Bit-Exact AI Inference Verification Without Performance Tradeoffs

Naci Cankaya [1]

Abstract

Verifying claims about AI workloads is a prerequisite for credible AI governance of covert adversaries (who comply with monitoring only when detection likelihood is high), yet the *apparent* non-determinism of GPU floating-point arithmetic forces auditors to accept approximate output matches. Covert adversaries can exploit unverifiable degrees of freedom in monitored computation. Attack vectors include steganography, unreported modification of inference software, and covert computation via unreported batch elements. Empirically, we analyze how modern inference engines (vLLM, HF transformers) produce deterministic but non-invariant outputs, without needing to set performance-compromising determinism flags, if the right information is available for re-computation and no atomic functions are called in the backend. We demonstrate that such bitwise-precise re-computation does not require access to identical hardware, via a software-only emulation of LLM inference across multiple NVIDIA GPU variants. Thus, accumulated rounding errors can be an auditable *signature* of the software and hardware setup used for inference, instead of a constraint on verifiability. Source code for the emulator is available `here`, and the repository for the empirical studies can be found `here`.

1. Introduction and Related Work

The motivation for this work is verification of the claimed ML computation of a covert adversary (Aumann & Lindell, 2010): an adversary who will exploit gaps in a verification setup while complying only if the monitoring ensures high likelihood of detecting false claims. Such monitoring and verification of covert adversaries describes the threat model of low-trust AI governance, e.g. verification of an international, mutual agreement between rival nations, for restrained AI development (Baker et al., 2025; Harack et al., 2025; Scher & Thiergart, 2024; Wasil et al., 2024). An untrusted owner of AI compute resources (the "prover") can make claims about the computation performed on their devices to an outside party (the "verifier") who chooses claimed computations for retroactive verification. The verifier can check a claim $f(x) = y$ for self-consistency by re-running the computation on their own hardware or via a zero-knowledge proof (Sun et al., 2024) in which the prover demonstrates correctness without revealing the input.

AI accelerators rely on parallel, low-precision computation. This work focuses on the resulting challenge to retroactive verification: rounding errors in $f(x)$ from non-associative summation across parallel reductions (Goldberg, 1991; Shanmugavelu et al., 2024; Yuan et al., 2025). In contemporary inference of AI algorithms, this is perceived as "non-deterministic" noise in outputs, and can create plausible deniability in a verification setup: (Rinberg et al., 2025) describe a steganographic attack vector exploiting unverifiable degrees of freedom on ML outputs for covert communication. There is increasing interest in network taps for ensuring only monitored communication in and out of untrusted servers (Cankaya et al., 2026), but steganography hides even in fully monitored traffic. Furthermore, unverifiable rounding errors may enable covert modification of inference and training software, in an attempt to hide additional computation on a monitored server (e.g. via unreported batch elements and kernel optimization). Statistical verification schemes (Karvonen et al., 2025; Rinberg et al., 2025) can upper-bound the covert bandwidth available to an adversary but cannot close it entirely. Zero-knowledge proofs on LLMs have made significant progress in terms of performance, but require determinism as a precondition (Sun et al., 2024). Prior work has achieved output invariance across batch sizes, using custom kernels, though at the cost of $\approx 20\%$ throughput loss (Thinking Machines Lab, 2025). Deepseek-V4's technical report goes into detail on batch-invariant and deterministic kernel libraries (DeepSeek-AI, 2026). The authors claim minimal overhead across the kernel suite, and negligible overhead specifically for batch-invariant decoding.

Without modifying inference engines with invariance-

[1]Machine Intelligence Research Institute. Correspondence to: Naci Cankaya <naci@intelligence.org>.

Second Workshop on Technical AI Governance Research (TAIGR) @ ICML 2026, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

enforcing kernels[1], we demonstrate bitwise verifiability of activation tensors: **A)** We investigate the root causes of differing rounding errors across software and hardware conditions, and distinguish true non-determinism (atomic functions) from non-invariance (deterministic, but different reduction trees) (Section 2). **B)** Experimentally, we demonstrate deterministic outputs, provided the right conditions are fixed (Section 3). **C)** We predict the precise outputs of LLM inference computed on different generations of NVIDIA accelerators (L40, L40S, A100, H100). For this, we use a software emulation — running entirely on CPU — that models every rounding decision of the GPU hardware and software stack (Section 4).

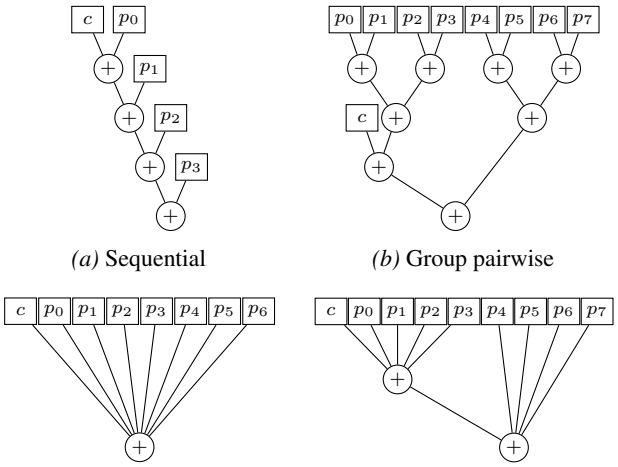
We conclude that modern AI inference engines produce deterministic results, which are bitwise verifiable provided the key factors are known by a verifier, which we isolate and list in Section 6. These factors are predominantly static at inference time (hardware setup, software versions, model weights, parallelism topology), with the exception of dynamic batch size. Recording the concurrent batch size is one additional integer per forward pass to track — negligible overhead for the prover. Reporting this information in addition to outputs removes any plausible deniability in output numerics, collapsing verification into a simple pass/fail. Notably, the numeric sensitivity of outputs is a high-fidelity *signal* for those reported properties, turning non-associativity from a noise source to a fingerprint of the prover's hardware and software stack.

2. Root Causes of Apparent Non-Determinism

Modern AI inference and training make heavy use of parallel computation. We distinguish two classes of arithmetic operations in transformer inference: **element-wise** and **reductions**.[2] Both can be sources of apparent non-determinism, for different reasons.

2.1. Element-Wise Operations

Element-wise operations executed by dedicated hardware acceleration can use non-standard approximations.[3] This is the case for some select operations on NVIDIA accelerators (NVIDIA Corporation, 2025e), and we found that accurately predicting the outputs of transformer FFN and attention blocks requires modeling special function units (SFUs) for exponentials, reciprocals, and square roots. Software is also a source of discrepancy: trigonometric functions in RoPE, for example, depend on the specific math library used (e.g., CUDA libdevice (NVIDIA Corporation, 2025c) vs. CPU libm), and different library versions may produce different results for a fraction of inputs (NVIDIA Corporation, 2026; Red Hat, 2023). Fast-math approximations are a well-known issue for reproducibility: DeepSeek-V4's TileLang defaults to strict IEEE 754 rounding for element-wise operations, treating `fast-math` as opt-in (DeepSeek-AI, 2026).



