# Data-Movement-Aware Optimization of Transformer Attention on Gemmini

Ivan Lok
ELEC 5140 Advanced Computer Architecture — Final Project Report

May 2026

**Abstract**

Transformer attention on Gemmini already uses hardware row-wise softmax, yet the official schedule still materializes the full $N \times N$ attention matrix in DRAM between softmax and $PV$. We treat this as a *data-movement* problem: the tensor is produced and consumed on-chip but routed through off-chip memory.

We first eliminate the DRAM round-trip by fusing softmax output into scratchpad (Experiment 1). At seq=128 this wins ($\sim$1.51$\times$ on the attention region) because saved DMA traffic exceeds control overhead. At longer sequences the same design fails: built-in full-row softmax forces an $O(N)$ mvout storm per Q-block, and $Q_{\text{block}}$=16 multiplies Q-block count by another $O(N)$ factor—together $O(N^2)$ RoCC traffic that dominates real matmul time and eventually crashes the interconnect at seq=512.

We abandon extending the shared Normalizer (Experiments 2/3) and build a dedicated **OnlineAttention** module that maintains FlashAttention-style running max/sum state, scans accumulator tiles in batch, and emits int8 weights to scratchpad without per-element mvout commands. A phased integration (Experiments 4–5) removes CPU-side score extraction, adopts single-pass $K_{\text{block}}$=seq_len, merges UPDATE+WEIGHTS into one RoCC opcode, and sweeps $Q_{\text{block}}$ against accumulator double-buffer capacity.

Final Verilator results (BERT-base, 12 heads, seq $\in \{128, 256, 512\}$): **1.63M / 3.27M / 7.43M** total cycles vs. baseline **2.15M / 3.97M / 8.34M** (1.12–1.32$\times$), with zero attention-matrix DRAM traffic and stable scaling where Experiment 1 slows $\sim$6.4$\times$ or crashes. The central lesson is that *fused data movement was right early*; the bottleneck was softmax *mechanism* and *tiling granularity*, not the fusion idea itself.

## Contents

# 1 Introduction

On Gemmini, BERT-base attention already runs as tiled matmuls with hardware softmax on the accumulator/normalizer mvout path. This project asks how much cost remains when the $N \times N$ weight matrix $P$ is still written to DRAM between softmax and $PV$, and what limits performance once that traffic is removed.

We study encoder attention in Verilator ($H$=768, 12 heads, $d$=64, seq_len $\in \{128, 256, 512\}$). Section 2 describes Gemmini memories, the baseline dataflow, and built-in softmax (Section 2.5). Baseline profiling quantifies stage cycles and $O(N^2)$ attention-buffer traffic. Experiment 1 fuses softmax output into scratchpad and shows why that path cannot scale. Experiments 2/3 reject extending the shared Normalizer; Experiment 4 integrates OnlineAttention (Section 7); Experiment 5 reports end-to-end cycles, fuses UPDATE+WEIGHTS, and sweeps $Q_{\text{block}}$ against accumulator limits.

# 2 Background: Gemmini, Attention, and Prior Art

## 2.1 Prior work and project framing

**Gemmini** [1] is a parameterized systolic-array generator in Chipyard. **FlashAttention** [2] streams $K/V$ blocks and maintains online softmax statistics $(m, \ell)$ so the full $N \times N$ matrix never sits in GPU HBM. Our target is different: a RoCC-attached accelerator with separate scratchpad and accumulator, a built-in `Normalizer`, and i-BERT fixed-point paths [4]. We do not build a compiler; we quantify how the official Transformer sample moves data and where the $QK^\top$+softmax $\rightarrow PV$ boundary hurts.

## 2.2 Gemmini system overview

Figure 1 shows the host–accelerator split. The **Rocket** core issues **RoCC** commands; Gemmini's **Controller** schedules DMA (`mvin`/`mvout`), dependency tracking, and address translation. Operands enter the $D_{\text{im}} \times D_{\text{im}}$ **systolic array** (here $D_{\text{im}}$=16) only through **scratchpad** (SP): 4 banks × 4096 rows, 16 bytes/row (∼256 KiB). Matmul outputs land in the **accumulator** (ACC, int32 tiles), not back in scratchpad. Row-wise softmax and LayerNorm run on the `mvout` path: DMA reads ACC, the **Normalizer** FSM updates per-row statistics across $J$-tiles, then **AccumulatorScale** applies `iexp` and narrowing before writeback.

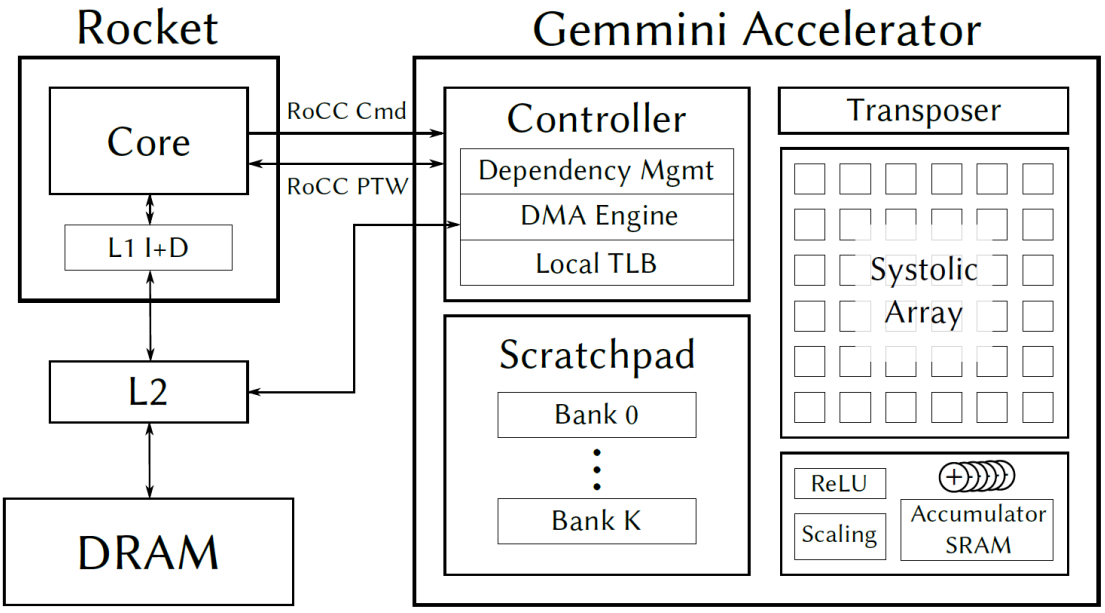



Figure 1: Rocket + Gemmini: RoCC commands from the core; DMA to L2/DRAM; scratchpad banks feed the systolic array; results and reductions use the accumulator (with scaling/ReLU) before writeback.

