THOX.AI RESEARCH · PREPRINT V2.0.0 · 2026-11-05 · NOT PEER REVIEWED

thox-edge-ops v2: fused decode kernels for small models on THOX edge hardware

Six fused inference operators for CUDA, Metal and CPU, distributed through the Hugging Face Kernel Hub, with runtime ISA dispatch on x86 and a fused QK-norm and RoPE prologue for Qwen3 and Gemma 3 class models.

THOX.ai Research · Corresponding author: Tommy Xaypanya, CTO, THOX.ai LLC (Cedar Park, Texas)

Abstract. Small language models on edge hardware spend their decode step in launch overhead and memory traffic, not arithmetic. thox-edge-ops v2 is a set of six fused inference operators (RMSNorm with a weight offset that covers the Llama and Gemma conventions, fused residual-add RMSNorm, SiLU and tanh-GELU gated activations, rotary embedding, and a fused per-head QK-norm plus RoPE prologue) with one source tree and three backends: CUDA, Metal and CPU. The CPU backend selects AVX2, FMA and F16C row kernels at run time from a single baseline-ISA artifact, which is what lets one Kernel Hub build load on every x86_64 host. We measure the kernels against eager PyTorch on an NVIDIA RTX 4060 Ti and an x86 CPU with 30 interleaved trials per configuration. At decode shapes the GPU kernels run 6.90x faster than eager for RMSNorm and 24.43x faster for the fused QK-norm and RoPE prologue; on the CPU, RMSNorm runs 2.42x and rotary embedding 6.75x faster. We also report where fusion does not pay: the bfloat16 gated activations at prefill sizes on an AVX-512 host are slower than eager, and we say why. The release is live on the Kernel Hub as 23 ABI-checked x86_64 builds, and the full test suite passes against builds downloaded from it. All numbers in this paper are generated from the released raw measurements.

Executive summary
1 Introduction
2 Background and related work
3 System model and requirements
4 Deployment in the THOX stack
5 Design
6 Implementation
7 Evaluation
8 Discussion and limitations
9 Conclusion
10 Roadmap
A Reproducibility
B Notation
C Disclosure
References

Executive summary

6.90x	24.43x	2.42x	438 / 438
RMSNorm decode speedup, fp16, RTX 4060 Ti	fused QK-norm + RoPE decode speedup, fp16, RTX 4060 Ti	RMSNorm decode speedup, fp32, x86 CPU with AVX2 dispatch	tests passing on CUDA; the same suite passes on CPU, AVX2 and baseline ISA

thox-edge-ops v2 replaces the v1 release of three operators with six, adds the Gemma weight convention and the Qwen3 and Gemma 3 attention prologue, and repairs the source tree, whose head could not build v1 after a repository sync overwrote the build definition. The CPU backend gains runtime ISA dispatch so a single artifact uses AVX2, FMA and F16C where the host has them. The release is live on the Kernel Hub as 23 builds for x86_64 CPU and CUDA, and the full suite passes against 3 of them downloaded from the Hub.

> **PRODUCT IMPLICATIONS**
>
> ThoxWork(TM) model packs that use Gemma 3 or Qwen3 architectures map directly onto `weight_offset` and `fused_qk_norm_rope`. Section 4 sets out how the operators sit in the THOX stack and which hardware classes are measured, and which are targets only. The gains are operator-level; model-level throughput on target devices is measured next, not claimed here.

1 Introduction

Decoding one token at a time through a small transformer issues hundreds of kernel launches per step, most of which move a few kilobytes. On a GPU the launch cost dominates; on a CPU the interpreter and allocator overhead of the eager path dominates. Fusing the seams between normalization, residual update, gated activation and rotary embedding removes launches and intermediate tensors without changing the model. [1] introduced the serving-side vocabulary for these fused operators; the Hugging Face Kernel Hub [2] turned them into versioned, loadable artifacts.

THOX ships small models on heterogeneous devices: NVIDIA Jetson Orin, Apple silicon and x86 or ARM CPUs. One kernel family therefore has to cover three backends with identical numerics, and a single CPU build has to load on hosts with and without AVX2. The v1 release met the first requirement for three operators. This paper describes v2, which extends coverage to the architectures THOX actually ships (Gemma 3 [3] and Qwen3 [4]) and fixes the build and distribution defects found during the review that preceded it.

Contributions:

1. A weight-offset form of RMSNorm that serves the Llama and Gemma conventions from one kernel, with a single rounding of the output.
2. Two new gated activations and a rotary embedding kernel, each for CUDA, Metal and CPU, with a shared in-place contract.
3. A fused per-head QK-norm plus RoPE prologue that replaces the eager Qwen3 and Gemma 3 attention preamble with one launch per token.
4. Runtime ISA dispatch on x86_64 that keeps every AVX2 instruction inside one translation unit with no shared inline symbols, so the baseline-ISA Hub artifact cannot execute AVX2 code on a host that lacks it; we verify the property on the built object.
5. A measured evaluation on a GPU and a CPU host with 13,680 raw latency samples, including the configurations where the kernels lose to eager PyTorch.
6. A release on the Kernel Hub of 23 kernel-builder builds for x86_64 CPU and CUDA, each passing the manylinux and Python ABI checks, with the full test suite rerun against 3 builds downloaded from the Hub.