(a) Sequential

(b) Group pairwise

(c) Fused, e.g. a single block FMA

(d) Chain of fused, e.g. MMA or reduction across tensor cores

Figure 1. Four summation topologies that compute $c + \sum_i p_i$. All produce the same result in exact arithmetic, but not in general under floating-point rounding. Adapted from (Xie et al., 2025).

2.2. Reductions

Reductions sum multiple parallel elements into one. Floating-point addition is non-associative: $a + (b + c) \neq (a + b) + c$. Each addition rounds, and the rounding depends on the relative magnitudes of the operands (Goldberg, 1991).

On a GPU computing a forward pass of a FFN or attention block in a transformer, there are two distinct factors deciding reduction orders of element summation (e.g. when computing a matrix element of a GEMM[4]):

1. **Hardware-level reduction:** An NVIDIA GPU performs GEMMs in subdivided tiles, with the smallest sub-tiles allocated to tensor cores (NVIDIA Corpora-

[1] All experiments used unmodified HuggingFace Transformers and vLLM. Software versions were logged for every experiment. Version sensitivity is itself one of the configuration factors we characterize in Sections 2 and 3. Representative stacks include vLLM 0.11.2 with PyTorch 2.9.x/CUDA 12.8, and HF Transformers 4.57.3 with flash-attn 2.8.3.

[2] Parallel prefix scans, used in state-space models such as Mamba (Gu & Dao, 2023), are a distinct topology class governed by the same non-associativity mechanism but not strictly a reduction. We focus on transformer inference in this work.

[3] The IEEE 754 standard (IEEE, 2019) specifies correctly-rounded results for basic arithmetic operations but does not standardize transcendental functions, leaving implementations free to approximate (Goldberg, 1991).

[4] General Matrix Multiply, a fundamental operation dominating most FLOPs of AI inference. Defined as $C \leftarrow \alpha AB + \beta C$ (NVIDIA Corporation, 2025b).

tion, 2025e). Tensor cores perform matrix multiply-accumulate (MMA) via chains (Figure 1(d)) of fused block multiply-add (block FMA) units. The topologies of both are *fixed in silicon* (i.e. the number of p-elements in block FMA as shown in Figure 1(c) and the chaining inside an MMA) (Khattak & Mikaitis, 2025; Xie et al., 2026)[5]. Software can chain many MMA operations together, but cannot change a single MMA's reduction order[6]. CUDA cores, in contrast, are scalar Arithmetic Logic Units (ALUs) without internal reduction. Any multi-operand reduction on CUDA cores is orchestrated by software instructions (warp shuffles, shared-memory trees).

2. **Software-level reduction:** Software decides the reduction order at every level above the MMA. Here we distinguish two cases:
   (a) **Static:** Kernel dispatch follows deterministic heuristics that depend on tensor shapes, data types, and library version. The reduction order is thus fully determined by the software stack and input shapes, not by runtime scheduling (Thinking Machines Lab, 2025).
   (b) **Atomic:** Some kernels reduce partial results via `atomicAdd` on floating-point outputs, where the accumulation order depends on the GPU's warp scheduler and is not reproducible across runs[7]. This is the sole source of genuine non-determinism we identified in our experiments, and was limited to specific INT de-quantization kernels (see Section 3).

On identical hardware, apparent non-determinism arises when reduction orders do not match. This is the case when using different (software version-dependent) kernel libraries or batch sizes (which change the tensor shapes, in turn triggering different kernel selection and launch-parameter choices such as tile sizes and split-K factor). On different hardware, additional factors include different MMA topologies in different tensor core generations, as well as different approximations used in element-wise calculation on special function units.

3. Empirical Verification of Determinism

3.1. Methodology

We generally use unmodified vLLM and HuggingFace Transformers on various NVIDIA GPUs hosted by RunPod and Vast.ai. No determinism flags are set in PyTorch or CUDA. We compute L2 distance between extracted signal vectors (log-probabilities, key vectors and occasionally hidden states for HF transformers, only log-probabilities for vLLM[8]) across multiple reference prompts, constructed from natural language prompts (typically at 10,000 token sequence length, but also with occasional stress-tests at longer sequences). When comparing decode inference outputs across conditions (across hardware, batch sizes, etc.), we used teacher-forcing to ensure computation ran on identical tokens.

3.2. Results

Reproducibility check within fixed conditions We ran prefill and decode inference with identical models and inputs and found bitwise identical results when fixing the following conditions ($L2 = 0$): hardware SKU; software stack (CUDA toolkit, PyTorch, inference engine and version, attention backend); quantization format and kernel variant; and tensor parallelism rank. When varying batch size, we left the first batch element's inputs unchanged and compared how element 0 is affected by batch neighbours. Different physical cards of the same SKU (e.g. A100 80 GB on RunPod (PCIe) vs. Vast.ai (SXM)) produce identical outputs, confirming that variance originates in software and tensor core layouts, not hardware manufacturing defects[9].

When testing models of different architectures and quantization formats, (including Mistral, Qwen, Kimi, Deepseek

[5]For instance, on the A100 BF16 tensor core (SM80), each block FMA sums 8 products with the accumulator c. One MMA instruction (`mma.sync.aligned.m16n8k16`) chains two block FMAs to cover its $K = 16$ reduction depth (Khattak & Mikaitis, 2025; Xie et al., 2026).

[6]There are variants of MMA using different chaining of block FMAs, but any individual MMA instruction has a fixed reduction tree.

[7]The warp scheduler is non-deterministic because it makes dispatch decisions based on runtime state that varies between runs: which warps are stalled on memory (NVIDIA Corporation, 2025d), cache state and clock frequency (which NVIDIA itself controls for profiler-reproducibility by purging caches and adjusting clocks (NVIDIA Corporation, 2025d)), thermal throttling under sustained load (Jia et al., 2019), and memory-controller request reordering across warps (Jog et al., 2014). Two identical kernel launches on the same GPU will have different warps ready at different cycles, so when multiple warps race to `atomicAdd` to the same output address, the order they arrive is effectively random (Shanmugavelu et al., 2024; Thinking Machines Lab, 2025).

[8]The public vLLM API only exposes log-probabilities. Intermediate tensors are technically accessible via PyTorch forward hooks on the internal model, but this is fragile across versions, incompatible with CUDA graph capture, and does not straightforwardly reconstruct the full KV cache under tensor parallelism.

