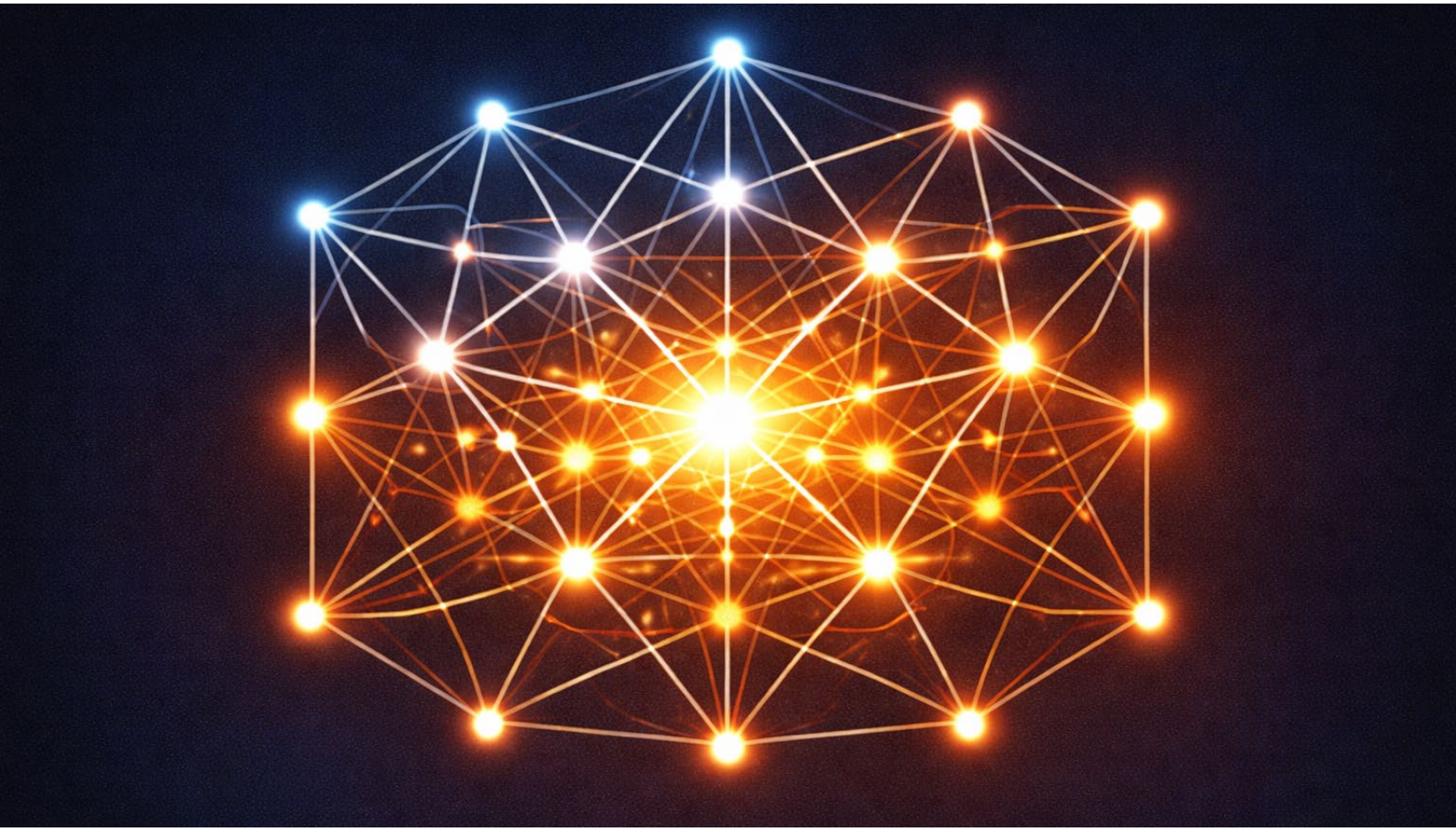
GSI TECHNOLOGY

Matrix Multiplication and Efficiency of Multimodal LLM with Gemini®-II



Introduction

Associative computing applies operations to data based on content rather than address, enabling massive parallelism across memory arrays. Compute-in-memory (CIM) extends this concept by performing computation directly in or on memory structures, minimizing data movement.