2 Background and related work

RMSNorm [5] drops the mean subtraction of LayerNorm and scales by the root mean square of the row. The Llama family [6] applies a learned weight w directly; Gemma [3] stores a zero-centred weight and applies $1 + w$ in fp32, a one-line difference that silently breaks any kernel that assumes the other. Rotary position embedding [7] rotates pairs of query and key coordinates by a position-dependent angle; two pairings are in use, the rotate-half layout of the GPT-NeoX lineage and the interleaved layout of GPT-J. Gated linear units with SiLU [8] and the tanh approximation of GELU [9] form the MLP of the Llama, Qwen and Gemma families respectively.

Qwen3 [4] and Gemma 3 [3] both normalize each query and key head with its own RMSNorm before applying RoPE. In eager PyTorch that prologue is two normalizations, two casts, a cos and sin gather, and the rotate-half arithmetic, roughly sixteen launches per layer; it is the highest-frequency seam in those decoders.

vLLM [1] maintains CUDA and ROCm implementations of fused residual RMSNorm, rotary embedding and gated activations inside its own tree; they are not distributed as standalone artifacts and have no Metal or CPU variants. The kernels library [2] and kernel-builder [10] define a Hub repository layout, a build matrix over PyTorch and CUDA versions, and an ABI check, which is the distribution model this work targets. The positioning table summarizes the delta.

System	Backends	Distribution	Gemma offset	QK-norm + RoPE fusion	CPU ISA dispatch
eager PyTorch (transformers)	any	n/a	per model class	no	ATen runtime dispatch
vLLM custom ops	CUDA, ROCm	in-tree	no	separate ops	n/a
thox-edge-ops v1	CUDA, Metal, CPU	Kernel Hub	no	no	baseline only
thox-edge-ops v2 (this work)	CUDA, Metal, CPU	Kernel Hub	yes, one kernel	yes, one launch per token	AVX2 at run time

Table 1. Positioning against the closest systems. Statements about other systems describe their public source trees as of this writing.

3 System model and requirements

A kernel call receives contiguous tensors of one dtype among float32, float16 and bfloat16 and returns in place or into a caller-provided buffer. The decode regime is one to a few dozen tokens per call with hidden sizes between 1024 and 2560 and head sizes of 128 or 256; prefill reaches a few hundred tokens. The hosts are a discrete or embedded NVIDIA GPU, an Apple GPU through MPS, or a CPU. A kernel artifact is downloaded once and may run on a different machine than the one it was fetched on.

1. **R1 Numerics.** Accumulate in fp32 for every dtype and round the output once; the fused residual sum is bit-identical to eager addition.
2. **R2 Conventions.** Serve both RMSNorm weight conventions and both rotary layouts from the same kernels through runtime flags.
3. **R3 Portability.** One CPU artifact loads and runs correctly on x86_64 hosts with and without AVX2, and on aarch64.
4. **R4 Composability.** Every op has a fake implementation so `torch.compile(fullgraph=True)` traces it without a graph break, and the layers slot into `kernelize`.
5. **R5 Safety.** No out-of-bounds access for any input the wrapper accepts; a position outside the rotary cache leaves that token untouched rather than reading past the cache.
6. **R6 Evidence.** Every published number is reproducible from the released scripts and raw data.

4 Deployment in the THOX stack

thox-edge-ops does not run alone on a THOX product. The deployment stack, from the silicon up, is: THOX edge hardware; ThoxInux or ThoxOS as the operating system; the MeshStack

runtime, which places and moves inference work across the devices; thox-edge-ops, loaded through `kernels.get_kernel` or applied to a model with `kernelize`; and the THOX small models themselves, in the Qwen3 and Gemma 3 shapes measured in Section 7. The operating system layer rests on the linux-thox kernel line (a Linux LTS fork). ThoxInux is the THOX Linux fork line published under Apache-2.0 (public since 2026-10-31); ThoxOS builds the THOX device experience on the same kernel line.

A shared kernel family matters to that stack because of R1 and R3 together. MeshStack can move a decode lease between nodes, a GPU hub, an AVX2 host and a baseline ISA host today, and heterogeneous devices as they qualify. That move is only safe if the operators produce identical numerics on every backend, which is what R1 fixes (fp32 accumulation, a single rounding, a fused residual stream bit-identical to eager addition), and only practical if one artifact loads across the fleet, which is what R3 requires. Portability without identical numerics would trade a launch saving for numerical drift between devices.

The operating system compatibility basis is narrow, and we state it as measured. The published Hub artifacts passed kernel-builder manylinux_2_28 and Python ABI checks and the `get_kernel` load check on Linux x86_64, and the full suite passed against 3 builds downloaded from the Hub. The artifacts have not yet been run on a ThoxOS or ThoxInux image, and no compatibility with a device image is claimed until roadmap item 1 runs the suite on the reference board under such an image.

Hardware class	Backend or target	Status for v2	Basis and gate
x86_64 workstation and hub class (NVIDIA RTX 4060 Ti GPU; x86 cloud sandbox, Xeon)	CUDA and CPU, AVX2 dispatch and baseline ISA	MEASURED	Full suite (438 / 438) and the benchmarks of Section 7 ran on this class. These are development and hub hosts, not THOX retail devices.
NVIDIA Jetson Orin	CUDA, compute capability sm_87 code target	NOT MEASURED	No Orin run was made for v2. The sm_89 ratios in Section 7 describe a desktop Ada GPU; they are not a Jetson result.
Apple silicon	Metal backend (MPS)	NOT RUN FOR V2	The Metal backend is in the source tree. The v1 Metal build was validated on an M4 Pro, per the v1 release record. v2 has not been run on a Mac, and no v2 Metal Hub build is published.