[9]The full GH100 and GA100 dies contain 144 and 128 SMs respectively, while the H100 SXM5 and A100 SXM4 products ship with 132 and 108 active SMs (NVIDIA Corporation, 2022; 2020). Disabling defective cores to match a lower advertised count is standard yield-management practice across CPU and GPU vendors, including explicitly for the H100 (Cerebras Systems, 2025). The active SM count is fixed per SKU, and tensor-core reduction trees are fixed in silicon (Khattak & Mikaitis, 2025; Xie et al., 2026) Bit-exactness across physical cards of the same SKU is therefore unsurprising

and GLM models at BF16, FP8, INT8 and INT4), we encountered genuine non-determinism for some, but not all INT-quantized models. Experimentally, we isolated the root cause to specific INT-dequantization kernels: The same model, Qwen 3 8B, produced deterministic outputs for AWQ quantized weights, and GPTQ quantization when activating vLLM's `marlin` kernel. When deactivating said kernel for GPTQ, log-probabilities of repeated prefill and decode runs diverged. Deactivating `marlin` makes vLLM default to the `exllama` kernel inside $vllm/csrc/quantization/gptq/q_gemm.cu$. For example:

```
//q_gemm.cu:319-320
atomicAdd(out,result01);
atomicAdd(out + 1, result23);
```

With other occasions of atomics in that same kernel. We conclude that our experimental measurements confirm that inference on NVIDIA GPUs using vLLM and HF Transformers kernels is deterministic unless atomic functions are called in the backend (which has become rare for modern inference engines).

Comparison across conditions We found outputs to be invariant for some conditions. Pipeline parallelism rank (tested with Mistral Small 3 spread over 1, 2 and 4 A100 GPUs) is an unsurprising example: GPUs process a transformer forward pass layer by layer, and transmission of activations from one GPU to the next is deterministic data transfer. Minor firmware differences also left outputs invariant, while major kernel library updates created 1-5% divergence in hidden states (L2 distance/magnitude) [10].

We observed sequence-length-dependent occasions of "equivalence classes" in outputs in vLLM and Transformers: While adding batch neighbors generally altered batch element 0's outputs, with increasing sequence length the differences in outputs across batch sizes began to disappear. This effect was most pronounced in prefill inference, where differences disappeared at larger batch sizes first, with complete batch size invariance beginning at sequence lengths above 4000 tokens. For decode inference, we observed a similar effect: occasionally, adjacent batch sizes left batch element 0 unaffected, (e.g. batch size 4 vs. batch size 5), while increasing sequence length made such invariant groups rarer. In no experiments did the *token identities* of batch neighbours affect the first batch element in any way.

These results are the product of the interaction of kernels with input tensor shape: In prefill inference, the forward pass behaves as a large matrix multiplication (GEMM) where

[10]The version bump cu128 to cu129 did not introduce A100-targeted kernel updates to cuBLAS, and inference results were unaffected. We did however measure differences across cu118 and cu12.0

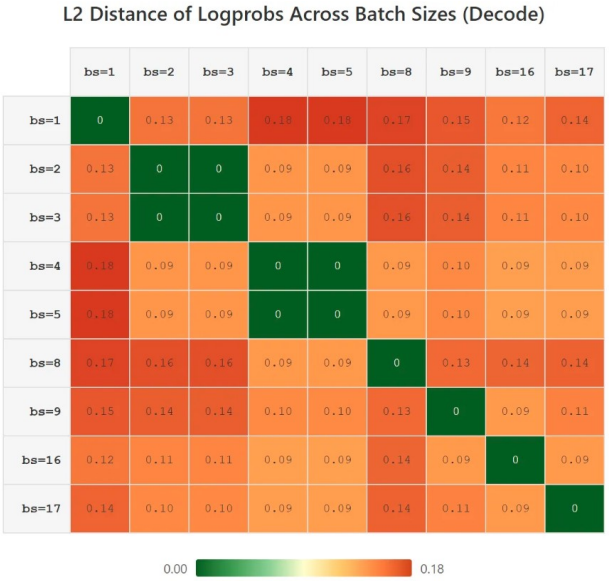


Figure 2. Comparison of top-5 logprobs (average across measurements at the last three token positions of the first batch element) across different batch sizes for decode inference in vLLM. We see that some changes in batch size can leave batch element 0's outputs unchanged. Such "equivalence classes" were consistent across prompts for any given fixed sequence length, but they became rarer with increasing sequence length.

the row dimension is M = Batch × Sequence Length. For small sequences, the kernel dispatcher employs different tiling strategies and Split-K variants to utilize the GPU, altering the reduction tree as batch size changes. As sequence length increases, we empirically observe that kernel choices stabilize, leading to invariant outputs across batch sizes for sufficiently long sequences (e.g. above ∼4000 tokens in our experiments). Decode inference behaves as a memory-bound matrix-vector operation (GEMV) where M = Batch Size. In this regime, implementations favor specific dimension alignments for efficient execution (e.g. multiples compatible with Tensor Core usage), which can lead to identical reduction trees for groups of batch sizes. However, as the KV-cache (and thus the attention reduction dimension) grows with sequence length, kernel selection and tiling may again change, gradually reducing these equivalence classes. Deepseek-AI independently reports on this issue in the technical report of their V4 model family: *"Traditional cuBLAS library cannot achieve batch invariance"*. (DeepSeek-AI, 2026)

Lastly, we examined the relative contribution of different sources of discrepancy and observed that rounding differences from distinct factors (e.g., hardware SKU, kernel implementation, and algorithmic choices) combine in an approximately additive manner in practice. This was particularly apparent during development of the emulator (Section 4), where incrementally correcting mismatches between the simulation and ground truth consistently reduced the re-

maining error. Empirically, we observe a rough hierarchy in the magnitude of these effects: quantization format and numerical precision have the largest impact, followed by the attention implementation (e.g. SDPA, eager, FlashAttention), hardware SKU, and then remaining factors such as inference mode (prefill vs. decode), batch size, and tensor-parallel configuration.

4. Bit-Exact Emulation of GPU-Accelerated LLM Inference

We built a software-only C and Python emulator capable of predicting every bit of intermediate tensors in a transformer forward pass across multiple NVIDIA GPU architectures. In most tests, we emulate individual FFN and ATTN blocks of Qwen 3 4B in isolation, but we also validated a full forward pass through the model: token embedding, all 36 transformer blocks in series (attention + FFN), final RMSNorm, and the language-model head. Our results are presented in Appendix E.