Scratchpad stages inputs; accumulator holds matmul outputs and reduction state. There is no unified on-chip tensor RAM—which port a tensor uses determines its datatype and consumer. That split drives the baseline attention-matrix DRAM round-trip.

Gemmini names matmul dimensions $I$, $J$, $K$. With ACC_ROWS=2048 and weight-stationary **double buffering**, only half the accumulator (∼1024 rows, ∼64 tiles) is active per matmul phase; exceeding that forces striping and becomes a knob in our $Q_{\text{block}}$ study (Section 9).

## 2.3 Attention in three stages

One head (sequence $N$, dimension $d$) decomposes as in the official sample: **(A)** $Q, K, V$ projections then $S = QK^\top$; **(B)** row-wise $P = \text{softmax}(S/\sqrt{d})$ (int8 weights); **(C)** $O = PV$ and output projection. $P$ is an intermediate tensor only between (B) and (C); the baseline stores it in DRAM anyway.

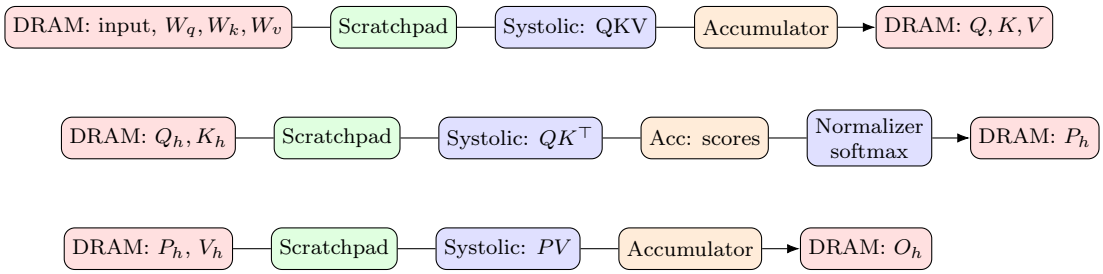
## 2.4 How the official Transformer sample runs attention on Gemmini

The Chipyard/Gemmini **Transformer attention sample** (our baseline) implements the encoder sublayer with `tiled_matmul_auto` and `SOFTMAX`. Below we trace data movement.

**Part 1 — QKV.** For each of $W_q, W_k, W_v$: `mvin` loads tiles, the systolic array writes to ACC, `mvout` stores $Q$_buf, $K$_buf, $V$_buf ($N \times 768$ each). At $N$=128 this is ∼960K cycles (∼45% of baseline)—mostly required weight/activation traffic.

**Part 2 — $QK^\top$ + softmax (per head).** `mvin` loads $Q_h, K_h$; `tiled_matmul_auto`($I$=$N$, $J$=$N$, $K$=$d$) with `transB=true` writes int32 scores to ACC. With `activation=SOFTMAX`, softmax is *not* in the mesh: a three-pass `mvout` sequence (MAX, SUM_EXP/INV_SUM_EXP, RESET) runs through the Normalizer (Section 2.5). Normalized int8 weights `mvout` to DRAM `attn_buf`$[h]$—the first leg of the round-trip.

**Part 3 — $PV$ (per head).** `mvin` reloads `attn_buf`$[h]$ as matmul $A$, loads $V_h$ as $B$, computes $PV$, `mvout` to `out_buf`. The `attn_buf` read is the second leg—data that was on-chip one phase earlier.




At $N$=128, $P_h$ alone accounts for $N^2$=16,384 int8 elements per head; write+read $\Rightarrow 2 \times 12 \times N^2 \approx$ 393 KiB per layer of avoidable DRAM traffic.

## 2.5 Softmax on Gemmini: accumulator residency, mvout, and the Normalizer

After $QK^\top$, each row is a length-$N$ vector of int32 logits in ACC SRAM. Softmax is row-wise along $J$; columns in a row share max and sum, so weights cannot finalize until the full row is visible. Software therefore sets `tile_J`= $N/D_{\text{im}}$ with `tile_K`= 1, `tile_I`= 1—partial-$J$ tiling is unsupported.

`activation=SOFTMAX` tags StoreController state and embeds a `norm_cmd` in each `gemmini_extended_mvout` address; softmax does not run in the PE mesh. The mvout path walks three queues (`write_dispatch_q` → `write_norm_q` → `write_scale_q`); only `RESET` enqueues to the scale queue and writes memory. For each row, software scans all $J$-blocks in three passes:

| Phase | Intermediate $J$-blocks | Final $J$-block |
|---|---|---|
| 0 | MAX (5) | MAX (5) |
| 1 | SUM_EXP (6) | INV_SUM_EXP (7) |
| 2 | RESET (0) | RESET (0) |

Phase 0 finds $m$; phase 1 accumulates $\sum_j \text{iexp}(x_j - m)$ and computes `inv_sum_exp` = $127/\sum$ on the last $J$-tile; phase 2 applies AccumulatorScale and emits int8. The Normalizer keeps two `stat_id` slots so two rows can overlap while one waits on the divider. Both Normalizer and AccumulatorScale share the same `iexp` polynomial (I-BERT constants via `config_norm`); $z \geq 32$ saturates to zero.

For row vector $x$,

$$\text{softmax}(x_i) = \frac{e^{x_i - m}}{\sum_j e^{x_j - m}}, \quad m = \max_j x_j.$$

The library issues one mvout per small slice (row batch × pass × $J$-block). At $N$=256 with a 16-row Q-block, $N_{\text{mvout/QB}} \approx 48 \times (N/D_{\text{im}}) = 768$ RoCC-class commands per Q-block, before multiplying by $N/16$ Q-blocks and 12 heads—the $O(N^2)$ control storm that breaks Experiment 1 (Section 5). Rows of $S$ are independent; columns within a row are not.