Hardware class	Backend or target	Status for v2	Basis and gate
Arm aarch64 device class. ThoxAir (Compute Class E2): quad-core efficiency Arm, 2 GB LPDDR4+ floor. ThoxMini (Compute Class P4): 8-core hybrid Arm, 4 GB LPDDR5+ floor, 3 TOPS NPU; reference board Orange Pi Zero 3W (Allwinner A733, 2 x A76 + 6 x A55), 12 GB SKU in hand. ThoxClip (Compute Class P6): the ThoxMini floors, with a 6 GB LPDDR5+ floor	CPU, aarch64 (source only)	TARGET ONLY	Target only for all three devices: no aarch64 Hub build is published for v2 and nothing was measured on Arm. Gated on roadmap item 1: an aarch64 build plus the suite on the reference board under a ThoxOS or ThoxInux image.
ThoxKey	None: host-assisted, with no on-device inference	OUT OF SCOPE	ThoxKey is explicitly out of scope for this work. No claim in this paper applies to it.

Table 2. Hardware qualification matrix for v2. MEASURED means the v2 suite and benchmarks ran on that hardware class. Every other status means they did not.

Two scope lines close the section. First, the x86_64 hosts in the MEASURED row are development and hub hosts, not THOX retail devices, so no result in this paper is a retail device result. Second, the small models at the top of the stack are the Qwen3 and Gemma 3 class models whose shapes Section 7 measures; model level throughput on any device class remains unmeasured, as Section 8 states.

5 Design

5.1 Operators

All six operators share one native signature across backends. With $x \in \mathbb{R}^H$ a row of the hidden state, $w \in \mathbb{R}^H$ the norm weight and $\delta \in \{0, 1\}$ the weight offset:

$$\mathrm{rms_norm}(x; w, \varepsilon, \delta)_i = \mathrm{round}_T\left(x_i \cdot \left(\tfrac{1}{H} \sum_j x_j^2 + \varepsilon \right)^{-1/2} \cdot (w_i + \delta) \right)$$

The fused residual step updates the residual stream and normalizes its rounded value in one pass:

$$r' = \mathrm{round}_T(x + r), \qquad x' = \mathrm{rms_norm}(r'; w, \varepsilon, \delta)$$

The gated activations take the concatenated gate and up projections $[g \,\|\, u] \in \mathbb{R}^{2d}$:

$$\mathrm{silu_and_mul}(g, u)_i = \frac{g_i}{1 + e^{-g_i}}\, u_i$$

$$\mathrm{gelu_tanh_and_mul}(g, u)_i = \tfrac{1}{2} g_i \left(1 + \tanh\!\left(\sqrt{\tfrac{2}{\pi}}\,(g_i + 0.044715\, g_i^3)\right)\right) u_i$$

Rotary embedding rotates the first ρ coordinates of every head of query and key by the angles cached for position t. With $c = \cos\theta_t, s = \sin\theta_t \in \mathbb{R}^{\rho/2}$ and pairs $(a_i, b_i) = (i, i + \rho/2)$ for the rotate-half layout or $(2i, 2i+1)$ for the interleaved layout:

$$h'_{a_i} = h_{a_i} c_i - h_{b_i} s_i, \qquad h'_{b_i} = h_{b_i} c_i + h_{a_i} s_i$$

The fused prologue applies a per-head RMSNorm with weight w_q or w_k of length equal to the head size, rounds to the input dtype, and then rotates, which is exactly the order of operations in the Qwen3 and Gemma 3 attention modules.

5.2 Fused residual RMSNorm

Algorithm 1: fused_add_rms_norm, one row

```
input: x[H], r[H] (dtype T), w[H], eps, delta
acc := 0 (fp32)
for each vector of lanes i:
    s := round_T(float(x[i]) + float(r[i]))      # residual update, rounded like eager
    r[i] := s
    acc += float(s)^2                            # square the rounded value
inv := rsqrt(acc / H + eps)
for each vector of lanes i:
    x[i] := round_T(float(r[i]) * inv * (float(w[i]) + delta))
```

Proposition 1 (residual bit-identity)

For any dtype T among fp32, fp16 and bf16, the residual written by Algorithm 1 equals the eager tensor `x + r` computed in T bit for bit.

Proof sketch. Eager addition in T converts both operands to fp32, adds, and rounds to nearest even in T. Line 4 performs the same three operations with the same rounding mode on every backend (the CUDA and Metal casts, the ATen `convert_from_float` helpers, and the AVX2 F16C and bf16 bit rounding all round to nearest even). The test suite asserts equality with zero tolerance.

Proposition 2 (single rounding)

The normalized output incurs one rounding to T after an fp32 computation whose relative error is bounded by a small constant multiple of the fp32 unit roundoff, independent of H up to the accumulation error of the sum of squares.

Proof sketch. Every multiply in line 9 is an fp32 operation on fp32 values; the sum of squares is a sequential fp32 accumulation over at most two vector accumulators, whose relative error is bounded by (H / lanes) u in the worst case and grows as the square root of that in practice. The reciprocal square root on CUDA (`rsqrtf`) and Metal (`rsqrt`) carries a two-ulp bound; on CPU it is a correctly rounded division and square root. The output rounding to T then adds at most half an ulp of T.

5.3 Runtime ISA dispatch on x86_64

Kernel Hub CPU builds target the baseline x86_64 ISA so the artifact loads on every host [10]. v1 therefore ran its norms with ATen's portable vector type, which on x86_64 without AVX2 flags is a scalar-backed implementation, and reached only parity with eager at prefill sizes. v2 compiles a second translation unit of row primitives for AVX2, FMA and F16C and selects it once per process with `__builtin_cpu_supports`.