This whitepaper discusses how GSI Technology's Associative Processing Unit (APU) performs matrix multiplication and then shows why it is a superior solution to achieve TTFT in multimodal LLM with Text and Video inputs that are useful for edge physical AI awareness.

Figure 1 shows a partial representation of the bit-processor structure used in GSI Technology's CIM chip, Gemini-II. This implements a highly optimal RISC Boolean processor due to its ability to read multiple memory bit cells onto a single memory line. Multiple word lines could turn on to generate Boolean functions on the bit line that would create a collision and garbage on a regular memory and is not an allowed operation on general purpose memory. The computation core array and memory cell are expounded upon in greater detail in patent US10877731B1.

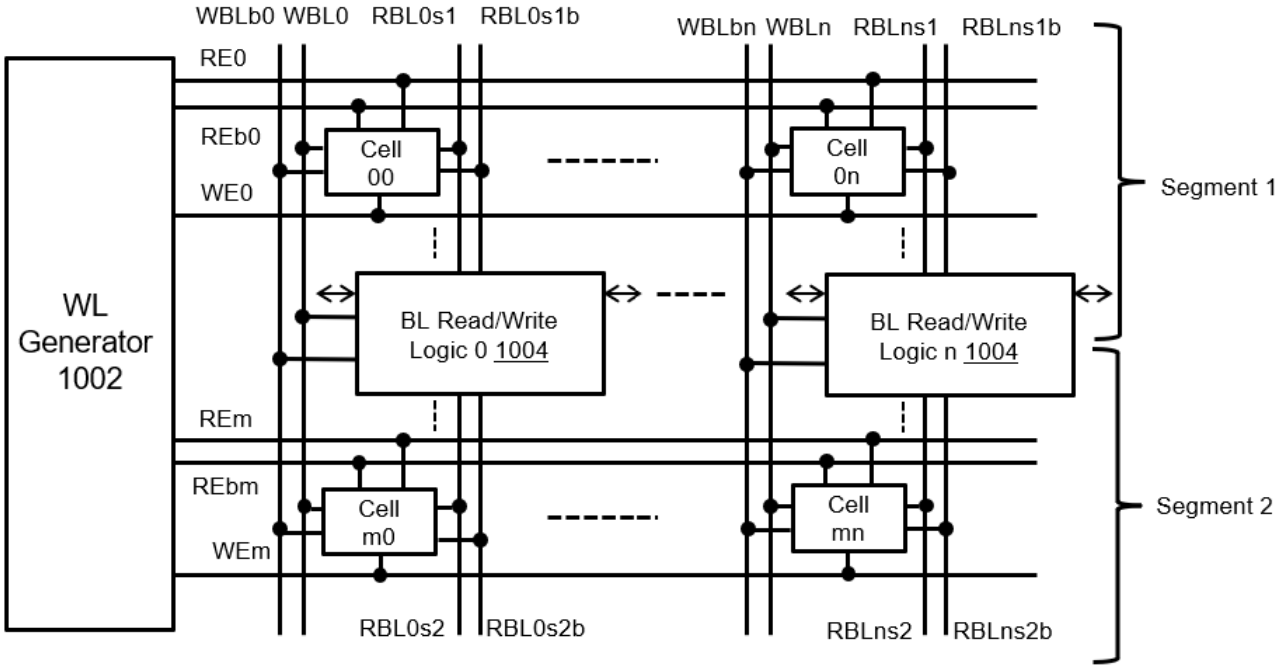


Figure 1: Bit-Processor Structure for Gemini-II

Addition and Accumulation

Figure 2 shows an illustration of a dual port SRAM cell capable of performing a XOR Boolean operation and a multi-bit full adder using the XOR capability. These are expounded upon in greater detail in patent US11604850B2, particularly their use in creating a hardware in-memory full adder.

