Autonomous Custom-Kernel Development on AWS Trainium: Fused RMSNorm, an Unsupported State-Space Model, and Where Hand-Written Kernels Beat the Compiler

Ali Asaria
Transformer Lab

Tony Salomone
Transformer Lab

Deep Gandhi*
Transformer Lab

Abstract

Non-NVIDIA accelerators such as AWS Trainium promise strong price-performance, but they demand hardware-specific kernel work that is scarce and expensive. We report an *autonomously conducted* study—design, kernel authoring, debugging, measurement, and analysis driven end-to-end by an agentic research system—of where custom Neuron-Kernel-Interface (NKI) kernels pay off on Trainium 1. We find a clean separation. At the *operation* level, a fused NKI RMSNorm kernel is 1.57× faster than a naive un-fused baseline (112 vs 176 μs at Llama-3.1-8B width), numerically bit-exact, and consistent with a memory-traffic roofline. *End-to-end*, however, the same kernel yields no measurable gain in a real Llama-3.1-8B block (0.999×): RMSNorm is ∼1% of block compute *and* the Neuron compiler already fuses it. We then push further and bring up **Mamba**, a non-transformer state-space model with no prior Trainium support: it runs on the chip, and we contribute the fused NKI kernel for its selective scan—the operation Neuron does not provide and which on GPUs requires a bespoke CUDA kernel. Our kernel is bit-exact (validated to the model's true 1536-wide state) and runs correctly *inside* the model on-device, giving Mamba a proper dedicated on-chip scan primitive rather than the non-scaling unrolled fallback. This kernel is a correctness-and-enablement milestone. We then implement a *parallel* scan—a Hillis–Steele prefix scan whose shifts are cast as matmuls against a constant shift matrix so they run on the systolic array—which is bit-exact and 1.14× faster than our sequential kernel, but still 0.74× the Neuron compiler's unrolled scan: Hillis–Steele lowers the serial *depth* yet raises total *work*, and Trainium here is throughput-bound. Closing the remaining gap needs a *work-efficient* chunked (Mamba2-style) scan, which Mamba-1's per-channel state transition complicates. Alongside the RMSNorm null, the study yields one decision-useful conclusion: hand-written kernels reliably beat *naive* baselines and are *necessary to enable* unsupported operations, while end-to-end speedups over the mature Neuron compiler come from *work-efficient parallel algorithms* (flash-attention, chunked scan), not naive fused, sequential, or work-inefficient kernels. We detail a device-free development methodology (simulator-validated correctness, on-device timing) that made the entire study cost only a few dollars of accelerator time.

1 Introduction

The economics of large-model training and inference increasingly favour non-NVIDIA silicon, but extracting that value requires kernels written to each accelerator's programming model. AWS

*Corresponding author: `deep@lab.cloud`

Trainium exposes the *Neuron Kernel Interface* (NKI), a Triton-like Python DSL. The open question for a practitioner is not *whether* custom kernels can be written, but *where they are worth writing*—which operations reward hand-optimization on Trainium and which the Neuron compiler already handles well.

This paper answers that question empirically for two representative operations, and does so under an unusual condition: the study was carried out *autonomously* by an agentic research system (Primus) that performed the literature review, formed the roofline hypothesis, authored and debugged the NKI kernels through a real bring-up gauntlet, ran and measured every job on a live `trn1.2xlarge`, and produced this report. Our contributions:

- A fused NKI **RMSNorm** kernel that is 1.57× faster than a naive un-fused baseline at the operation level, bit-exact, and roofline-consistent (§4).
- An honest **end-to-end null**: in a real Llama-3.1-8B block the same kernel gives 0.999×, with a mechanistic explanation (§4).
- A bring-up of **Mamba**—a non-transformer SSM with no Trainium support—plus a **fused NKI selective-scan kernel** that is bit-exact and runs correctly inside the model on-chip (§5).
- A **parallel** (matmul-shift Hillis–Steele) scan that is bit-exact and 1.14× faster than our sequential kernel, yet still 0.74× the compiler— with the diagnosis (depth reduced, work not) of what a compiler-beating scan actually requires (§5, §6).
- The resulting **central finding** on where custom kernels beat the compiler, and where a mature compiler already wins (§6).
- A device-free **methodology** that decouples simulator-validated correctness from on-device timing and keeps iteration cheap (§3).

2 Background

Trainium and NKI. Trainium 1 (NeuronCore-v2) is a systolic/vector accelerator programmed via PyTorch/JAX or, at lower level, NKI. NKI kernels tile computation across a partition axis (≤ 128) and a free axis, with explicit on-chip SBUF buffers. We use `neuronxcc` 2.26.6360 throughout .

RMSNorm. Root-mean-square normalization, $y = x/\sqrt{\text{mean}(x^2) + \epsilon} \cdot g$, appears in every transformer layer. It is bandwidth-bound: at bf16 its arithmetic intensity is ≈ 1.25 FLOP/byte , far below Trainium's compute:bandwidth ridge, so latency is set by memory traffic. Fusing the square, reduction, reciprocal-sqrt, and scale into one pass avoids intermediate HBM round-trips.

