config/experimental/enable_op_determinism
15. zkML: zero-knowledge proofs of machine-learning inference, EuroSys 2024. https://dl.acm.org/doi/10.1145/3627703.3650088
16. Survey of verifiable machine learning (arXiv 2502.18535). https://arxiv.org/abs/2502.18535
17. Ivanov et al. 2023, SAGE: software-based attestation for GPU execution, USENIX ATC. https://www.usenix.org/system/files/atc23-ivanov.pdf
18. WebGPU conformance test suite, floating-point primer. https://chromium.googlesource.com/external/github.com/gpuweb/cts/+/07f15b8e1e73c5cc52d94ab9916a59c04f9c95ec/docs/fp_primer.md
19. W3C WebGPU Shading Language, Candidate Recommendation Draft 2026-09-15. https://www.w3.org/TR/2026/CRD-WGSL-20260915/
20. WGSL CRD 2026-09-15, section 15.7 floating-point evaluation. https://www.w3.org/TR/2026/CRD-WGSL-20260915/#floating-point-evaluation
21. WgPy: a NumPy-like GPU array library for the browser (arXiv 2503.00279). https://arxiv.org/abs/2503.00279
22. RustyBamboo/hash-shader, SHA-256 WebGPU compute shader. https://github.com/RustyBamboo/hash-shader
23. aaravkohli1/BitExact, deterministic CUDA kernels. https://github.com/aaravkohli1/BitExact
24. 11vated/PAradigm cross-vendor GPU determinism brief. https://github.com/11vated/PAradigm-reference/blob/main/research/001-gpu-determinism-cross-vendor.md