4.1. Tensor Core Arithmetic Model

We emulate the tensor core's block FMA operation and the PTX-instructed MMA (see footnote 5). Following the hardware characterizations by Khattak & Mikaitis (2025) and Xie et al. (2026), block FMA computes raw products without FP32 normalization. Instead, products are aligned to a maximum exponent within a fixed-point window (e.g., 26 bits for the A100: 2 integer, 23 fraction, 1 extra alignment bit), bits shifted out of the window are truncated, the components are summed as integers, and the final result is truncated to FP32. Our emulator replicates this exact arithmetic, with tensor core profiles for each SKU (we created profiles for Lovelace, Ampere, Hopper and Blackwell SKUs based on prior work by Khattak & Mikaitis (2025) and Xie et al. (2026)).

4.2. Special Function Units

NVIDIA GPUs accelerate transcendental functions via Multi-Function Units (MUFU). Instructions like `MUFU.RSQ` (reciprocal square root), `MUFU.EX2` (exponential with base 2), and `MUFU.RCP` (reciprocal) rely on architecture-specific silicon lookup tables. We found their outputs are deterministic but deviate from IEEE-correct(IEEE, 2019) rounding by up to ± 2 ULP. By exhaustively probing these instructions via inline PTX, we cached their exact hardware outputs (e.g. a 4GB table mapping every 32-bit input for `MUFU.EX2`), ensuring bit-exact SFU emulation without needing GPU access at verification time.

4.3. Software Reduction, Kernel and RoPE Emulation

Matching GPU-accelerated outputs also requires meticulously modeling kernel-specific reduction trees and compiler optimizations:

- **GEMM:** Most FLOPs in transformer inference are in matrix multiplication, and above the tile sizes handled fully within tensor cores, software decides tile boundaries and reduction ordering. To achieve bit-exact emulation, our code replicates the exact accumulation path dictated by the chosen kernel configuration:
  - **K-Iteration Order:** The intermediate value of the FP32 accumulator dictates the maximum exponent used for the hardware's fixed-point alignment window. Therefore, the exact sequence in which blocks are added is mathematically load-bearing. Our emulator mirrors the sequential K-walk used by the target kernel's mainloop (CUTLASS's default configuration).
  - **BF16 epilogue:** After each projection's FP32 accumulation, the GPU stores the result to global memory as BF16. Our emulator mirrors this by casting the raw FP32 accumulator to BF16 before it enters the next stage. We verify both the raw FP32 accumulator and the BF16-cast output against CUTLASS to confirm that neither the accumulator nor the epilogue hides a discrepancy.
  - **Live Accumulator State (Fused Kernels):** Standard projections typically initialize the accumulator (depicted as "c" in figure 1) to zero. However, in fused operations like FlashAttention-2, the PV matrix multiplication accumulates directly into a live, running register (O_{acc}) across KV tiles. Our emulator addresses this using a specialized `block_fma_batch` primitive that applies the hardware block FMA to an existing accumulator state, matching the alignment window shifts that occur on hardware.
- **Memory Boundaries:** When the GPU writes BF16 tensors to global memory between pipeline stages (e.g., after RoPE and before FA2), we explicitly enforce a BF16 quantization boundary in the emulator to ensure the tensor core receives the exact same significands as the physical hardware.
- **RMSNorm:** We emulate PyTorch's parallel `reduce_kernel` (including its warp-shuffle topology, which changes across PyTorch versions), the `nvcc` compiler's optimization of division into multiply-by-reciprocal, the `MUFU.RSQ` rounding, and the specific cast ordering (FP32 normalization $\rightarrow$ BF16 cast $\rightarrow$ BF16 weight multiply).
- **FlashAttention-2:** FA2 (2.8.3) emulation required resolving several nuanced interactions between the hardware

and the compiler's optimizations:

- **Accumulator Initialization:** As explained above under "GEMM".
- **Online Softmax and SFU Usage:** We emulate FA2's block-wise execution pattern: FA2 maintains running statistics for the maximum score $m^{(i)}$ and the sum of exponentials $l^{(i)}$ for each block i. When transitioning to a new block, previous running values are scaled by $2^{(m^{(i-1)}-m^{(i)})}$. Our emulator computes these scale factors and the attention weights $P = 2^{S-m^{(i)}}$, and final normalization using our probed MUFU hardware models.
- **Compiler FMA Fusion:** The `nvcc` compiler fuses operations across inline function boundaries. Specifically, the $l^{(i)}$ rescale multiplication is fused with the addition of the first element of P into a single hardware FMA (FFMA) instruction, resulting in one rounding instead of two. Our emulator mimics this single-rounding step using float64 intermediates.

- **RoPE:** We enforce strictly matching the GPU's CUDA `libm cosf/sinf` (which differ slightly from CPU `glibc`) and snap values to BF16 precisely at every stage boundary including QK-norm, RoPE, and FA2.

To extend emulation from an initial CUTLASS-target to the proprietary cuBLAS library, we build a one-time per-SKU *dispatch catalog* via cuBLASLt's introspection API, mapping each matmul shape (M,N,K,dtype,layout) to the exact kernel ID cuBLAS dispatches. The search space for any particular model can be substantially narrowed by iterating the weight tensor shapes over the sequence length dimension, which takes minutes and is a one-time setup. For each cataloged kernel, the emulator maps to one of the four reduction topologies that we could narrow down. [11]

4.4. Diagnostics and Results

We tested our emulator on LLM activations (Qwen3-4B) against GPU ground truth computed using CUTLASS kernel libraries and —via the dispatch catalog described in Section 4.3 — against cuBLAS, which is used by standard Pytorch. Our emulation models individual FFN and ATTN blocks of the transformer stack and can be stacked to simulate a full forward pass.

For the full FFN block (RMSNorm, three matmul projections, SiLU, and residual add), we achieved exactly 0 BF16 diffs (and 0 FP32 accumulator diffs) across all intermediate elements on A100, L40, L40S and H100. For the (significantly more complex) attention block featuring FlashAttention-2, we achieved 0 BF16 diffs (and 0 FP32 accumulator diffs) out of 71M elements (projections + QK-norm + FA2 scores), at a sequence length of 4,000 tokens. By "elements", we mean intermediate tensors captured at multiple in-between states:

For the FFN block we captured and compared six specific intermediate operations: the raw outputs of the `gate` and `up` projections, the element-wise SiLU activation applied to the gate output, the element-wise multiplication of the activated gate and the up projection (`SiLU(gate) * up`), the output of the `down` projection, and the final FFN block output following the residual addition. For the ATTN block, we captured and compared the pre-attention RMSNorm output, the individual Query, Key, and Value (`Q`, `K`, `V`) matrix projections, the per-head QK-normalization outputs, the core FlashAttention-2 outputs (which encompasses the Rotary Positional Embeddings and the fused online softmax with QK/PV matmuls), the final Output (`O`) projection, and the overall attention block output after the residual connection.