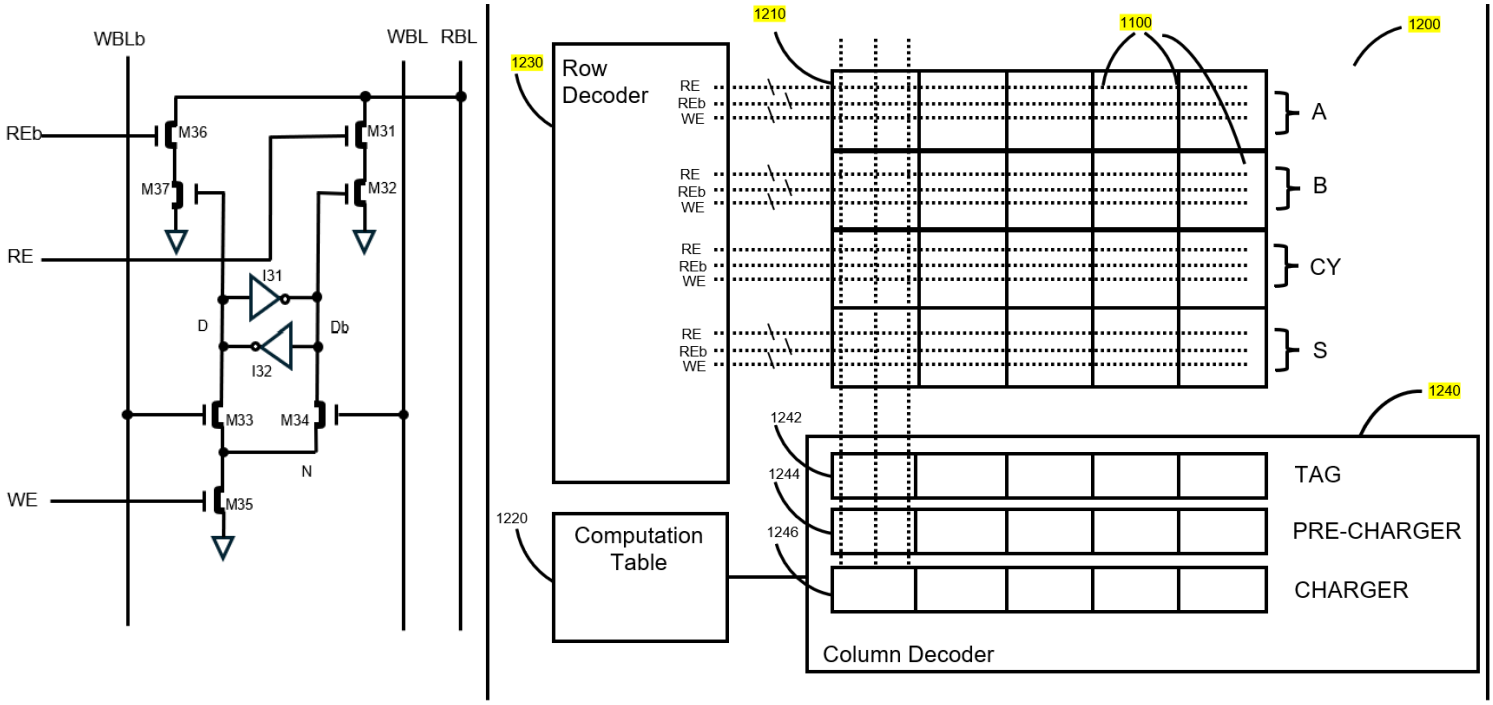


Figure 2: Dual Port SRAM Cell

Ultimate Framework Flexibility

A key property of bit-line CIM is **bit-slice flexibility**:

- Values are represented as collections of bit-planes
- Precision is controlled by how many bit-planes are processed
- 1-bit, INT4, INT8, or higher precision (e.g., INT2048) are achieved by iterating over bit-planes and accumulating results

GSI emphasizes **1-bit granularity** and user-defined bit frameworks, enabling dynamic precision tradeoffs between accuracy, performance, and energy. To provide an example of the flexibility though, it is noted that a customer in the medical research industry used this architecture to implement a 2048-bit “word” for use in molecular representation.