The hazard in that design is COMDAT folding: if the AVX2 unit instantiated any inline function that the baseline unit also instantiates (a `c10::Half` conversion, an ATen vector helper), the linker may keep the AVX2 copy for the whole shared object and a baseline host would execute it. The AVX2 unit therefore includes no ATen or c10 header, is written with raw intrinsics on `uint16_t` storage for fp16 and bf16, places every helper in an anonymous namespace, and exports only uniquely named entry points. Standard library headers are included before the target pragma so their inline functions keep the baseline target.

Proposition 3 (ISA confinement)

No instruction that requires AVX, AVX2, FMA or F16C is reachable from a code path that executes before the dispatcher has confirmed host support.

Proof sketch. By construction the only functions compiled under the AVX2 target are those defined in the AVX2 unit after its pragma; all have internal linkage or a unique name in `thox_cpu_avx2`, so none can replace a baseline symbol. The dispatcher reads a function-local static initialized from `__builtin_cpu_supports` before the first call. We verified the built object with `objdump`: every instruction operating on a ymm register lies in a symbol of the `thox_cpu_avx2` namespace, and the full test suite passes with the dispatcher forced to the baseline path.

The AVX2 unit also provides vectorized exp and tanh (Cephes-style range reduction with a degree-5 polynomial) so the gated activations no longer delegate to ATen, and packs two vectors per store for fp16 and bf16, which halves the per-vector conversion cost of the bf16 round-to-nearest-even sequence.

5.4 Fused QK-norm and RoPE

On CUDA one block handles one token and one warp handles one head: the warp reduces the head's sum of squares with shuffles, normalizes and rounds in place, synchronizes, and rotates the first ρ coordinates. The Metal kernel maps the same structure onto simdgroups with `simd_sum`. On CPU the token loop is the parallel unit and the head loop reuses the norm and rotation primitives of the other operators. Query and key heads share one launch; a token whose position lies outside the cache is normalized but not rotated, which keeps R5 without a device synchronization on every decode step.

5.5 Security argument

The kernels are pure functions of their tensor arguments with no I/O, no allocation beyond the caller's output buffer, and no data-dependent control flow other than the position range check. Every entry point validates rank, shape, dtype, device and contiguity before any pointer arithmetic (R5), and the Python wrapper routes unsupported combinations to an eager reference rather than to native code. The artifact is loaded through `kernels` with `trust_remote_code` scoped to the THOX repository, and offline deployments pin an exact revision with a lock file, so the code that runs on a device is the code that was reviewed.

6 Implementation

The source tree holds one Python package, three backend directories and a build definition for kernel-builder [10]. The CUDA backend is about 560 lines, Metal about 500 across the host and shader files, and CPU about 900 across the dispatcher, the baseline primitives and the AVX2 unit. The test suite has 438 cases per backend covering three dtypes, hidden sizes from 37 to 2560, one to 64 rows, three-dimensional inputs, strided views, unaligned storage, empty tensors, in-place pointer identity, invalid arguments, layer swaps through `kernelize` with `use_fallback=False`, and `torch.compile(fullgraph=True)`.

Requirement	Test or check	Result
R1 numerics	`test_fused_add_rms_norm` (residual, zero tolerance) and all reference comparisons	pass on CPU AVX2, CPU baseline, CUDA (438 each)
R2 conventions	`test_rms_norm_gemma_offset`, `test_rotary_embedding` over both layouts, `test_fused_qk_norm_rope` over both offsets and layouts	pass
R3 portability	full suite with `THOX_EDGE_OPS_CPU_ISA=baseline`; `objdump` scan of ymm use	pass; ymm only in `thox_cpu_avx2`
R4 composability	`test_torch_compile_*` (zero graph breaks), `test_kernelize_replaces_rms_norm`	pass
R5 safety	`test_rotary_embedding_out_of_range_position_is_skipped`, argument rejection tests	pass
R6 evidence	`exp/edge_ops_experiments.py` raw JSON, `exp/analyze.py`, this document built from `summary.json`	released
Distribution	kernel-builder ABI and load check on every build; full suite against builds downloaded from the Hub with `get_kernel(version=2)`	23 of 23 builds pass; suite passes on 3 Hub builds

Table 3. Requirement to test mapping.

7 Evaluation

7.1 Setup

Host	Hardware	Software
GPU	NVIDIA GeForce RTX 4060 Ti (sm 8.9), host Intel(R) Core(TM) i7-14700F, Linux-6.18.33.2-microsoft-standard-WSL2-x86_64-with-glibc2.43	torch 2.14.1+cu126, CUDA 12.6, kernels 0.18.1
CPU	Intel(R) Xeon(R) Processor @ 2.80GHz, 2 vCPU, Linux-6.18.44-fc-v236-x86_64-with-glibc2.39	torch 2.15.0+cpu, kernels 0.18.1, 2 intra-op threads

Table 4. Measurement hosts.