Experiment 1 tried `GEMMINI_LOCAL_STORE_FLAG` (bit 39) to steer mvout to scratchpad; examined RTL does not decode it on the TileLink writer, so it is a software convention. OnlineAttention (Section 7) uses `io.sp_write` instead: same `iexp`, streaming $(m, \ell)$ over up to 128–256 Q-rows, batched RoCC iteration, no per-slice mvout—the mechanism change Experiment 1 could not get by redirecting DRAM alone.

Table 1: Where each attention tensor lives during baseline execution.

| Tensor | Shape / head | After which step | Physical location |
|---|---|---|---|
| $Q, K, V$ | $N \times d$ | QKV matmul | DRAM buffers |
| $S$ (pre-softmax) | $N \times N$ | $QK^\top$ | Accumulator (int32) |
| $P$ (weights) | $N \times N$ | softmax mvout | DRAM `attn_buf` |
| $P$ for $PV$ | $N \times N$ | mvin before $PV$ | Scratchpad (int8 $A$) |
| $O$ | $N \times d$ | $PV$ | DRAM `out_buf` |

### 2.6 Constraints that motivated later design choices

Built-in SOFTMAX requires full-row ACC residency and a three-pass mvout loop per row, blocking cheap $J$ tiling. WS double buffering caps $QK^\top$ tile size at ACC_ROWS/2. At long sequences, fine-grained mvout loops dominate—RoCC overhead, not bandwidth alone. Fixed-point `iexp` ($z \geq 32 \to 0$) needs hardware-matched verification. Removing DRAM for $P$ (Experiment 1) was necessary but insufficient; we replaced the Normalizer softmax *mechanism* while keeping fused data movement.

## 3 Methodology

### 3.1 Simulation environment

All results are from Verilator simulation of `IBertGemminiRocketConfig` ($D_{\text{im}}$=16, WS dataflow, ACC_ROWS=2048, 16 parallel `iexp` lanes in OnlineAttention). Cycles are read from a RISC-V cycle counter bracketing hardware phases.

### 3.2 Measurement discipline

Experiment 5 enforced three rules learned from early runs: (1) no CPU reference softmax inside timed regions (once inflated seq=512 to ∼10M cycles); (2) no UART `printf` in hot paths (∼$10^2$K cycles at $N$=128); (3) separate attention-core sweeps (config outside timing) from full-pipeline numbers (config inside).

### 3.3 Bottleneck taxonomy

We label each configuration memory-bound (DRAM attn traffic), schedule-bound (RoCC/mvout count), accumulator-capacity-bound (WS spill), or compute-bound (systolic matmul). The goal is to shift the dominant class by changing data movement and tiling, then measure what limits next.

## 4 Baseline Characterization

Section 2 traced baseline data movement; here we quantify time and confirm the attention-buffer round-trip is the main removable cost at $N$=128.

Table 2: Official baseline (cycles, seq=128).

| Stage | Cycles | Share of total |
|---|---|---|
| QKV projection | 960,126 | 44.7% |
| $QK^\top$ + softmax | 345,240 | 16.0% |
| $PV$ | 266,050 | 12.4% |
| Attention core (combined) | 611,290 | 28.4% |
| $W_o$ + LayerNorm | 496,826 | 23.1% |
| **Total** | **2,148,501** | 100% |

### 4.1 Cycle breakdown at seq=128

Attention is not the largest single stage (QKV is), but it is the best target for **avoidable** traffic: QKV must touch weights and activations at least once; $P$ does not need DRAM at all.

### 4.2 Quantifying the attention-matrix round-trip

Per head, int8 $P$ has $N^2$ elements. Baseline traffic:

$$\text{DRAM}_{\text{attn}} \approx 2 \times N^2 \times \texttt{sizeof(elem\_t)} \quad \text{(write + read)}$$

For $N$=128, 12 heads: ∼393 KiB; for $N$=512: ∼6.3 MiB per layer. This grows as $O(N^2)$ while $P$ remains an ephemeral tensor—the classic motivation for FlashAttention-style fusion, adapted here to Gemmini's scratchpad/accumulator split.

## 5 Experiment 1: Data-Movement Fusion and Its Scaling Collapse

### 5.1 Hypothesis

If $P$ never leaves the chip, we should save DMA setup, TileLink transfers, and scratchpad fill latency for the $PV$ $A$ operand. Experiment 1 implements **scratchpad-local fusion** for $P$: the benchmark lays out int8 weights in scratchpad tiles and runs $PV$ with `A=NULL`, eliminating the DRAM write/read of `attn_buf`. Software tags mvout addresses with `GEMMINI_LOCAL_STORE_FLAG` (bit 39) as a convention; examined RTL does not decode that bit to bypass TileLink (Section 2.5). The $N$=128 win therefore comes from the **benchmark schedule** (where $P$ lives between softmax and $PV$), not from a new hardware mvout-to-scratchpad route—which is why the built-in Normalizer's full-row mvout storm still dominates at longer sequences.

This is **not** monolithic kernel fusion. Each Q-block still runs separate $QK^\top$+SOFTMAX and $PV$ matmuls. We only change *where* softmax writes—DRAM → scratchpad—preserving the built-in three-pass Normalizer algorithm.

### 5.2 Success at seq=128

DRAM traffic for $P$ drops to zero. Attention core cycles fall from ∼611K to ∼358K (∼41%); full layer from ∼2.15M to ∼1.88M. At this length, eliminated DMA dominates the extra on-chip mvout work: only 8 Q-blocks per head and 8 $J$-tiles per row keep the mvout storm small (∼3,072 mvout-class operations per head).

### 5.3 Analytical failure model: two $O(N)$ constraints multiply

Experiment 1 fails at scale for reasons visible in the microarchitecture, not bad luck.

**Constraint A — full-row softmax inside `tiled_matmul`.** When activation is `SOFTMAX`, tiling fixes `tile_J` $= N/D_{\text{im}}$: the entire row width must be processed. After each $Q_{\text{block}}$ matmul, the normalizer

iterates all $J$-blocks with multiple passes (max, sum-of-exp, normalize). Per Q-block mvout count scales as:

$$N_{\text{mvout/QB}} \approx 48 \times \frac{N}{D_{\text{im}}}$$

(e.g. $N$=256 $\Rightarrow$ 768 mvout-equivalent RoCC operations per 16-row Q-block).