MAC Implementation on Bit-Lines

Let's start with multiplication.

Any multi-bit multiplication can be decomposed into **bitwise partial products**:

$$a \times b = \sum_{i,j} (a_i \wedge b_j)\, 2^{i+j}$$

Thus, multiplication reduces to:

1. Bitwise Boolean operations (AND)
2. Shifts
3. Addition and accumulation

Digital CIM arrays are well-suited to step (1) at massive parallelism and can implement steps (2)–(3) using in-memory adders.

A multiply-accumulate (MAC) operation computes:

$$\text{acc} \leftarrow \text{acc} + (a \times b)$$

This definition is **implementation-independent**. A MAC may be:

- Bit-parallel or bit-serial
- Single-cycle or multi-cycle
- Implemented in a CPU, GPU, DSP, ASIC, or **inside memory arrays**

Let's look at a as an n-bit unsigned integer, b as an m-bit unsigned integer, and acc as a k-bit unsigned integer accumulator.

Each Bit-line Processor (BP) in the Gemini-II APU contains local memory and a 1-bit full adder. The full adder can perform one full-adder operation per cycle.

Multiplying an n-bit number by an m-bit number requires n × m full-adder operations. Adding the result to the k-bit accumulator acc requires k additional full-adder operations. Therefore, a single MAC requires n × m + k full-adder operations when all values are stored in BP memory.

All arithmetic described here assumes unsigned integers.

The Gemini-II APU contains 1 million (2^{20}) Bit-line Processors. Each BP performs one full-adder operation per cycle, with a cycle time of 0.83 nanoseconds (1.2GHz clock).

Single MAC Performance

n (*a* bits)	m (*b* bits)	k = n + m	Cycles	Gemini-II TOPS
4	2	6	14	180
8	2	10	26	97
8	4	12	44	57

We can also do a sequence of MAC operations.

In matrix multiplication:

$$C_{m,n} = \sum_{k} A_{m,k} \times B_{k,n}$$

tiles of matrices A and B are loaded into the APU memory. Each BP computes one output element of matrix C as a dot product, which consists of a sequence of MAC operations.

For each MAC in the sequence, a new value of A is broadcast to the Bit-line Processors.

When multiple MAC operations are performed, the accumulator must be wider to avoid overflow. For a sequence of 64 MAC operations, the accumulator size is k = n + m + 6 bits.

Broadcasting the value of A to the Bit-line Processors takes n + 5 cycles per MAC.

Performance for 64 MAC Operations

n (A bits)	m (B bits)	k (Accumulator bits)	Cycles for 64 MACs	Gemini-II TOPS
4	2	12	1856	87
8	2	16	2880	56
8	4	18	4032	40

Comparing to Edge GPU Implementation for TTFT

The Jetson Orin series uses Ampere GPU with varying SMs, CUDA cores, Tensor Cores, and shared memory/L1, L2 and DRAM. The Jetson Thor uses a Blackwell GPU with ~20 SMs, 2560 CUDA cores, 96 Tensor Cores, ~228 KB shared memory per SM, and ~32 MB L2.

Tensor cores operate on matrix fragments in registers, which must be loaded from shared memory. General matrix-to-matrix multiplication, GEMM, kernels rely on tiling into SMEM, asynchronously copies from generalized memory management to shared memory (GMEM←→ SMEM), and reuse. For large GEMMs, performance often becomes memory-/data-movement-bound, not Tensor Core FLOP-bound.

Thus, Tensor Cores raise compute throughput, but do not change the requirements to tile A/B and repeatedly move tiles between DRAM, L2, SMEM, and registers when matrices are larger than SMEM/L2.