Each configuration (operator, dtype, token count, width) runs 20 warm-up calls per path, then 30 trials; each trial times a block of 50 back-to-back calls with a device synchronization after the block and records the mean microseconds per call. Eager and kernel trials alternate so slow drift on the host affects both. We report the median across trials and the speedup as the ratio of medians; per-operator summaries are geometric means over configurations. Token counts are 1, 32 and 128; widths follow Qwen3-0.6B (hidden 1024, intermediate 3072, head 128), Gemma 3 1B (1152, 6912) and Gemma 3 4B and Qwen3-4B (2560, 10240, head 256 and

128). GPU dtypes are fp16 and bf16; CPU dtypes are fp32 and bf16. The eager baselines are written the way transformers writes them, including its double rounding in the Llama norm and the cos and sin gather of `apply_rotary_pos_emb`. Timings include Python dispatch, since that is what a decode loop pays.

7.2 Q1: how much do the fused kernels save on a GPU at decode?

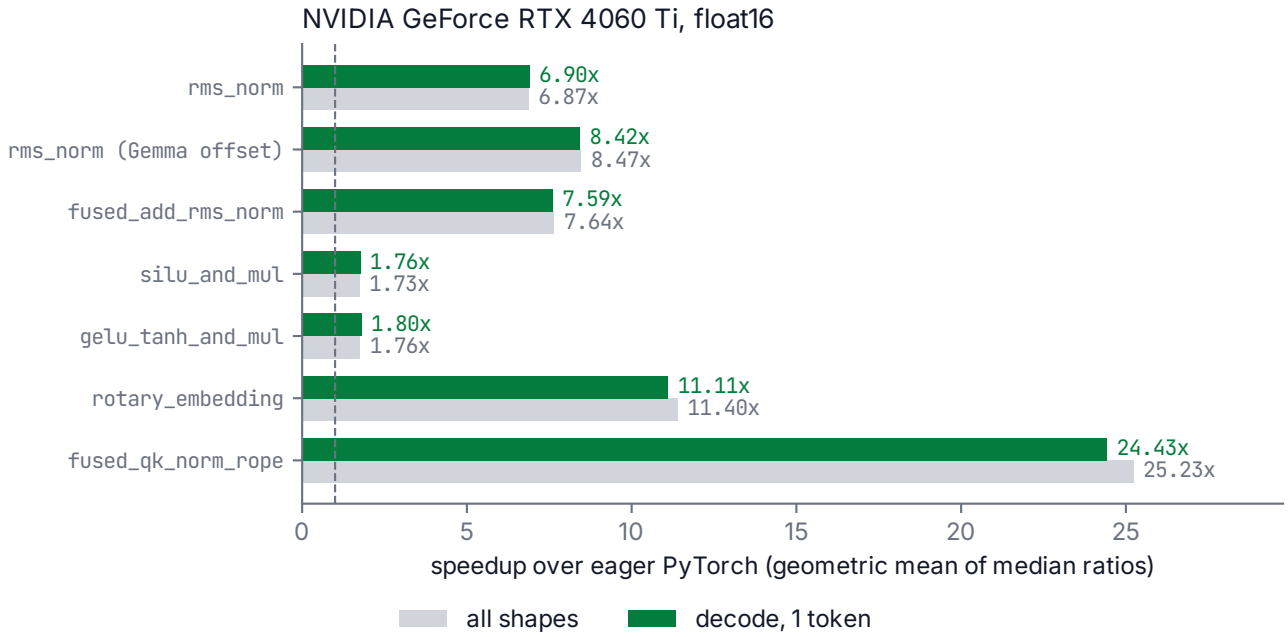


Figure 1. Speedup over eager PyTorch per operator on the NVIDIA GeForce RTX 4060 Ti, fp16. Dark bars: decode (one token); light bars: geometric mean over all shapes.

Every operator is launch bound at these sizes: the kernel latency barely moves between one and 128 tokens (RMSNorm at hidden 2560: 190.6 us at one token, 174.7 us at 128), and so does the eager latency, so the speedups are nearly constant across shapes. The fused QK-norm and RoPE prologue replaces about sixteen eager launches with one and runs 24.43x faster at decode (4,336.1 us eager versus 177.4 us, Qwen3-0.6B head layout); at 128 tokens the ratio is 29.89x. RMSNorm runs 6.90x faster, the fused residual step 7.59x, rotary embedding alone 11.11x. The gated activations gain the least (1.76x SiLU, 1.80x tanh-GELU) because eager already needs only two launches for them.