Mamba and the selective scan. Mamba [4] replaces attention with a state-space recurrence ("selective scan"): $h_t = \bar{A}_t\, h_{t-1} + \bar{B}_t x_t$, $y_t = \sum_n C_t h_t$. On GPUs this needs a hand-written CUDA kernel; on Trainium the Neuron stack provides no equivalent, so the operation falls back to an unrolled pure-PyTorch path.

Table 1: RMSNorm on `trn1.2xlarge` (512×4096). Op-level: fused vs naive un-fused. End-to-end: a real Llama-3.1-8B block, stock vs our kernel.

Setting	Baseline	Ours (fused NKI)	Speedup
Operation latency (μs)	176 (naive)	112	1.57×
Llama-3.1-8B block (μs)	10949 (stock)	10954	0.999×

3 Methodology

Kernel development on a scarce, finicky accelerator is dominated by iteration cost. We separate concerns:

- **Correctness, device-free.** We validate every kernel against an fp64 reference in the *NKI simulator* on a CPU host—no Trainium instance required. This caught all algorithmic bugs cheaply.
- **Timing, on-device.** We measure latency on a real `trn1.2xlarge`. For isolated kernels we use `nki.benchmark`; for in-model kernels we use `torch_neuronx` tracing. We found `nki.benchmark`'s *returned output* is subject to a nondeterministic readback race (1 corrupt row on one run, all rows on the next) while its *latency* is stable and reproducible; we therefore decouple correctness (simulator) from timing (device), which is standard kernel-benchmarking practice.

Every reported number traces to a specific recorded job (evidence ledger, Appendix). The full study consumed only a few dollars of accelerator time.

4 RMSNorm: an operation-level win and an end-to-end null

Op-level. We benchmark a fused single-pass NKI RMSNorm against a naive un-fused baseline (separate square/mean/rsqrt/scale, materialized through HBM) at Llama-3.1-8B width (512×4096, bf16). Both are bit-exact against fp64 in the simulator (fused 9.96e-07) . On-device (Table 1), the fused kernel runs at $112\,\mu$s (CI $[112, 113]$) versus $176\,\mu$s (CI $[176, 178]$) for the naive path—a 1.57× speedup with non-overlapping intervals. A memory-traffic model predicts a ceiling near 2.5× (naive moves ∼5 full-tensor transfers vs ∼2 for the fused kernel); the measured 1.57× sits below the ceiling, as expected once reduction overhead and imperfect bandwidth utilization are included.

End-to-end. We then place the kernel in a real Llama-3.1-8B decoder sub-block (`LlamaRMSNorm` + `LlamaMLP`) traced on-device, with the MLP identical across arms so the delta isolates the norm. The block runs at $10949\,\mu$s with stock RMSNorm and $10954\,\mu$s with our kernel—0.999×, within noise . Two mechanisms explain the null: RMSNorm is ∼1% of block latency (the MLP and data movement dominate), and the Neuron compiler already lowers `LlamaRMSNorm` to a fused kernel, so our hand-written version matches rather than beats it. Projected across all 65 RMSNorm ops in the model , the whole-model saving is negligible.

5 Mamba: enabling an unsupported architecture

Bring-up. `mamba-130m-hf`, a non-transformer SSM absent from the Neuron model zoo, loads and runs a correct CPU forward, and a real `MambaBlock` traces and executes on `trn1` via `torch_neuronx` : the architecture runs on the chip. Its selective scan lowers through the unrolled pure-PyTorch fallback, which does not scale in sequence length.

The kernel. We write a fused NKI selective-scan that precomputes the discretization in torch and keeps the recurrent state resident in SBUF across the sequence (an in-place loop-carried update). It is bit-exact against fp64: 2.50e-07 at a single tile and 2.75e-07 at Mamba's true inner width d_{inner}=1536 via partition-tiling .

In-model, on-chip. Running the Mamba mixer on `trn1` with our kernel in place of the sequential scan reproduces the reference mixer output to relative error 6.75e-08 : the kernel is a numerically-correct drop-in for Mamba's core operation, executing inside the model on the accelerator.

Enablement first, then throughput. The value here is *enablement*: Trainium had no selective-scan primitive, and our kernel provides a correct, in-model one—the analogue of the CUDA scan kernel that makes Mamba practical on GPUs, and a prerequisite for running the model at scale (the unrolled fallback does not scale in sequence length). What remains is throughput. Our first kernel realizes the scan *sequentially*, and that serial structure is its bottleneck: at L=256, d_{inner}=1536 it runs at 11635 μs versus 5980 μs for the compiler's unrolled form (0.51×) , because a strictly sequential recurrence over tiny tiles underutilizes a massively parallel machine while the compiler pipelines the unrolled version.