Consider a single large GEMM on the TTFT critical path (e.g., in Gemma-3 12B + SigLIP):

- Context length: 1024 tokens.
- Hidden dimension: 4096.
- Output dimension: 16,384.

Matrices:

- $A \in \mathbb{R}^{1024\times4096}$ (1024 × 4K).
- $B \in \mathbb{R}^{4096\times16384}$ (4K × 16K).
- $C \in \mathbb{R}^{1024\times16384}$ (1024 × 16K).

Assuming float32 (4 bytes):

- $|A| = 4{,}194{,}304$ elements ≈ 16 MB.
- $|B| = 67{,}108{,}864$ elements ≈ 268 MB.
- $|C| = 16{,}777{,}216$ elements ≈ 64 MB.

One token (row of A) = 4096 × 4 = 16 KB.

In Gemini-II compute-in-memory capacity ≈ 96 MB.

Max rows of A fitting on-chip:

- 96 MB/16 KB =~ 6144 rows

For a 1024-token context:

- Entire A fits in compute memory with >5× headroom.
- A can be loaded once and reused in-place.

Execution on Gemini-II:

1. Load A once from DRAM → compute memory: ≈16 MB reads.
2. Stream B once: ≈268 MB reads.
3. Perform MACs in-memory using bitwise MAC pipeline.
4. Write C once back: ≈64 MB writes.

TTFT is dominated by one pass over A/B and one C write.

On Orin (Ampere):

- SM shared memory and registers are much smaller than A/B → must tile.

Using typical Tensor Core GEMM tiling (per NVIDIA docs):

- Thread block tile: $T_M=64,\ T_N=128,\ T_K=128$
- Tiles along M: $\lceil 1024/64 \rceil = 16$.
- Tiles along N: $16384/128 = 128$.
- K-blocks: $4096/128 = 32$.
- Total C tiles: 16 × 128 = 2048.

Each C tile:

- For each K-block, loads:
    - A sub-tile (64×128) and B sub-tile (128×128) from DRAM/L2 into SMEM.
- Uses Tensor Core mma instructions on fragments in registers.
- Writes the C tile once.

Effective DRAM read factors for large GEMMs remain approximately:

- A: ≈32×.
- B: ≈16×.
- C: 0–1× reads, 1× writes.

On Thor:

- Larger shared memory (~228 KB/SM) and ~32 MB L2 help reuse larger tiles.
- Tensor Memory Accelerator (TMA) optimizes GMEM↔SMEM transfers.

Yet, as CUTLASS Blackwell docs and microbenchmarks show:

- GEMM still uses tiling into SMEM and cluster-level shared memory.
- Matrices much larger than SMEM/L2 still require multiple DRAM passes.

Thor lowers effective DRAM reads per element vs Orin Nano/AGX but does not reach Gemini-II's ~1× read.

TTFT Summary

Platform	A DRAM Reads	B DRAM Reads	C DRAM Reads	C DRAM Writes	Structural Behavior
Gemini-2	≈ 16 MB (1×)	≈ 268 MB (1×)	0–64 MB	≈ 64 MB (1×)	Full A resident in compute memory; in-memory MAC; single-pass A/B.
Orin Nano	≈ 0.54 GB (~32×)	≈ 4.29 GB (~16×)	0–64 MB	≈ 64 MB	Tensor Core GEMM with SMEM tiling; multiple DRAM passes over A/B.
AGX Orin	≈ 0.54 GB (~32×)	≈ 4.29 GB (~16×)	0–64 MB	≈ 64 MB	More SMs; same tiling class; DRAM still dominates.
AGX Thor	< Orin, still >>16 MB	< Orin, still >>268 MB	0–64 MB	≈ 64 MB	Larger SMEM/L2, TMA; reduced but still multi-pass over A/B.