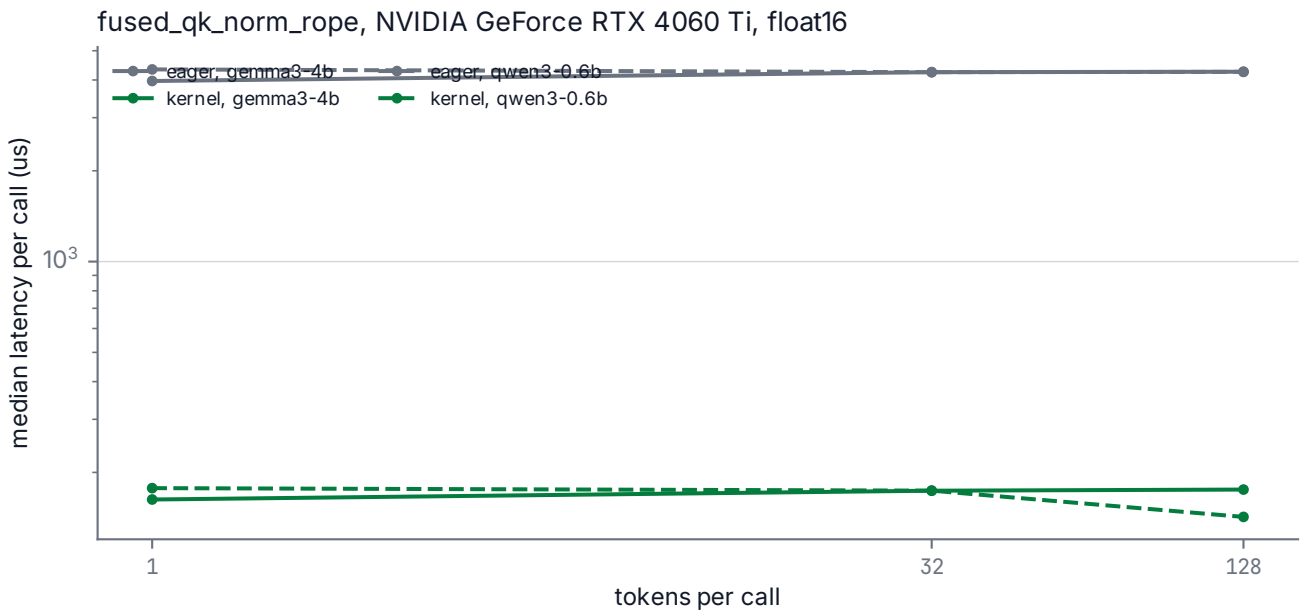


Figure 2. Median latency per call of the eager prologue and the fused kernel versus tokens per call on the GPU, fp16. Both lines are flat: the operator is launch bound up to 128 tokens.

bf16 behaves like fp16 on this host, within the trial spread. The absolute numbers belong to a desktop Ada GPU under WSL2; a Jetson Orin has the same instruction set (sm 8.7 versus 8.9) but a slower launch path, so the ratios should be read as the launch-bound regime of the kernel, not as a Jetson result.

7.3 Q2: what does runtime ISA dispatch buy on the CPU?

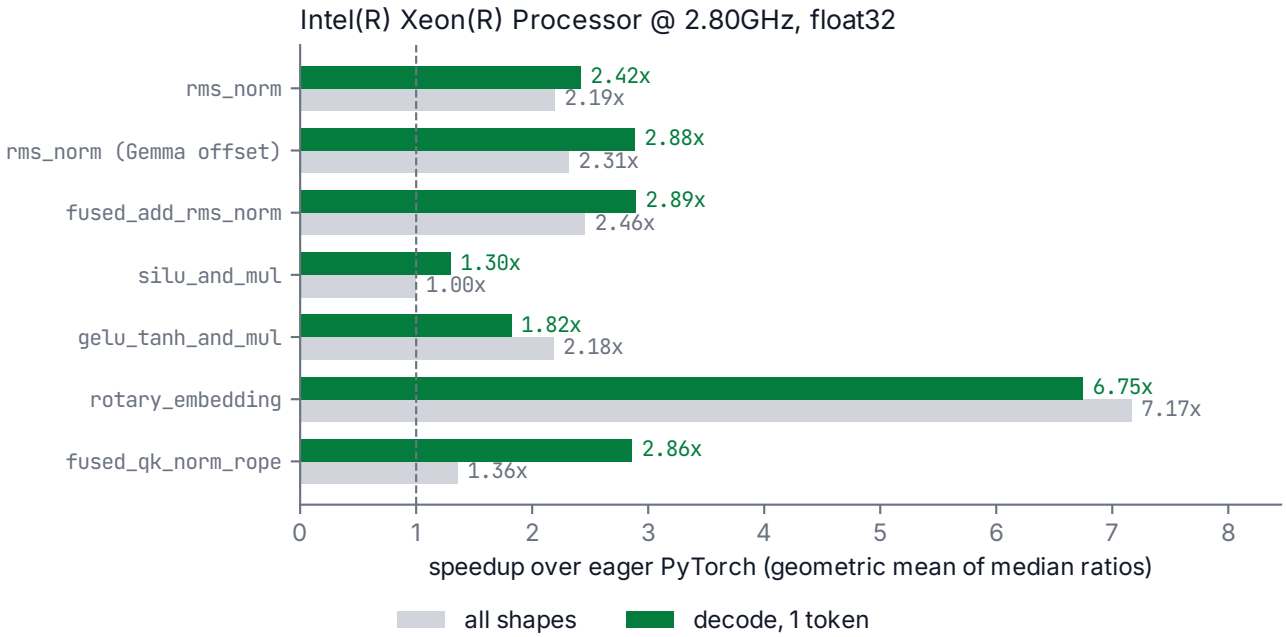


Figure 3. Speedup over eager PyTorch per operator on the CPU host, fp32, with AVX2 dispatch active.

At decode shapes the CPU kernels remove the eager dispatch chain: RMSNorm 2.42x, the fused residual step 2.89x, rotary embedding 6.75x and the fused prologue 2.86x faster than eager. The tanh-GELU gate gains 1.82x because eager evaluates its polynomial and tanh in separate passes; SiLU gains only 1.30x because ATen's runtime-dispatched SiLU is already a single fused pass.

