perplexity

Summarize the article

Here's a summary of the article "MTIA: First Generation Silicon Targeting Meta's Recommendation Systems" (ISCA'23):

Purpose and Motivation:

- Meta traditionally used CPU-based servers for inference workloads like Deep Learning Recommendation Models (DLRM). Rising compute and memory requirements drove the shift to specialized hardware like GPUs and accelerators.
- GPUs, while powerful, aren't ideal for inference workloads in terms of cost, efficiency, and power consumption.
- To address these issues, Meta developed MTIA, its first in-house inference accelerator chip, specifically targeting recommendation systems.

MTIA Architecture:

- MTIA is designed as an array of Processing Elements (PEs) connected in a grid, with each PE having on-chip memory and specialized compute units.
- Each PE includes:
  - Two customized RISC-V processor cores (one scalar, one vector) for flexibility and parallelism.
  - Fixed function units for memory layout, dot-products, reductions, SIMD elementwise ops, and communication.
  - Local memory and circular buffers for efficient data reuse and simplified producer-consumer synchronization.
- The interconnect includes AXI-based NoC with multicast features and a specialized reduction network for efficient partial-sum accumulation across PEs.

Programming Model and Software Stack:

- The MTIA software stack is built around PyTorch, with integration enabling both eager execution and graph compilation.
- Model-level optimizations focus on Tensor placement in on-chip memory and workload partitioning across accelerator cards or sub-grids.
- Custom compilers and DSLs are used for kernel development, targeting efficient use of hardware features.

Performance Evaluation:

- MTIA achieves near parity in performance-per-watt (perf/W) with GPUs for DLRM workloads and exceeds legacy NNPI accelerators.
- GEMM operations (fully connected layers) on MTIA reach up to 2x perf/W compared to GPU.
- For small batch sizes and operator-dominated models, MTIA outperforms GPU in efficiency. For larger, more complex models, optimizations in the software stack can further improve performance.

Lessons Learned and Future Directions:

- Dual-core PEs enable decoupling data movement from compute and facilitate parallel execution, but require careful software synchronization.
- Hardware features enabling parallelism (instruction/data/thread/memory level), multicasting, and specialized memory architectures are key to efficiency for recommendation models.
- There is significant room for further software and architecture optimizations, especially regarding data locality and operator fusion for next-generation designs.

Significance:

- MTIA demonstrates that a custom silicon solution, closely tailored to target workloads, can deliver substantial cost, efficiency, and programmability gains for large-scale recommendation systems in production data centers.[1]

If you need details on any specific section (e.g., microarchitecture or software stack), let me know!

⁂

what was the SOTA model they evaluated in the paper? back then

The **state-of-the-art (SOTA) model** evaluated in the paper — at that time — was the **Deep Learning Recommendation Model (DLRM)**, which was Meta's main production recommendation system workload and widely adopted in industry. The authors specifically report running benchmarks and comparisons using a set of DLRM models with varying complexities, shown in their Table IV:

- Low Complexity 1 (LC1)
- Low Complexity 2 (LC2)
- Medium Complexity 1 (MC1)
- Medium Complexity 2 (MC2)
- High Complexity (HC)

All these were variants of DLRM with different sizes and FLOPS/batch, ranging from basic to heavily-optimized versions that reflected Meta's production models and the industry's reference standard for recommendation system inference workloads at the time. DLRM itself was the

SOTA reference for large-scale recommender inference in both industry and research circles when the chip was evaluated.

They did not evaluate transformer-based recommenders or other emerging architectures as mainstream SOTA for that particular workload back then—the focus was primarily on DLRM and its optimized derivatives.[2]

⁂

why did they choose RISC-V

Meta chose **RISC-V** for the processor cores within each MTIA Processing Element (PE) for several reasons as described in the paper:

- **Customizability:** RISC-V's open ISA enabled Meta to heavily customize the cores with custom interfaces, registers, instructions, and exceptions. These tailor-made features allowed tight integration with MTIA's fixed-function accelerators, enabling efficient command issuance and data movement.
- **Programmability:** The addition of general-purpose compute (using RISC-V) extended programmability, supporting workloads and operators that didn't map well to fixed function units. The inclusion of RISC-V vector extensions gave the architecture flexibility for a wider range of neural network operations.
- **Open Ecosystem & Tooling:** By leveraging the open-source RISC-V community and existing vendor support, Meta benefited from well-supported toolchains (such as LLVM for codegen) and reduced time-to-market for firmware and driver development.
- **Future-Proofing:** The modular and extensible nature of RISC-V allowed for evolving the architecture to match future workloads or requirements.
- **Integration:** Using RISC-V made it easier to connect with Meta's in-house compiler toolchain, facilitating optimizations like register allocation, inlining, and seamless PyTorch integration.