**Constraint B — $Q_{\text{block}} = D_{\text{im}} = 16$.** The local-store addressing path only supported $Q_{\text{block}}$=16, so Q-blocks per head = $N/16$ grow linearly with $N$.

**Combined effect.** Total mvout-scale work per head:

$$N_{\text{mvout/head}} \approx \frac{N}{16} \times 48 \times \frac{N}{16} = O(N^2).$$

At $N$=256: ∼12,288 mvout calls per head, ∼147,456 per layer. Timed matmul (∼1.47M cycles for QK+PV) explains only a fraction of the measured ∼6.68M attention-core cycles; ∼5.2M cycles are **unaccounted overhead**—consistent with dispatch-dominated fine-grain stores (∼78% of attention at $N$=256).

Table 3: Experiment 1 scaling summary (attention core unless noted).

| $N$ | mvout/head | Attn cycles | vs baseline attn | Outcome |
|---|---|---|---|---|
| 128 | 3,072 | ∼358K | ∼0.59× | Win |
| 256 | 12,288 | ∼6,681K | ∼6.4× | Catastrophic loss |
| 512 | 49,152 | — | crash @ ∼204M | TL assertion |

### 5.4 Why seq=512 crashes

At $N$=512, ∼590K local-store writes per layer hammer the scratchpad port shared with DMA readers. Congestion produces unexpected TileLink message types on a read-oriented path (PutFull on a read-only port) and triggers a protocol monitor assertion. The baseline avoids this path by using explicit DRAM mvout/mvin rather than the high-rate local-store side channel.

### 5.5 What Experiment 1 taught us

- **Data-movement fusion is necessary but not sufficient.** Removing DRAM for $P$ is the right direction.
- **Keeping built-in full-row softmax** preserves the $O(N^2)$ mvout structure; changing the destination does not change the algorithm.
- $Q_{\text{block}}$ **granularity is a first-class design knob**, as important as fusion itself.

## 6 Why We Rejected Experiments 2 and 3 (Normalizer-Based Online Softmax)

Before building OnlineAttention, we tried to extend Gemmini's existing `Normalizer` with online-softmax state—conceptually close to FlashAttention's $(m, \ell)$ updates but wired into shared infrastructure.

This path failed for **integration** reasons, not algorithmic ones:

- Extending `NormCmd` bit encodings risked breaking existing `SOFTMAX` and LayerNorm commands tied to address-bit shifts.
- Online state reused `stat_id` slots; concurrent $PV$ or mvin traffic could corrupt running max/sum.
- At $D_{\text{im}}$=16, rescaling still required many RoCC-visible steps per row (∼4× per row vs. one batched command in OnlineAttention), exploding to tens of thousands of ops at $N$=512.

- Every debug cycle touched a large, stateful module shared with unrelated features—long rebuild-test loops and fragile regressions.

**Design decision:** allocate a dedicated RoCC function (funct=23), private per-row state arrays, and batch internal iteration over column chunks. The Normalizer returns to baseline-only softmax/LayerNorm; online attention becomes an opt-in datapath with explicit mutual exclusion on accumulator ports.

# 7 The OnlineAttention Module: Online Softmax in Depth

Experiment 4's central deliverable is **OnlineAttention**—a dedicated RoCC module (funct 23) that replaces the batch three-pass Normalizer mvout path (Section 2.5) with **streaming online statistics** over accumulator tiles. This section explains the algorithm, how data enters and leaves the block, how the FSM iterates, and the hardware cost. Integration milestones (Phases 4A–4M) are summarized afterward in Section 8.

## 7.1 Placement in the attention pipeline

For each head and Q-block, the scalar core orchestrates three hardware phases:

1. $QK^\top$ **matmul** (`gemmini_loop_ws`): $Q$ and $K$ tiles move through the systolic array; **int32 scores** land in the accumulator at a known base address (WS double-buffer offset tracked in software).
2. **Online softmax** (`OP_FUSED_BATCH` in Experiment 5): OnlineAttention reads those scores, updates per-row $(m, s)$ state, writes **int8 weights** to scratchpad—no DRAM, no Normalizer mvout loop.
3. $PV$ **matmul**: reads weights from scratchpad with `A=NULL`; $V$ via internal MVIN.

OnlineAttention never performs matrix multiply; it is a **reduction + activation + layout** engine between accumulator and scratchpad.

## 7.2 Online softmax algorithm (per row)

FlashAttention-style online softmax maintains, for each query row $i$, a running maximum $m_i$ and sum $\ell_i$ as key columns arrive in chunks. When a new chunk reveals a larger maximum, previous contributions to $\ell_i$ must be rescaled.

For chunk with local scores $\{x_j\}$ (16 values per hardware step):

$$m_i' = \max(m_i,\ \max_j x_j) \tag{1}$$

$$\ell_i' = \begin{cases} \sum_j \mathrm{iexp}(x_j - m_i') & \text{first chunk} \\ \ell_i \cdot \mathrm{iexp}(m_i - m_i') + \sum_j \mathrm{iexp}(x_j - m_i') & \text{if } m_i' > m_i \\ \ell_i + \sum_j \mathrm{iexp}(x_j - m_i') & \text{otherwise} \end{cases} \tag{2}$$

After all $K$-columns are processed, unnormalized weights are $\mathrm{iexp}(x_j - m_i)$; final softmax weights use $\ell_i$ as the normalization denominator (applied in the WEIGHTS phase when writing int8 tiles for $PV$).

Built-in softmax (Section 2.5) needs three full-row mvout passes through the Normalizer before any output; OnlineAttention updates $(m, \ell)$ incrementally as $D_{\mathrm{im}}$-wide column chunks arrive from ACC.

## 7.3 Where data lives before OnlineAttention runs

**Input: accumulator score matrix.** After $QK^\top$, scores for the current Q-block occupy contiguous accumulator rows. Each logical element is **int32** (systolic output width). Physically, data is stored in $D_{\mathrm{im}} \times D_{\mathrm{im}}$ tiles: one accumulator row holds one lane of one tile; a row of $N$ scores spans $N/D_{\mathrm{im}}$ tile-columns $\times$ $D_{\mathrm{im}}$ lanes.