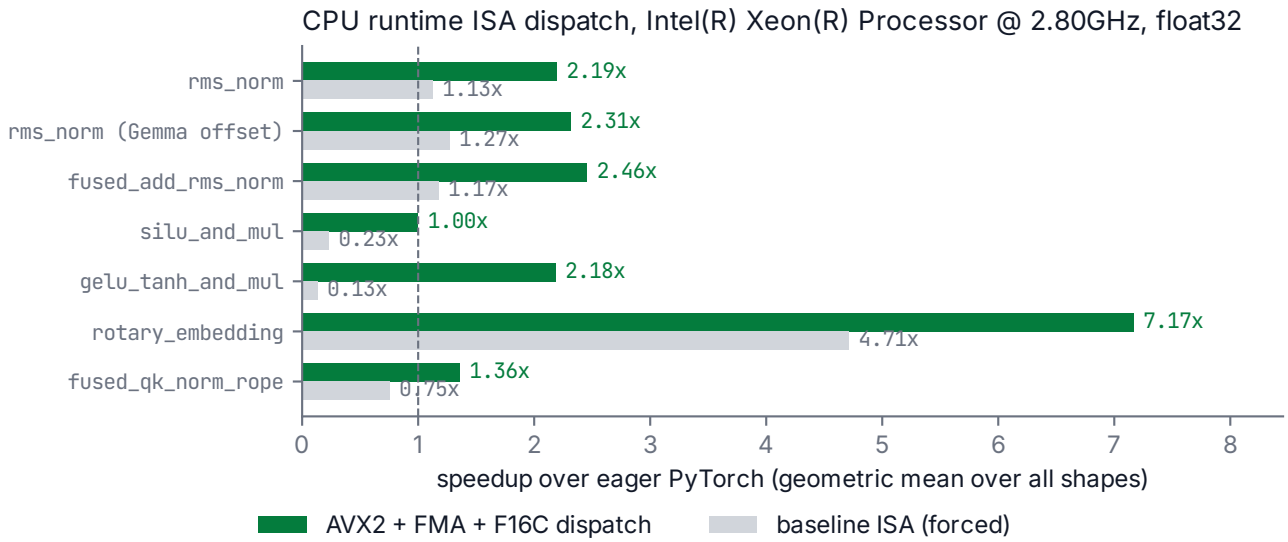


Figure 4. Geometric-mean speedup per operator with the AVX2 unit selected versus the same artifact forced to the baseline ISA through THOX_EDGE_OPS_CPU_ISA=baseline, fp32.

The same artifact forced to the baseline path reaches 1.13x on RMSNorm, 0.13x on tanh-GELU and 4.71x on rotary embedding against 2.19x, 2.18x and 7.17x with AVX2. That difference is what a v1-style baseline-only build leaves on the table on every AVX2 host.

7.4 Q3: where does fusion not pay, and why?

Three negative results shape the recommendations. First, bf16 SiLU at prefill sizes on this CPU is slower than eager (0.64x at 128 tokens, intermediate 6912; 0.57x at worst), and fp32 SiLU is at parity (0.87x). The host has AVX-512, which ATen uses at run time for the eager path with sixteen lanes and a vendor exp; the kernel's AVX2 path has eight lanes. At prefill the operator is bound by that arithmetic and by memory traffic that fits in L2, so eager's extra pass costs little. Second, the fused prologue at 128 tokens on the CPU is 0.81x: the per-head norm of eager runs as one wide reduction over all heads, which ATen parallelizes better than the token-parallel loop of the kernel. Third, bf16 norms at prefill gain less than fp32 because the bf16 round-to-nearest-even store costs more instructions than an F16C conversion.

The recommendation that follows is to keep the kernels on the decode path and, for CPU prefill of gated activations in bf16 on AVX-512 hosts, either route to eager or add an AVX-512 unit; the latter is scoped in the roadmap. We do not hide these configurations behind a heuristic in this release, because the crossover depends on the host and we have measured only one.

7.5 Q4: are the numerics within contract?

All 438 tests pass on the GPU host (fp16, bf16, fp32) and on the CPU host with the AVX2 unit selected and with the baseline path forced. Tolerances against the fp32 eager reference are 1e-5 for fp32, 2e-3 for fp16 and 1.6e-2 for bf16; the fused residual stream is asserted bit-identical to eager addition with zero tolerance; the fused prologue is asserted equal to `rms_norm` followed by `rotary_embedding` at 1e-5. The AVX2 exp and tanh approximations are covered by the same tolerances, which hold with margin for the sampled inputs.

7.6 Q5: do the published artifacts honour the contract?

A build that passes on a developer host can still fail once packaged. kernel-builder [10] built all 23 variants in a clean environment, checked each shared object for manylinux_2_28 and the stable Python ABI, and loaded each through `get_kernel`. We then removed the local build from the loader path, downloaded builds from the Hub and ran the full suite against them on real hardware (Table 5).

Hub build	Host	Result
`torch215-cpu-x86_64-linux`	cloud x86_64 sandbox, torch 2.15.0+cpu	438 passed
`torch214-cu126-x86_64-linux`	the THOX RTX 4060 Ti workstation WSL2, NVIDIA GeForce RTX 4060 Ti, torch 2.14.1+cu126	438 passed
`torch213-cu130-x86_64-linux`	the THOX RTX 4060 Ti workstation WSL2, NVIDIA GeForce RTX 4060 Ti, torch 2.13.0+cu130	438 passed

Table 5. Test suite against builds downloaded from the Kernel Hub.

8 Discussion and limitations

- **Hosts.** One GPU (NVIDIA GeForce RTX 4060 Ti, under WSL2) and one CPU (Intel(R) Xeon(R) Processor @ 2.80GHz, 2 vCPU in a cloud sandbox). No Jetson Orin, no Apple silicon and no aarch64 CPU was measured for v2. According to the v1 release record, the Metal backend was compiled and validated on an M4 Pro for v1, and that build remains on the Hub's v1 branch; the v2 Metal code is in the tree, builds in the same way and is tested by the same suite, but it has not yet been run on a Mac, so no Metal result appears here.
- **Scope.** Operator-level latency only. We did not measure a model end to end, tokens per second, or energy; those are the next experiments, not claims.
- **Statistics.** 30 trials per configuration with 50 calls each; medians and quartiles are released. The cloud CPU host showed drift between runs of up to 15 percent on the smallest shapes, which is why we interleave the two paths.
- **Unbuilt.** No AVX-512 unit, no aarch64-specific tuning beyond ATen's NEON vector type, no backward pass, no paged KV write, no attention kernel.