A parallel scan: faster than sequential, still short of the compiler. We then implement a parallel prefix scan. The recurrence $h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t$ is associative under $(a, b) \circ (a', b') = (aa',\ ab' + b)$, so a Hillis–Steele scan reduces the serial depth from L to $\log_2 L$. The obstacle is the per-step shift: the NKI simulator does not support vectorized free-axis window reads, so we place the sequence on the partition axis and realize "shift by 2^k" as a left-multiply by a constant $[L, L]$ shift matrix—a matmul on the systolic array. The kernel is bit-exact (validated at L=128, d_{inner}=1536; $\max_{\text{abs}}$ =5.09e-07, allclose) . On device, at L=128, d_{inner}=1536, the three arms run at 3646 μs (compiler), 5663 μs (sequential NKI), and 4954 μs (parallel NKI) : the parallel scan is 1.14× faster than our sequential kernel—the algorithmic direction pays off—but 0.74× the compiler, so it does not beat it. The reason is instructive: casting the shift matmul to bf16 (the PE array's native rate) yields *no* speedup, so the matmul is not the bottleneck ; Hillis–Steele lowers serial *depth* but its full-width tiles and dense shift matrices raise total *work*, and Trainium here is throughput-bound rather than latency-bound. (bf16 was also rejected on correctness: it breaks device accuracy, a failure the simulator masks because it does not model bf16 rounding.) Beating the compiler therefore requires a *work-efficient* chunked scan (the Mamba2/FlashMamba route), which Mamba-1's fully per-channel state transition—a distinct $[L, L]$ decay per channel, not a shared one— complicates; that is the identified next step.

6 Where hand-written kernels beat the compiler

Our two negatives are independent but share a mechanism: the mature Neuron compiler already parallelizes and fuses *naive* formulations effectively. Consequently:

- Custom kernels reliably beat *naive* baselines (RMSNorm 1.57×).
- Custom kernels are *necessary to enable* operations the stack does not support (Mamba gains a correct, in-model on-chip selective scan—the missing primitive—only because we wrote it).
- Enablement and throughput are separate milestones: our scan is correct and integrated; a parallel reformulation beats our own sequential kernel (1.14×) but still does not out-run the compiler (0.74×), because reducing serial depth is not enough— the algorithm must also be *work-efficient* (a chunked scan), the true remaining fix.
- Where the compiler already fuses or parallelizes an operation well, a hand-written kernel matches rather than beats it—whether a naive fused RMSNorm (0.999× end-to-end) or a depth-reduced-but-work-heavy parallel scan (0.74×).

The actionable implication for a Trainium roadmap: direct scarce kernel effort at *work-efficient parallel algorithms*—fused/flash attention, chunked state-space scans, communication-fused collectives—rather than naive fusions the compiler already captures or depth-only reformulations that add work.

7 Related work

Fused normalization kernels (Liger [5], FlashNorm [3]) establish the op-level fusion win on GPUs; our RMSNorm result transfers that to NKI/Trainium and adds an honest in-model null. Agentic and learned kernel generation for Trainium/NKI [1] and rigorous kernel-benchmarking methodology [2] inform our timing discipline. Mamba [4] and its hardware-aware selective scan motivate the SSM track; efficient parallel scans remain the open frontier we identify.

8 Limitations

On-device correctness for isolated kernels is established in the simulator rather than from `nki.benchmark`'s (racy) returned tensor; the in-model Mamba result and the parallel-scan device check verify correct device output directly. Latency units follow `nki.benchmark`'s reported values. The parallel scan is capped at $L \leq 128$ (the sequence rides the 128-lane partition axis) and is compared against both baselines re-timed at $L=128$; a *work-efficient* chunked scan—which we diagnose as the true requirement to beat the compiler, but which Mamba-1's per-channel state transition complicates—was not implemented. A notable methodology caveat surfaced here: the NKI simulator does not model bf16 matmul rounding, so a bf16 variant that passed in simulation failed on device; we therefore verify reduced-precision kernels against device output, not the simulator. Benchmarks use Trainium 1 (`trn1.2xlarge`); Trainium 2 was out of scope.

9 Conclusion

An agentic system autonomously developed, validated, and honestly evaluated custom NKI kernels on Trainium 1. The results delineate exactly where hand-written kernels help: they beat naive

baselines and enable unsupported architectures (a non-transformer SSM now runs, with a correct custom scan kernel), but they do not out-run the mature Neuron compiler on operations it already fuses or can parallelize. The frontier for custom-kernel value on Trainium is sophisticated parallel algorithms.

Availability. Code and reproduction package available on request.

References

[1] Anonymous. AccelOpt: A Self-Improving LLM Agentic System for AI Accelerator Kernel Optimization. *arXiv preprint arXiv:2511.15915*, 2025.

[2] Anonymous. SOL-ExecBench: Speed-of-Light Benchmarking for Real-World GPU Kernels. *arXiv preprint arXiv:2603.19173*, 2026.

[3] Nils Graef et al. FlashNorm: Fast Normalization for Large Language Models. *arXiv preprint arXiv:2407.09577*, 2024.

[4] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[5] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, et al. Liger Kernel: Efficient Triton Kernels for LLM Training. *arXiv preprint arXiv:2410.10989*, 2024.

A Evidence ledger

Every number in this paper is tagged `% [E`*n*`]` in the source and maps to a recorded Transformer Lab job on `trn1.2xlarge` (experiment `trn-llama-rmsnorm`); see `evidence.md`.