Software passes:

- `scores_addr` — accumulator base of the $QK^\top$ result (must match WS buffer half).
- `total_cols` = $K_{\mathrm{block}}$ (typically $N$).
- `num_rows` = Q-block height (e.g. 128 or 256).
- `sp_addr` — scratchpad base where int8 $A$ tiles for $PV$ will appear.

**Output: scratchpad weight matrix.** WEIGHTS writes packed int8 vectors (16 bytes per chunk) into scratchpad banks. $PV$ then uses `A=NULL`—the matmul loader skips mvin for $A$ because weights are already resident.

**Why scores are not buffered inside OnlineAttention.** A full $128 \times 128$ int32 block is $\sim$64 KiB—impractical to duplicate next to 3 KiB of state. The design **re-reads** the accumulator during WEIGHTS. The accumulator is idle during `gemmini_fence()` between $QK^\top$ and softmax; re-read bandwidth is on-chip SRAM, not RoCC dispatch per element. OnlineAttention uses the **raw ACC bypass** path (reads without an mvout entry in `write_norm_q`), which Section 2.5's batch SOFTMAX path does not use.

## 7.4 Hardware state and configuration

**Per-row registers** (parameter `onlineAttentionMaxRows`, 128 then 256):

| Register | Width | Role |
|---|---|---|
| `max_state[r]` | int32 | Running row maximum $m$ |
| `sum_state[r]` | int32 | Running sum $\ell$ (unnormalized exp sum) |
| `state_valid[r]` | 1 bit | Whether row has seen at least one chunk |
| `rescale_mul[r]` | int32 | $\text{iexp}(m_{\text{old}} - m_{\text{new}})$ when max increases |

Storage: $R \times (2 \times 32 + 1 + 32)$ bits $\approx 3$ KiB for $R$=256—modest compared to scratchpad ($\sim$256 KiB) or accumulator ($\sim$128 KiB).

**Iexp configuration** (four `OP_CONFIG` writes once per layer): Coefficients for $\text{iexp}(z) \approx 2^{az^2+bz+c}$ (same `AccumulatorScale.iexp` as baseline softmax). Software sets `ONLINE_BERT_SCALE=0.05` so scaled scores stay in range; hardware saturates $z \geq 32 \to 0$.

**Compute datapath. 16 parallel combinational `iexp` units** (one per $D_{\text{im}}$ lane) form the only non-trivial arithmetic. Everything else is comparators, muxes, and the 7-state FSM—control-dominated, not a second systolic array.

## 7.5 Internal iteration model

One RoCC command (`BATCH_UPDATE`, `BATCH_WEIGHTS`, or `OP_FUSED_BATCH`) performs **all** inner iteration in hardware:

$$\texttt{total\_chunks} = \lceil \texttt{total\_cols}/D_{\text{im}} \rceil$$

For $N$=128: 8 chunks per row; for $N$=512: 32 chunks per row.

Nested loops (conceptual):

```
for row in 0 .. num_rows-1:
  for chunk in 0 .. total_chunks-1:
    read 16 int32 scores from acc[row, chunk*16 : chunk*16+15]
    UPDATE or WEIGHTS micro-sequence (FSM)
```

The CPU issues **one** RoCC instruction; the FSM walks rows and chunks, issuing accumulator read requests and (in WEIGHTS) scratchpad writes until `s_done`.

Compare to built-in softmax: $\sim 48\times(N/16)$ mvout-class RoCC ops *per 16-row Q-block*—OnlineAttention reduces softmax control to **1–2 RoCC ops per Q-block** (fused: 1).

## 7.6 FSM micro-sequence (7 states)

States: `s_idle` → `s_read_req` → `s_read_wait` → `s_compute` → `s_update` → `s_write` → `s_done`.

**UPDATE path (per chunk).**

1. **Read:** request one $D_{\text{im}}$-wide slice from accumulator (`bits.data`, int32).
2. **Compute:** 16-input max tree (MSB-flip trick for signed compare); merge with `max_state[cur_row]`; if max increased, compute `rescale_mul` = iexp($m_{\text{old}} - m_{\text{new}}$).
3. **Update:** apply `iexp` to (score$-m_{\text{new}}$) for all 16 lanes; sum lanes → `sum_exp`; merge into `sum_state` with rescaling if needed; advance chunk or row.

**WEIGHTS path (per chunk).** Re-read the same accumulator slice; set `iexp_offset` $= m$; compute iexp(score $- m$); clamp to $[0, 127]$; pack int8; **single-cycle scratchpad write** (no TileLink mvout).

**OP_FUSED_BATCH (Experiment 5).** For each row, run all UPDATE chunks, then flip `fused_phase` and run all WEIGHTS chunks **without a second RoCC dispatch**:

```
for row:
  phase=UPDATE:  chunks 0..C-1  (update m, s)
  phase=WEIGHTS: chunks 0..C-1  (write int8 to SP)
```

Saves one `gemmini_fence()` + RoCC round-trip per Q-block (∼400 cycles measured).

## 7.7 Worked example: one row, three $J$-chunks ($N = 48$, $D_{\text{im}} = 16$)

Table 4: Streaming UPDATE for one query row (illustrative scores).

| Step | Chunk scores (16 cols each; only 4 shown) | $m$ | Action on $\ell$ |
|---|---|---|---|
| Init | — | $-\infty$ | $\ell = 0$ |
| 1 | $[3, 1, 4, 1, \ldots]$ | 4 | $\ell \leftarrow \sum e^{x-4}$ |
| 2 | $[7, 2, 1, 3, \ldots]$ | 7 | $\ell \leftarrow \ell \cdot e^{4-7} + \sum e^{x-7}$ (rescale) |
| 3 | $[2, 5, 1, 2, \ldots]$ | 7 | $\ell \leftarrow \ell + \sum e^{x-7}$ (max unchanged) |

WEIGHTS then re-reads each chunk, emits int8 iexp$(x-m)/\ell$ (fixed-point equivalent) into scratchpad tiles for $PV$.