Our three-way diagnostic confirmed that the emulator perfectly matches the CUTLASS intermediates with zero differences at all tested sequence lengths (from 64 up to 8000, see Appendix E). We found that CUTLASS-generated outputs diverge from a forward pass using cuBLAS kernels, particularly at short sequence lengths. In the attention block at 4,000 tokens, cuBLAS and CUTLASS converge on identical outputs across all pre- and post-attention projections (`q`, `k`, `v`, `o`), so the CUTLASS-target emulator already matches cuBLAS here without the dispatch catalog being used.

5. Limitations

MoE inference. Our experiments discussed in Section 3.2 include Mixture-of-Experts models such as Qwen3-30B-A3B, GLM-4.6 and Deepseek-v2-coder-lite under tensor parallelism at ∼120k-token contexts, all of which produce bit-identical log-probabilities across repeated vLLM runs[12]. However, our emulator targets dense FFN and attention blocks, whereas most frontier models are MoE. Beyond the experimental results, we expect emulation to be generalizable to MoE for a more direct reason: Direct inspection of the HuggingFace `Qwen3MoeSparseMoeBlock` confirms that an MoE extension would require no numerical primitives not already included in our model. For details, see the Appendix B. We also note that DeepSeek-V4 demonstrates end-to-end batch-invariant and deterministic MoE inference at frontier scale (DeepSeek-AI, 2026).

Only NVIDIA GPUs. While our experiments were limited to NVIDIA's technology stack, the methodology and

[11] We found no obvious pattern in the way cuBLAS dispatches kernels based on tensor shapes, so we continue to rely on cataloging. For the three tensor shapes of the FFN weight matrices in the Qwen 3 4B model, a catalog of hundreds of thousands of possible tensor shapes is only megabytes of INT indices.

[12] Kimi-K2-Thinking was the sole non-deterministic model in our tests, and we traced this to its INT4 de-quantization, see Section 3.2

general principles transfer directly to other hardware. (Khattak & Mikaitis, 2025) and (Xie et al., 2026) have profiled AMD's matrix cores (CDNA2 and CDNA3, i.e., MI200- and MI300-series Instinct GPUs) in a similar manner as they did for NVIDIA's tensor cores, so we expect that further emulation work can demonstrate the same end-to-end inference verification that we applied to NVIDIA's GPUs, CUDA and CUTLASS.

The `nvjet` kernel family on Hopper. We found that on Hopper GPUs and sequence lengths below 250 tokens in particular, cuBLAS can occasionally dispatch to deterministic, but proprietary kernels from the `nvjet` family. This is an edge case that vanishes for longer sequences, and one that could in principle be profiled via black-box methods such as those already used to find MMA reduction order in tensor cores (Khattak & Mikaitis, 2025; Xie et al., 2026).

Backward pass and gradient accumulation in LLM training. Our experimental and emulation scope was inference only. Training makes use of backward pass kernels and all-reduce, which can introduce atomic functions at multiple levels. FlashAttention-3's Hopper backward accumulates dQ, dK and dV via `atomicAdd` on the non-TMA code paths (Shah et al., 2024)[13] and fused normalization backward kernels use atomics for weight gradients. There are deterministic alternatives to atomics, and they are already preferred in frontier LLM training for better reproducibility and stability (DeepSeek-AI, 2026).

6. Implications for AI Governance

We showed that for any (NVIDIA) hardware and software setup used in inference **that makes no use of floating-point atomic functions**, outputs can be bitwise reproducible without performance-degrading settings if the following information is recorded for later reference[14]:

1. **Hardware SKU** (determines tensor core arithmetic and special function unit behavior)
2. **Exact model weights** (in the deployed quantization format)
3. **Parallelism topology** (separately for prefill and decode stages)
4. **Software versions and any custom kernels** (CUDA toolkit, PyTorch, attention backend, quantization library)
5. **Batch size at each forward pass** (one integer per decode step under continuous batching[15], or record whole batches rather than single sequences)

The verifier can then reconstruct the prover's outputs, or by using identical hardware, by using custom accelerators with identical hardware MMA reduction trees and SFUs, or by emulating all rounding decisions in software.

A caveat applies to the last item. Recording concurrent batch size is only necessary because cuBLAS and standard inference engines are not batch-invariant. If the prover uses batch-invariant kernels such as those deployed in DeepSeek-V4 (DeepSeek-AI, 2026), batch size need not be recorded. However, this also makes batch size undetectable *in outputs alone*. More generally, a prover may choose (or be obligated to use) kernels that are not invariant to a governance-relevant property, precisely so that property becomes numerically verifiable.

Cost of re-computation. Sparse random sampling makes even software-only emulation with no hardware-acceleration viable. The verifier's statistical confidence $P_{detect}(\geq 1)$ for an upper-bound percentage p_{false} of misreported records depends only on the total number of samples k, not the total number of records to sample from (if this number is $>> k$).

$$P_{detect}(\geq 1) = 1 - (1 - p_{false})^k \tag{1}$$

This works against a covert adversary that attempts to misreport a *percentage* of computation in large-scale AI inference. Verifying enough samples to provide strong statistical assurance requires only tens to hundreds of CPUs running continuously (see Appendix C), regardless of the prover's total throughput.

For precise verification, the prover need not store all activation tensors, which would be infeasible. A cryptographic hash (or a projection as used by (Karvonen et al., 2025)) of a single tensor deep in the computational chain — such as the hidden state at the last layer and last token position of a long sequence — serves as a highly collision-resistant fingerprint for the entire upstream computation. Bit-exact emulation lets the verifier recompute the expected hash and compare.

Bit-exact reproducibility transforms non-associative rounding from an obstacle to verification into a lever for technical

[13] `atomicAdd` on dQ at lines 964, 983, 992 of `hopper/mainloop_bwd_sm90_tma_gmma_ws.hpp`; on dV at line 480 and dK at line 511 of `hopper/epilogue_bwd.hpp`. A `Deterministic` template parameter in both files gates per-block semaphore serialization (`Barrier::arrive_inc`/`wait_eq`) around these accumulations.

[14] We only list information related to reduction ordering and hardware-specific element-wise operations, not software-sampling such as temperature

[15] If prefill and decode are mixed within a single forward pass (e.g. under continuous batching without prefill-decode disaggregation), the per-entry prefill/decode status and sequence lengths must also be recorded, as these determine the kernel-facing tensor shape.

governance capable of identifying the hardware and software stack used for inference. Any *numerically relevant* modifications of kernel libraries, batch composition, and model parallelism can be detected directly in the outputs.