Summary and Relevance to LLM Workloads (e.g., Gemma-3, 12B)

Transformer inference relies heavily on matrix multiplication in:

- Q/K/V projections
- Attention score computation (QK^T)
- Attention output
- MLP layers

This paper showed efficiencies available in the GSI CIM Associative Processing Unit when implementing arithmetic logical operations. The combination of dynamic frameworks, and efficient data sequencing available through flexible bit-slice arithmetic logic implementation provides multi-modal vision, text, and sensor LLM processing at latencies that make awareness for edge physical AI. For example, running full Gemma-3, 12B with SigLip allows a streaming analysis whereby only the first tokens are analyzed for environmental awareness. This was shown to have reduced external DRAM transfers, which are the limiting factor for LLMs. This was shown on a recent press release[7] on a Gemini-II processor such a workload results in 3 second response times (e.g., person performing suspicious activity, city infrastructure damaged, etc.) at 30 watts, which is usable for edge awareness applications such as smart city pole cameras, autonomous mobile robots, or drones.

References (Complete)

1. GSI CIM Whitepaper (D0069)
   https://gsitechnology.com/wp-content/uploads/sites/default/files/Whitepapers/D0069-GSIT-Compute-in-Memory-Application.pdf
2. US 10,877,731 B1 – Processing array device that performs one cycle full adder operation and bit line read/write logic features
3. US 12,423,112 B2 – Pipeline Architecture for Bitwise Multiplier-Accumulator (MAC)
4. US 11,604,850 B2 – In-Memory Full Adder
5. US 10,846,365 B2 – Sparse Matrix Multiplication in Associative Memory Device
6. MICRO'25 Paper (Cornell)
   https://www.csl.cornell.edu/~zhiruz/pdfs/apu-micro2025.pdf
7. GSI HPC Overview (1-bit granularity)
   https://gsitechnology.com/hpc-overview/
8. Gemma-3 12B TTFT Press-Release

   https://ir.gsitechnology.com/news-releases/news-release-details/gsi-technology-reports-3-second-time-first-token-edge-multimodal
9. Jetson AGX Orin Series Technical Brief:
   https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf
10. Jetson Orin NX Data Sheet:
    https://developer.nvidia.com/downloads/jetson-orin-nx-series-data-sheet
    https://developer.download.nvidia.com/assets/embedded/secure/jetson/orin_nx/docs/Jetson_Orin_NX_DS-10712-001_v0.5.pdf
11. Jetson Orin overview:
    https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
12. Jetson Thor product page:
    https://www.nvidia.com/en-eu/autonomous-machines/embedded-systems/jetson-thor/

13. Jetson Thor Series Modules Data Sheet DS-11945-001:
https://developer.nvidia.com/downloads/assets/embedded/secure/jetson/thor/docs/jetson-thor-series-modules-datasheet_ds-11945-001.pdf

14. Jetson Thor Series Modules Design Guide:
https://developer.nvidia.com/downloads/assets/embedded/secure/jetson/thor/docs/jetson_thor_series_modules_designguide.pdf

15. Blackwell GPU overview (Thor):
https://developer.ridgerun.com/wiki/index.php/NVIDIA_Jetson_AGX_Thor/Blackwell_GPU

16. NVIDIA blog:
https://developer.nvidia.com/blog/introducing-nvidia-jetson-thor-the-ultimate-platform-for-physical-ai/

17. Ampere architecture whitepaper (A100 Tensor Core GPU):
https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

18. DL performance GEMM guide:
https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

19. PTX ISA (mma instructions):
https://docs.nvidia.com/cuda/parallel-thread-execution/

20. High-performance matmul analysis:
https://aleksagordic.com/blog/matmul

21. Blackwell GEMM & Tensor Memory Accelerator (CUTLASS):
https://research.colfax-intl.com/cutlass-tutorial-writing-gemm-kernels-using-tensor-memory-for-nvidia-blackwell-gpus/
https://research.colfax-intl.com/cutlass-tutorial-gemm-with-thread-block-clusters-on-nvidia-blackwell-gpus/
https://docs.nvidia.com/cutlass/4.3.4/media/docs/cpp/blackwell_functionality.html