- **Release.** 23 ×86_64 builds (CPU and CUDA) are on the Hub. Each passed kernel-builder's ABI and load checks; the full suite ran against 3 of them on real hardware, not all 23. No Metal or aarch64 build is published for v2; v1's Metal build remains on its branch.

9 Conclusion

v2 turns thox-edge-ops from a three-operator proof into the operator set the THOX model families need at decode, with one source tree for three backends, one CPU artifact that uses AVX2 where it exists and stays correct where it does not, and an evaluation that reports the losses next to the wins. The fused QK-norm and RoPE prologue is the largest single gain and the clearest argument for continuing to fuse seams rather than operators.

10 Roadmap

1. Qualify the target hardware in this order: first, an aarch64 Linux host, the Orange Pi Zero 3W reference board, under a ThoxOS or ThoxInux image, with an aarch64 Hub build and the full suite on that board; second, a Metal run on Apple M4 Pro; third, a run on Jetson Orin. Publish the numbers in the kernel card and update the qualification matrix in Section 4.
2. Add an AVX-512 row unit under the same confinement rules and measure the prefill crossover.
3. Measure a Gemma 3 1B and a Qwen3-0.6B decode loop end to end with `kernelize`, in tokens per second and tokens per joule.
4. Paged GQA attention with sliding window and a KV write fused into the prologue, as a separate Hub repository.
5. Fused MoE dispatch and a quantized matvec (NVFP4, FP8, Q4_K) with the quantization contract in the manifest.
6. WebGPU twins of the decode kernels under the Hugging Face WebGPU kernel contract, with a per-device tuning database.

A Reproducibility

```
# load the published release
pip install -U kernels torch
python -c "from kernels import get_kernel; get_kernel('Thox-ai/thox-edge-ops', version=2,
trust_remote_code=['Thox-ai/thox-edge-ops'])"

# build and test the CPU backend from source
pip install torch --index-url https://download.pytorch.org/whl/cpu
pip install kernels pytest ninja numpy
python scripts/local_build.py --backend cpu
```

```
export LOCAL_KERNELS=Thox-ai/thox-edge-ops=$PWD/build-local
python -m pytest tests/test_edge_ops.py -q
THOX_EDGE_OPS_CPU_ISA=baseline python -m pytest tests/test_edge_ops.py -q

# CUDA host (WSL2 or Linux with an NVIDIA GPU)
bash scripts/wsl_cuda_validate.sh all

# benchmarks and paper
python exp/edge_ops_experiments.py --label <host-label> --trials 30 --calls 50
python exp/analyze.py
python wp/figures.py && python wp/build.py
node wp/katex_render.cjs wp/out/paper.raw.html wp/out/paper.html
python wp/pdf.py wp/out/paper.html wp/out/thox-edge-ops-v2.pdf
```

Raw results: `exp/results/*.json` (git commit 1a247366730c for the CPU run, 31aa1b72c9f3 for the GPU run). Summary: `exp/summary.json`.

B Notation

Symbol	Meaning
H	hidden size (row length of a norm)
d	intermediate size of the gated MLP (half the input width)
ρ	rotary dimension, at most the head size
δ	weight offset: 0 for Llama and Qwen, 1 for Gemma
round_T	round to nearest even in dtype T
u	unit roundoff of fp32

Table 6. Notation.

C Disclosure

This preprint has not been peer reviewed. The corresponding author, Tommy Xaypanya, is responsible for the content. No AI system is listed as an author. Measurements were taken on hardware owned by THOX.ai LLC and on a cloud sandbox; no third-party figures are presented as THOX results.

References

1. Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang and Ion Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. SOSP 2023; arXiv:2309.06180. https://arxiv.org/abs/2309.06180

2. Hugging Face. *Kernels API Reference (kernels library documentation)*. huggingface.co/docs/kernels, accessed 2026-11-04. https://huggingface.co/docs/kernels/en/api/kernels
3. Gemma Team (Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin and 116 additional authors). *Gemma 3 Technical Report*. arXiv:2503.19786, 2025. https://arxiv.org/abs/2503.19786
4. An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng and 54 additional authors. *Qwen3 Technical Report*. arXiv:2505.09388, 2025. https://arxiv.org/abs/2505.09388
5. Biao Zhang and Rico Sennrich. *Root Mean Square Layer Normalization*. arXiv:1910.07467, 2019. https://arxiv.org/abs/1910.07467
6. Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971, 2023. https://arxiv.org/abs/2302.13971
7. Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen and Yunfeng Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv:2104.09864, 2021 (v5, 2023). https://arxiv.org/abs/2104.09864
8. Noam Shazeer. *GLU Variants Improve Transformer*. arXiv:2002.05202, 2020. https://arxiv.org/abs/2002.05202
9. Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. arXiv:1606.08415, 2016 (v5, 2023). https://arxiv.org/abs/1606.08415
10. Hugging Face. *Build with Nix (kernel-builder documentation)*. huggingface.co/docs/kernels/builder/build, accessed 2026-11-04. https://huggingface.co/docs/kernels/builder/build