LLM Usage Statement

Large language models were used as a research and writing aid throughout this work. In the research process, they assisted with reading and interpreting vendor documentation and library source code, debugging the emulator, and discussing results. In writing, they provided editorial review, drafting for the author's consideration, and formatting. In this process, several LLM-generated claims were found to be incorrect and discarded. The emulator and experimental code were written largely by coding agents under the author's direction. All design decisions, the methodology, and the verification of correctness are the author's. All experimental measurements and the final prose are the author's own, and no reported numerical result was produced or modified by an LLM.

References

Advanced Micro Devices. AMD EPYC 9965 processor. https://www.amd.com/en/products/processors/server/epyc/9005-series/amd-epyc-9965.html, 2024. 192 Zen 5 cores, full 512-bit AVX-512 datapath.

Aumann, Y. and Lindell, Y. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

Baker, M., Kulp, G., Marks, O., Brundage, M., and Heim, L. Verifying international agreements on AI: Six layers of verification for rules on large-scale AI development and deployment. *RAND Corporation*, 2025.

Cankaya, N., Kryś, J., Ng, J., Marks, L., and Krückel, F. Fingerprinting all AI cluster I/O without mutually trusted processors. https://apartresearch.com, 2026. Research submission to the Technical AI Governance Challenge hosted by Apart Research.

Cerebras Systems. 100x Defect Tolerance: How Cerebras Solved the Yield Problem. Cerebras Engineering Blog, 2025. URL https://www.cerebras.ai/blog/100x-defect-tolerance-how-cerebras-solved-the-yield-problem.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report, DeepSeek-AI, April 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Harack, B., Trager, R. F., Reuel, A., Manheim, D., Brundage, M., Aarne, O., Scher, A., Pan, Y., Xiao, J., Loke, K., Adan, S. N., Bas, G., Caputo, N. A., Morse, J. C., Ahuja, J., Duan, I., Egan, J., Bucknall, B., Rosen, B., Araujo, R., Boulanin, V., Lall, R., Barez, F., Alvira, S., Katzke, C., Atamli, A., and Awad, A. Verification for international AI governance. Technical report, Oxford Martin AI Governance Initiative, 2025.

IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul 2019. doi: 10.1109/IEEESTD.8766229.

Jia, Z., Maggioni, M., Smith, J., and Scarpazza, D. P. Dissecting the NVidia Turing T4 GPU via microbenchmarking. Technical report, Citadel, 2019. arXiv:1903.07486.

Jog, A., Kayiran, O., Pai, A., Kandemir, M. T., Mishra, A. K., Iyer, R., and Das, C. R. Managing DRAM latency divergence in irregular GPGPU applications. In *SC '14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 128–139. IEEE, 2014.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Karvonen, A., Reuter, D., Rinberg, R., Marks, L., Garriga-Alonso, A., and Warr, K. DiFR: Inference verification despite nondeterminism. *arXiv preprint arXiv:2511.20621*, 2025.

Khattak, F. A. and Mikaitis, M. Accurate models of NVIDIA tensor cores. *arXiv preprint arXiv:2512.07004*, 2025.

NVIDIA Corporation. NVIDIA A100 Tensor Core GPU Architecture. Technical report, NVIDIA, 2020. Whitepaper v1.0.

NVIDIA Corporation. NVIDIA H100 Tensor Core GPU Architecture. Technical report, NVIDIA, 2022. Whitepaper v1.04.

NVIDIA Corporation. CUDA toolkit 12.9 release notes. Technical report, NVIDIA, 2025a. URL https://docs.nvidia.com/cuda/archive/12.9.0/cuda-toolkit-release-notes/index.html.

NVIDIA Corporation. NVIDIA deep learning performance guide: Matrix multiplication. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, 2025b.

NVIDIA Corporation. NVIDIA libdevice user's guide. https://docs.nvidia.com/cuda/libdevice-users-guide/, 2025c.

NVIDIA Corporation. NVIDIA Nsight Compute Profiling Guide. https://docs.nvidia.com/nsight-compute/ProfilingGuide/, 2025d.

NVIDIA Corporation. NVIDIA PTX ISA reference. https://docs.nvidia.com/cuda/parallel-thread-execution/, 2025e. Documents `ex2.approx.f32`, `rcp.approx.f32`, `rsqrt.approx.f32` SFU instructions.

NVIDIA Corporation. CUDA toolkit release notes. https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/, 2026. Documents libdevice math function accuracy changes across toolkit versions.

OpenRouter. State of AI 2025: 100T token LLM usage study. https://openrouter.ai/state-of-ai, 2025.

Red Hat. glibc floating point math functions provide slightly different results between RHEL major releases. https://access.redhat.com/solutions/7032163, 2023. Documents precision differences in exp2f() across glibc 2.17, 2.28, and subsequent versions.

Rinberg, R., Karvonen, A., Hoover, A., Reuter, D., and Warr, K. Verifying LLM inference to detect model weight exfiltration. *arXiv preprint arXiv:2511.02620*, 2025.

Scher, A. and Thiergart, L. Mechanisms to verify international agreements about AI development. Technical report, Machine Intelligence Research Institute, 2024. Also available as arXiv:2506.15867.

Shah, J., Bikshandi, G., Zhang, Y., Thakkar, V., Ramani, P., and Dao, T. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Shanmugavelu, S., Taillefumier, M., Culver, C., Hernandez, O., Coletti, M., and Sedova, A. Impacts of floating-point non-associativity on reproducibility for HPC and deep learning applications. In *SC '24 Workshops*, 2024.

Sun, H., Li, J., and Zhang, H. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of ACM CCS*, 2024.

Thinking Machines Lab. Defeating nondeterminism in LLM inference. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/, 2025.

Wasil, A. R., Reed, T., Miller, J. W., and Barnett, P. Verification methods for international AI agreements. *arXiv preprint arXiv:2408.16074*, 2024.

Xie, P., Wang, Y., Yang, F., and Yang, M. MMA-Sim: Bit-accurate reference model of tensor cores and matrix cores. *arXiv preprint arXiv:2511.10909v1*, 2025.

Xie, P., Xu, S., Wang, Y., Yang, F., and Yang, M. Bit-accurate modeling of GPU matrix multiply-accumulate units: Demystifying numerical discrepancy and accuracy. *arXiv preprint arXiv:2511.10909v2*, 2026.

Yuan, J., Li, H., Ding, X., Xie, W., Li, Y.-J., Zhao, W., Wan, K., Shi, J., Hu, X., and Liu, Z. Give me FP32 or give me death? Challenges and solutions for reproducible reasoning. *arXiv preprint arXiv:2506.09501*, 2025.