## 7.8 Hardware cost summary

Table 5: OnlineAttention resource summary (IBert config).

| Resource | Cost |
|---|---|
| Chisel RTL | ∼700 lines (`OnlineAttention.scala`) |
| Controller + scratchpad mux | ∼85 lines added |
| State SRAM | $R \times 3 \times 32$ bits ($R$=256 ⇒ 3 KiB) |
| `iexp` units | 16× combinational (reused design) |
| RoCC functs | 23 (commands); 24 reserved for macro-sequencer |
| Accumulator port | Shared with executor via arbiter + busy mux |
| Scratchpad write | Dedicated OR-mux port (priority over DMA) |

No additional systolic PEs, no FP units, no duplicate score SRAM—the module is deliberately **small control logic** that removes an $O(N^2)$ RoCC dispatch pattern.

### 7.9 Contrast with built-in Normalizer softmax

| | Built-in `SOFTMAX` (Section 2.5) | OnlineAttention |
|---|---|---|
| Trigger | `mvout` + `norm_cmd` in ACC addr; `act=SOFTMAX` | RoCC batch cmd after $QK^\top$ |
| Scope | Full row $J$ in ACC; offline 3-pass | Streaming $K$-blocks; 128–256 row state |
| Column handling | MAX, SUM_EXP, RESET mvout scans | Chunk-wise UPDATE of $(m, \ell)$ |
| `iexp` | Normalizer + AccumulatorScale (shared RTL) | Same function, local regs |
| RoCC ops @ $N$=256, 16-row QB | ∼768 mvout-class | 1 fused batch |
| Output path | DRAM or LOCAL_STORE mvout | Direct `io.sp_write` |
| ACC access | StoreController mvout → norm → scale | Raw bypass read/write; arbiter with executor |

# 8 Experiment 4 Integration: Step-by-Step Validation

Experiment 4 is not a single leap; it is gated hardware/software bring-up validating Section 7 at increasing scale. Early projections underestimated matmul cost and overestimated `gemmini_loop_ws` overhead—measurement drove the design.

## 8.1 Phase progression and what each gate proved

**Bring-up (Phases 4A–4D).** Establish RoCC decoding, accumulator read/write FSM, and `iexp` numerical agreement with hardware-saturated reference ($z \geq 32 \to 0$). Integrate raw accumulator bypass reads after `gemmini_fence()`—without bypass, reads could deadlock behind an empty normalization queue. Confirm systolic-produced int32 scores (not MVIN-truncated test paths) feed UPDATE correctly.

**Micro-kernel (Phase 4F).** End-to-end online softmax on $16 \times 16$ tiles validates WS double-buffer offsets and multi-row state before BERT dimensions.

**First BERT integration (Phase 4G)–per-row RoCC.** Functional correctness with 12 heads, but ∼14× slower than Experiment 1 on attention—dominated by **per-row** commands and CPU-side score movement, not matmul arithmetic.

**Batch commands (Phase 4H).** `BATCH_UPDATE` and `BATCH_WEIGHTS` collapse inner loops to one command per Q-block per phase, roughly halving attention time vs. 4G—proving control-plane cost was the early killer.

**Single-pass $K$ (Phase 4I).** Setting $K_{\text{block}} = N$ removes outer $K$ loops so each head processes one score rectangle—matching Experiment 1's "one big $K$" intuition but with OnlineAttention instead of Normalizer mvout.

**Wrong turns we measured, not assumed.**

- **Scratchpad-resident $K/V$ with CPU extraction** looked algorithmically elegant but spent ∼170K cycles/head moving scores on the scalar core—far more than DRAM internal MVIN saved.
- **Larger `onlineAttentionMaxRows` alone (Phase 4K)** did not help at $N$=512 once benchmarks were apples-to-apples: $Q_{\text{block}}$=256 overflows WS accumulator halves (256 tiles vs. 64 tile slots/half), and DMA striping tax cancels fewer Q-blocks.

**Breakthrough (Phase 4M).** Keep $K$ in DRAM but let $QK^\top$ use **internal MVIN** ($B$ non-null) so the CPU never extracts tiles. Combined with batch softmax, full-layer totals reach ∼1.79M at $N$=128—faster than Experiment 1's ∼1.88M with the same zero-DRAM property for $P$.

### 8.2 What "fused" means in Experiment 4 vs. Experiment 1

Both experiments are **data-movement fused**: $P$ lives in scratchpad between softmax and $PV$. Neither fuses $QK^\top$, softmax, and $PV$ into one RoCC macro-op. The performance gap comes from:

| | Experiment 1 | Experiment 4/5 |
|---|---|---|
| Softmax engine | Built-in Normalizer (full-row) | OnlineAttention (batched scan) |
| Typical $Q_{\text{block}}$ at $N$=128 | 16 | 128 |
| RoCC softmax work per Q-block | $O(N)$ mvout storm | 1 fused batch cmd |
| Scaling at $N$=256 | 16 Q-blocks/head | 1–2 Q-blocks/head |

At $N$=256, Experiment 4 issues on the order of **24** Q-block iterations per layer vs. **192** for Experiment 1—an 8× reduction in outer-loop control before counting inner mvout differences.

## 9 Experiment 5: Trustworthy Measurement and the $Q_{\text{block}}$ U-Curve

Experiment 5 is the scientific capstone: hold RTL fixed, remove measurement artifacts, pick $Q_{\text{block}}$ from evidence, and report full-layer cycles.

### 9.1 Improvements over Experiment 4

- `OP_FUSED_BATCH`: one command per Q-block for UPDATE→WEIGHTS (∼1% attention savings alone; ∼400 cycles/Q-block in A/B tests).
- Zero `printf` inside timed brackets.
- Attention-core sweep with config *outside* timing; full pipeline with config *inside* (realistic).

### 9.2 $Q_{\text{block}}$ sweep (attention core only)

Holding $K_{\text{block}}$=$N$, we swept $Q_{\text{block}} \in \{16, 32, 64, 128, 256\}$:

Table 6: Attention-core cycles vs. $Q_{\text{block}}$ (Experiment 5 sweep).

| $N$ | $Q$=16 | 64 | 128 | 256 | Optimal |
|---|---|---|---|---|---|
| 128 | 374K | 251K | **238K** | — | 128 |
| 256 | 955K | 531K | 486K | **471K** | 256 |
| 512 | 3,504K | 1,982K | **1,825K** | 1,883K | 128 |

**Interpretation.** For $N \in \{128, 256\}$, maximal $Q_{\text{block}}$ wins even when accumulator occupancy exceeds 100%: **RoCC dispatch savings dominate** spill cost. At $N$=512, $Q_{\text{block}}$=256 becomes *worse* than 128 (+3.2%)—the U-curve tipping point where 8× ACC overflow DMA exceeds benefit from halving Q-block count. This is a concrete hardware–software trade-off: larger on-chip accumulators would likely shift the optimum rightward.