A. CUDA Version Comparison

RunPod and Vast.ai A100 instances running CUDA 12.8 and 12.9 (cuBLAS 12.8.3 vs 12.9.0) produced bit-identical hidden states and key vectors across all 29 sampled layers. The cuBLAS changelog for this update lists no additions targeting A100 kernel paths (NVIDIA Corporation, 2025a), leaving kernel paths unchanged, and the bit-exact results confirm this empirically. However, when comparing between CUDA 11.8 and 12.1, we found 1-5% relative error (L2 distance/magnitude) in hidden states, propagating and accumulating through the model's depth (Qwen 2.5 7B). The cuBLAS 12.0 release notes document kernel-catalog changes (removal of matmul stage constants, Hopper-specific kernel additions that alter dispatch heuristics, and a bias-gradient correctness fix on Ampere), which we identify as the root cause for the change in kernel selection for A100 GEMMs.

B. MoE Router Analysis

We inspected the Qwen3 MoE router code to verify that it introduces no new numerical primitives beyond those already modeled.

- The router is `F.linear` → FP32 `softmax` → `torch.topk` → ℓ_1-normalization of the top-k weights. Each component is a strict subset of machinery the emulator already provides: the linear is a GEMM (Section 4.3); the softmax uses `MUFU.EX2` and `MUFU.RCP`, both exhaustively probed (Section 4.2); `topk` is an integer sort with no floating-point rounding; and the top-k normalization $x \cdot \text{reciprocal}(\sum x)$ is structurally identical to the final divide in RMSNorm.
- Each expert is a gated FFN (`gate_up_proj` → SiLU-gate → `down_proj`), i.e. the exact computation the emulator already reproduces bit-exactly (Section 4.4).
- Per-expert outputs are accumulated into `final_hidden_states` via a Python `for` loop over `expert_hit`, returned in sorted order by `.nonzero()`. Within each `index_add_` call, `token_idx` is duplicate-free (`torch.topk` returns distinct experts per token), so no atomic contention occurs; across iterations, the reduction order is fixed by the deterministic `nonzero` ordering. vLLM replaces the loop with a fused grouped-GEMM kernel, but the per-expert GEMM and weighted sum are again strict subsets of existing emulator primitives.

C. Inference FLOP Calculation

Forward-pass FLOP estimate. We use the standard approximation $\text{FLOP}_{\text{fwd}} \approx 2 \cdot N_{\text{active}} \cdot T$ for a transformer forward pass (Kaplan et al., 2020), where N_{active} is the number of active parameters per token and T is the total number of tokens processed. The factor of two comes from the multiply-accumulate operation dominating matrix multiplication. This formula counts activation-weight matmuls in QKV/O projections and FFN blocks; it does not include the activation-activation matmuls inside attention ($QK^\top$ and PV), which scale as $\mathcal{O}(L \cdot d_{\text{model}} \cdot N_{\text{context}})$ per token. For short-context workloads the under-count is negligible; the crossover at which attention FLOPs equal FFN FLOPs occurs near $N_{\text{context}} \approx 3\, d_{\text{model}}$, and this is without factoring current trends towards sparse or linear attention.

Verification workload. Random-sample verification of B batches, each containing S sequences of average length T_{seq}, requires

$$\text{FLOP}_{\text{verify}} = 2 \cdot N_{\text{active}} \cdot B \cdot S \cdot T_{\text{seq}}.$$

Published metadata from OpenRouter's 100T-token usage study (OpenRouter, 2025) reports an average prompt length of approximately 5,400 tokens and an average generated response of 600 tokens per request (late 2025). With $B = 50$, $S = 64$, $T_{\text{seq}} = 6{,}000$, and a representative $N_{\text{active}} = 100$B mixture-of-experts model, this yields $k = B \cdot S = 3{,}200$ independently verifiable sequences and approximately 3.8 EFLOP of recomputation. Scaling to $B = 500$ gives $k = 32{,}000$ and 38 EFLOP. For workloads with substantially longer average sequences, the budget scales linearly in T_{seq}: at $T_{\text{seq}} = 30{,}000$ it becomes 19 EFLOP ($B = 50$) or 192 EFLOP ($B = 500$); at $T_{\text{seq}} = 100{,}000$, 64 EFLOP or 640 EFLOP respectively.

Scale of audited workload. For context, OpenRouter reports aggregate daily token traffic of approximately 2×10^{12} tokens (OpenRouter, 2025). At $N_{\text{active}} = 100$B (overestimate for open-weights models in early 2026) this corresponds to roughly 400 ZFLOP per day. The $B = 50$ verification workload is therefore approximately 10^{-5} of daily OpenRouter-scale compute at the measured average sequence length, rising to approximately 1.6×10^{-3} at the $B = 500$, $T_{\text{seq}} = 100{,}000$ upper estimate.

Detection-probability scaling. Applying Equation (1) to a covert adversary misreporting 0.1% of records ($p_{\text{false}} = 10^{-3}$):

$$P_{\text{detect}}(k = 3{,}200) \approx 0.96, \qquad P_{\text{detect}}(k = 32{,}000) \approx 1 - 10^{-14}.$$

The $10\times$ scale-up from $B = 50$ ($k = 3{,}200$) to $B = 500$ ($k = 32{,}000$) does not materially improve detection at 0.1% misreporting (already saturated), but extends reliable detection into the 0.01% range (at 96% confidence, or 0.005% at 80% confidence).

CPU hardware requirement. Converting the verification-FLOP figures to sustained throughput over a $86{,}400$-second day yields requirements from approximately 44 TFLOP/s (at $B = 50$, measured average $T_{\text{seq}} = 6{,}000$) to 7.4 PFLOP/s (at $B = 500$, $T_{\text{seq}} = 100{,}000$). A current-generation AMD EPYC 9965 CPU provides approximately 27 TFLOP/s FP32 peak (Advanced Micro Devices, 2024); at 60% sustained utilization, the raw-FLOP requirement maps to roughly 3 CPUs for the grounded estimate and ≈ 275 CPUs for the pessimistic estimate.

Both numbers likely need to be multiplied by at least $10\times$, factoring in the overhead of emulating reduction trees. Still, even with a conservative estimate of 100B active parameters, we conclude that the resource requirements of even software-only bitwise inference verification are manageable, thanks to the favourable scaling of random sampling.

D. Extended Reproducibility Results

Table 1. Cross-hardware L_2 distance between Qwen2.5-7B-Instruct prefill hidden states (FP16, 1134-token prompt, batch size 1, 10 runs per GPU). Diagonal entries are within-hardware across 10 repeats and are all exactly 0.000000 ± 0.000000. Off-diagonal zeros identify SKUs sharing a tensor-core generation; non-zeros identify architecture boundaries. Source: `different_hardware.py`.