## 10 Evaluation

### 10.1 Full-layer results (Experiment 5)

### 10.2 Scaling relative to Experiment 5 at $N$=128

| Configuration | 128 → 256 | 256 → 512 |
|---|---|---|
| Baseline total | 1.85× | 2.10× |
| Experiment 1 | 5.07× | crash |
| **Experiment 5** | **2.00**× | **2.27**× |

Table 7: Full BERT attention sublayer cycles (Verilator).

| $N$ | Config | QKV | Attn | $W_o$ | Norm | Total | vs BL |
|---|---|---|---|---|---|---|---|
| 128 | Baseline | 960K | 611K | 309K | 188K | 2,149K | 1.00× |
| 128 | Exp 1 fused | 962K | 358K | 309K | 176K | 1,882K | 1.14× |
| 128 | **Exp 5** | **913K** | **245K** | **309K** | **164K** | **1,631K** | **1.32×** |
| 256 | Baseline | 1,851K | 1,038K | 610K | 395K | 3,973K | 1.00× |
| 256 | Exp 1 | ∼1,851K | ∼6,681K | ∼610K | ∼395K | ∼9,536K | 0.42× |
| 256 | **Exp 5** | **1,804K** | **486K** | **609K** | **369K** | **3,269K** | **1.22×** |
| 512 | Baseline | 3,629K | 2,623K | 1,204K | 801K | 8,336K | 1.00× |
| 512 | Exp 1 | | CRASH (TileLink assertion) | | | | |
| 512 | **Exp 5** | **3,582K** | **1,882K** | **1,204K** | **759K** | **7,427K** | **1.12×** |

Experiment 5 tracks near-linear layer scaling because attention no longer suffers $O(N^2)$ *control* growth.

### 10.3 Bottleneck shift across the project

Table 8: Dominant bottleneck by phase.

| Phase | Dominant limit | Evidence |
|---|---|---|
| Baseline | DRAM attn traffic + matmul | ∼393 KiB/head round-trip |
| Exp 1 @ $N$=128 | DRAM removed; matmul wins | 1.51× attn |
| Exp 1 @ $N$=256 | RoCC/mvout schedule | 78% overhead vs matmul |
| Exp 4G | CPU extraction + per-row cmds | ≫ Exp 1 despite fusion |
| Exp 4M–5 | Matmul + ACC spill trade-off | U-curve at $N$=512 |

At $N$=512, attention rises to ∼25% of Experiment 5 total cycles (vs. ∼15% at $N$=128)—expected $O(N^2)$ matmul growth in the attention core, not pathological control.

### 10.4 Hardware cost (modest)

OnlineAttention adds ∼700 lines of Chisel in `OnlineAttention.scala` (Table 5), ∼3 KiB of per-row state (256 rows × three int32 fields), and reuses existing `iexp` logic. It does not enlarge the systolic array or scratchpad capacity—the remaining headroom question is whether ACC_ROWS should double to unlock larger $Q_{\text{block}}$ at long sequences.

## 11 Discussion

### 11.1 Three design lessons (aligned with our presentation)

**1. $Q_{\text{block}}$ granularity is king.** Once DRAM traffic for $P$ is gone, performance is often set by how many times the scalar core launches matmul and custom-op sequences—not by the arithmetic inside the array.

**2. Fused data movement was correct from day one.** Experiment 1 proved the traffic analysis; failure was misattributed if one concluded "fusion does not work." The softmax *implementation* had to change.

**3. Dedicated hardware beats overloading shared units.** The Normalizer path couples unrelated concerns (LayerNorm, softmax, local store). OnlineAttention isolates online statistics, simplifies verification, and enables batch commands without risking baseline operators.

### 11.2 Measurement honesty

We report cycle counts only from timed Verilator regions, but several early milestones were wrong until we re-measured:

- **CPU work inside the timer.** A seq=512 bring-up run still called CPU LayerNorm inside the timed bracket, inflating totals by millions of cycles. After moving all reference checks outside the timer, Phases 4J and 4K matched within ∼0.005% at the same configuration.
- **UART `printf` in hot paths.** Debug prints added $\sim 10^2$K cycles at $N$=128; Experiment 5 benchmarks omit them entirely in timed code.
- **Config and sweep semantics.** $Q_{\text{block}}$ sweeps time only the attention core with configuration *outside* the bracket; full-layer Experiment 5 numbers include realistic per-run setup *inside* the bracket (Section 3.2). Comparing a kernel-sweep row to a full-layer total without reading the caption will mis-rank designs.
- **Plans vs. measurements.** We once assumed a macro-sequencer or CPU score extraction would be mandatory to beat Experiment 1; Phase 4M showed internal MVIN removed the extraction tax, so we stopped drawing architectural conclusions from projections alone.

Experiment 5 is the conservative capstone: same RTL, `OP_FUSED_BATCH`, no `printf`, and paired baseline/Exp 1/Exp 5 scripts in the submission tree (Appendix A) so reported speedups are reproducible from the bundled `code/` layout, not from a private log directory.

## 11.3 Limitations

This report's claims are bounded by what we actually built and measured.

- **Simulation only.** All cycle counts are Verilator runs of `IBertGemminiRocketConfig`; we report no FPGA frequency, critical-path closure, or energy.
- **Scope of the workload.** We time a single BERT-base encoder attention sublayer with *static* weights loaded once—not a full 12-layer model, not training, and not end-to-end accuracy regression after quantization changes. Numerical checks use hardware-matched references *outside* timed regions (Section 11.2).
- **Fixed Gemmini geometry.** Results use $D_{\text{im}}$=16, WS dataflow, and ACC_ROWS=2048. Larger $Q_{\text{block}}$ at long sequence is partly capped by accumulator capacity (Section 5); we did not sweep silicon parameters in this submission.
- **Incomplete attention-core fusion.** Intra-attention DRAM traffic for the attention matrix $P$ is eliminated, but $PV$ still **reloads $V$ from DRAM** on every Q-block. Scratchpad residency for $V$ (or chaining stages in hardware) remains open.
- **Scalar-core orchestration in the hot path.** Even in Experiment 5, the CPU loops over heads and Q-blocks and issues separate RoCC/matmul commands plus `gemmini_fence` between $QK^\top$, online softmax, and $PV$. That dispatch overhead is real but not modeled as its own line item in every table.
- **Verification environment.** Some TileLink assertions were relaxed after we showed they were over-constraining legal burst patterns during bring-up. A tape-out would need correct back-pressure and FIFO sizing, not disabled monitors.
- **What the tables do and do not include.** Kernel sweeps bracket only the attention core; full-layer numbers include projections, LayerNorm, and setup inside the timed window. Comparing rows across sections requires reading the caption and measurement rules (Section 3.2).