	A100-SXM4	A100-PCIe	A40	H100-NVL	H100-PCIe	H200	L40S
A100-SXM4	–						
A100-PCIe	**0.000**	–					
A40	0.496	0.496	–				
H100-NVL	0.526	0.526	0.549	–			
H100-PCIe	0.526	0.526	0.549	**0.000**	–		
H200	0.526	0.526	0.549	**0.000**	**0.000**	–	
L40S	0.450	0.450	0.576	0.456	0.456	0.456	–

The within-SKU zeros (A100-SXM4 $\leftrightarrow$ A100-PCIe) rule out physical-card variance: yield-binned chips of the same SKU produce bit-identical outputs. The within-architecture zeros (H100-NVL, H100-PCIe, H200) show that tensor-core arithmetic is shared across products built on the same compute capability, regardless of memory subsystem or product tier. All non-zero off-diagonal distances correspond to crossing an architecture boundary (Ampere $\rightarrow$ Hopper, Ampere $\rightarrow$ Ada, etc.). Batch-size-2 and batch-size-4 panels (omitted for space) show the same pattern with slightly different magnitudes. Within-hardware statistical noise was 0.0 for all seven SKUs at all three batch sizes.

We also investigated the relative magnitude of numerical deviation in a suite of ablations. A particular focus was on the detectability of software-settings when replaying on different hardware, both in prefill and decode. We measure this as a signal-to-noise ratio:

$$\text{SNR} = \frac{\|y_{\text{claimed } A} - y_{\text{replayed } B}\|_2}{\Delta_{hardware}},$$

The denominator is the same-configuration cross-hardware floor averaged over multiple samples, the numerator averages over configuration mismatches. SNR $\gg 1$ means the mismatch is detectable above hardware noise,SNR ≈ 1 means it is not.

Table 2. Cross-hardware detectability of configuration differences. All SNR values are aggregated across reference prompts; logprob SNR is reported for comparability, since vLLM only exposes logprobs. "A100↔H100" is the mean over both generation directions.

Property varied	Test setup	Prefill SNR	Decode SNR	Detectable
Attention impl. (eager/SDPA/FA2)	HF, Qwen2.5-7B, 10k tok, A100↔H100	10.5×	5.0×	✓
Quant. format (AWQ vs GPTQ, INT4)	vLLM, Qwen3-8B, A100→H100	26.7×	21.0×	✓
Marlin kernel toggle (same format)	vLLM, Qwen3-8B, A100→H100	0.99×	0.95×	×
Batch size (9 sizes, 1–17)	HF, Qwen2.5-7B, 10k tok, A100→H100	1.08×	1.03×	×
Tensor parallelism (1/2/4)	vLLM, Qwen2.5-7B, 10k tok, A100→H100	0.90×	—	×
Decode batch size via prefill re-exec	HF, Qwen2.5-7B, A100 only	—	0.98×	×

E. Emulator Three-Way Diagnostic

Table 3. BF16 diff counts for the emulator's three-way diagnostic on Qwen3-4B, layer 20. Each cell is (Emu vs CUTLASS) / (Emu vs Model) / (CUT vs Model). "Model" is HuggingFace Transformers forward with `torch.matmul`, which dispatches to cuBLAS. Denominators are element counts per stage at the given sequence length. **0** = bit-exact. Percentage figures refer to the fraction of non-identical elements, regardless of the magnitude of deviation (typically by 1-2 ULP). The emulator matches CUTLASS bit-exactly at every tested hardware/sequence-length combination. Divergence from cuBLAS depends on kernel dispatch: on L40 the `down_proj` kernel choice differs at both tested sequence lengths, while on A100 and H100 at 8k tokens cuBLAS and CUTLASS converge.

	A100, 8k	H100, 256	H100, 8k	L40, 256	L40, 8k	A100 attn, 4k
RMSNorm out	0/20.48M	0/0.66M	0/20.48M	0/0.66M	0/20.48M	0/10.24M
gate_proj	0/77.82M / 0 / 0	0/2.49M / 0 / 0	0/77.82M / 0 / 0	0/2.49M / **37.3%**	0/77.82M / 0 / 0	–
up_proj	0/77.82M / 0 / 0	0/2.49M / 0 / 0	0/77.82M / 0 / 0	0/2.49M / **37.0%**	0/77.82M / 0 / 0	–
SiLU(gate)	0/77.82M / 0 / 0	0/2.49M / 0 / 0	0/77.82M / 0 / 0	0/2.49M / **26.6%**	0/77.82M / 0 / 0	–
SiLU · up	0/77.82M / 0 / 0	0/2.49M / 0 / 0	0/77.82M / 0 / 0	0/2.49M / **48.2%**	0/77.82M / 0 / 0	–
down_proj	0/20.48M / 0 / 0	0/0.66M / **0.47%**	0/20.48M / 0 / 0	0/0.66M / **71.0%**	0/20.48M / **37.6%**	–
FFN block out	0/20.48M / 0 / 0	0/0.66M / **0.09%**	0/20.48M / 0 / 0	0/0.66M / **42.1%**	0/20.48M / **16.7%**	–
Q projection	–	–	–	–	–	0/16.38M / 0 / 0
K projection	–	–	–	–	–	0/4.10M / 0 / 0
V projection	–	–	–	–	–	0/4.10M / 0 / 0
O projection	–	–	–	–	–	0/10.24M / 0 / 0
Q-norm	–	–	–	–	–	0/16.38M / 0 / 0
K-norm	–	–	–	–	–	0/4.10M / 0 / 0
FA2 core	–	–	–	–	–	0/16.38M / 0 / 0

Table 4. Bit-exactness validation across GPUs and sequence lengths. All entries are BF16 diff counts (0 = bit-exact). Percentage figures refer to the fraction of non-identical elements, regardless of the magnitude of deviation (typically by 1-2 ULP). The only remaining gap is the `nvjet` kernel family on H100 at seq=100 down_proj. CC = Confidential Computing mode.

GPU	Mode	seq=100	250	1000	4000	Full model
A100 (sm_80)	CUTLASS	0	0	0	0	–
A100	cuBLAS	0	0	0	0	0 diffs (seq=32,250)
L40S (Ada sm_89)	CUTLASS	0	0	0	0	–
L40S	cuBLAS	0	0	0	0	–
H100 (sm_90)	CUTLASS	0	0	0	0	–
H100	cuBLAS	37.9%*	0	0	0	–
H100 (CC)	cuBLAS	same*	0	0	0	–

* The 37.9% diffs occur only in `down_proj` at seq=100; the cuBLAS-dispatched kernel is from the proprietary `nvjet` family.