## 11.4 Future work

Three directions follow directly from the bottleneck-shift story above; two shorter items sit at the end.

**1. Macro-sequencer (RoCC funct=24 reserved).** Today, $QK^\top$, online softmax, and $PV$ are **three separate** Gemmini/RoCC invocations. The scalar core walks heads and Q-blocks, programs each stage, and fences between them so accumulator state and scratchpad layout stay consistent. That pattern is easy to debug but expensive: every phase pays RoCC decode, CSR setup, and `gemmini_fence` latency even when scratchpad already holds the tiles the next stage needs.

We reserved RoCC `funct`=24 for a **macro-sequencer**: one custom command that walks a fixed micro-program inside the accelerator—$QK^\top \rightarrow$ OnlineAttention $\rightarrow PV$—without scalar-core orchestration in the inner loop. The Chisel cost should stay modest (a small FSM plus reuse of existing matmul and OnlineAttention hooks); the win is eliminating per-phase dispatch and fence storms when $Q_{\text{block}}$ is small and head count is 12. A fair evaluation would compare baseline and Experiment 5 *both* with the macro enabled, so speedup is not confounded with "fewer CPU instructions" alone.

**2. Larger accumulator (ACC_ROWS) and fair baselines.** Our $Q_{\text{block}}$ sweep showed a U-shaped curve: at $N$=128 smaller blocks win on control traffic; at $N$=512 the optimum moves upward but ACC_ROWS=2048 overflows when $Q_{\text{block}}$ is large—WS spill and retries dominate before matmul does. Raising ACC_ROWS (e.g. to 4096) could unlock $Q_{\text{block}}$=256 at seq=512, where Experiment 5 still leaves headroom.

**Critical for honest science:** any claim from a bigger accumulator must be measured with the **baseline re-run on the identical Gemmini config**, not by comparing an enlarged optimized design against an old baseline number. Otherwise reported speedup mixes algorithmic fusion with unequal hardware. The right methodology is paired runs under the same $D_{\text{im}}$, ACC_ROWS, and scratchpad budget—or explicit design-space exploration that states a silicon budget and places both baseline and fused paths on the same Pareto point.

**3. Inter-layer fusion (operator-boundary traffic).** Intra-attention fusion removed the $P$ buffer; the next boundary is **between layers**. Following the "Data Movement Is All You Need" view, traffic spikes at operator boundaries even when each kernel is "fast." A concrete target: after $W_o$, keep the attention sublayer output in scratchpad (or accumulator) and feed the FFN's first projection directly—skipping the DRAM write/read pair that today treats attention output as a standalone tensor. The same accounting extends to residual add paths and later encoder blocks: count bytes moved per layer boundary, then prototype one fusion (analytical traffic table plus a minimal Verilator harness) before claiming full-model savings.

**Other directions (brief). FlashAttention-style blocking on the systolic array:** tile $Q$, $K$, and $V$ so $N \times N$ scores never need a full scratchpad footprint; pairs naturally with online softmax but needs a careful mapping to Gemmini's WS loop and accumulator layout. **Compiler / schedule hints:** annotate fusion-safe regions (e.g. "$P$ never escapes scratchpad") so future Gemmini or LLVM passes can coalesce mvins/mvouts without hand-written RoCC sequences—useful if macro-sequencer hardware is not available.

# 12 Conclusions

We set out to analyze and remove avoidable data movement in Gemmini Transformer attention under a fixed BERT-base workload. The baseline already provides hardware softmax; the dominant fixable cost was materializing $P$ in DRAM between softmax and $PV$.

Experiment 1 showed that scratchpad-local fusion works at $N$=128 but scales catastrophically because built-in full-row softmax and $Q_{\text{block}}$=16 jointly create $O(N^2)$ fine-grain control traffic. Rejecting Normalizer-based online softmax, we built OnlineAttention and integrated it through measured phases, culminating in Experiment 5: **1.63M / 3.27M / 7.43M** cycles at $N \in \{128, 256, 512\}$, **1.12–1.32**× over baseline, **zero** attention-matrix DRAM traffic, and stable scaling where Experiment 1 fails.

The project's architectural contribution is the *bottleneck-shift story*: from DRAM-bound attention buffer, to schedule-bound mvout storms, to matmul- and accumulator-capacity-bound online fusion—with quantitative evidence at each step.

# 13 Statement of Work

Single-person project. **Ivan Lok** performed problem formulation, simulator and Gemmini integration, all benchmark and RTL development, experiment campaigns, analysis documents, presentation, and this report.

# References

# References


[1] K. Hegde et al., "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," DAC, 2021.

[2] T. Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS, 2022.

[3] A. Amid et al., "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," IEEE Design & Test, 2020.

[4] S. Kim et al., "I-BERT: Integer-only BERT Inference," HPCA, 2021.


# A Reproducibility (brief)

All cycle counts use Verilator with `IBertGemminiRocketConfig` and `VERILATOR_THREADS=16`.

The course submission bundle reproduces experiments from `code/` (no simulation logs shipped):

- Apply `code/patches/gemmini_hw_all_changes.patch` and `gemmini_sw_all_changes.patch`, or copy `code/hardware/` and `code/software/` into a Chipyard tree (see `code/patches/README.md` and `COMMITS.txt`).
- Table 7 totals: `code/scripts/exp5/run_exp5_full_seq{128,256,512}.sh`; baseline and Experiment 1: `code/scripts/exp1/`.
- Benchmark–script mapping and the reported cycle table: `code/README.md`.

Patches satisfy the course `git diff` requirement for modified Gemmini repositories.