In summary, RISC-V was selected for its **customizability, extensibility, open toolchain support, and the ability to tightly couple with fixed function units on MTIA**, making it the optimal choice for a rapidly evolving inference accelerator targeting recommendation systems.[3]

⁂

how is MTIA different from a GPGPU?

MTIA differs from a **GPGPU** (General Purpose GPU) in several fundamental ways, as detailed in the paper:

1. Architectural Focus:

- **MTIA** is custom-designed for inference workloads, especially deep learning recommendation models (DLRM), emphasizing low-latency, efficient compute, and tight

memory/data movement for neural network inference.

- **GPGPUs** are general-purpose, highly-parallel processors originally built for graphics and later retrofitted for deep learning. They excel at large, regular compute kernels (matrix multiplies, convolutions), but are not optimized for sparse, irregular, low-batch inference typical of recommendations.

2. Processing Elements:

- **MTIA:** Uses an array of custom PEs, each with two heavily-customized RISC-V cores (scalar + vector) tightly integrated with fixed-function units (Dot-Product, Reduction, SIMD, Memory Layout) and significant local memory. These are specialized for DLRM-style compute and data reuse patterns.
- **GPU:** Uses many general-purpose SIMD cores (CUDA cores) designed to execute regular, vectorized workloads in parallel. Memory hierarchy is built for high throughput, but can be less efficient for sparse or low-occupancy tasks.

3. Memory System:

- **MTIA:** Offers abundant on-chip SRAM (scratchpad), large local memories per PE, circular buffer abstractions, and a memory subsystem/NoC tuned for broadcast, multicast, and reduction operations—ideal for recommender workloads.
- **GPU:** Relies on high-bandwidth HBM or GDDR memory and cache hierarchies, primarily optimized for throughput instead of low-latency, sparse access.

4. Data Movement & Parallelism:

- **MTIA:** Enhances data movement efficiency via multicast on the NoC, specialized reduction networks, atomic primitives, and software-hardware managed producer-consumer execution (using circular buffers and synchronization).
- **GPU:** Generic NoC/interconnect, limited hardware primitives for the specific reductions and synchronization seen in recommendation inference.

5. Programmability & Software Stack:

- **MTIA:** Python/PyTorch-first software stack, with custom graph compilers, operator fusion, and hardware intrinsics designed directly for recommendations.
- **GPU:** Strong software ecosystem, but relies on more general operator fusion/scheduling, less aware of DLRM inference specifics.

6. Perf/Watt & Efficiency:

- **MTIA:** Achieves higher efficiency (perf/W) for sparse and small-batch inference due to its specialization, while remaining programmable for evolving workloads.
- **GPU:** Usually delivers more raw FLOPS, but less perf/W for MTIA's target workloads due to architectural generality and overheads for non-dense or non-large-batch inference.

Summary Table:

Feature	MTIA	GPGPU
Target Workload	DLRM, inference	Graphics, ML (general)
Cores/PEs	Custom PEs, RISC-V + FFUs	SIMD/CUDA cores
Memory System	On-chip SRAM, CBs, multicasts	GDDR/HBM, cache hierarchy
Data Movement	Multicast, reduction net	Generic NoC
Software Stack	PyTorch-first, DLRM-tuned	CUDA, ROCm
Efficiency (perf/W)	Higher (for DLRM)	High (for dense)
Flexibility	Specialization + extensions	General-purpose

MTIA is fundamentally a **domain-specific accelerator** for inference with architectural specialization for recommendation models, while GPGPU is a general-purpose high-throughput engine best for large, regular, and dense workloads. MTIA's specialization drives its memory, compute, and software stack advantages for its target use case.[4]

⁂

what is the compiler stack of MTIA

The **compiler/software stack of MTIA** is designed to tightly couple with PyTorch and optimize for recommendation system inference workloads. Here is how it is structured:

- **PyTorch Integration:** MTIA is deeply integrated as a backend for PyTorch, supporting both eager execution and full graph compilation.
- **PyTorch FX-based Model Compiler:** Transforms the PyTorch graph (represented as FX IR) and progressively lowers it to LLVM IR. This stage applies model-level optimizations (tensor placement, workload partitioning, operator fusion) and sub-graph splitting for multi-card and multi-subgrid execution.
- **Domain-Specific Language (DSL) Compiler (KNYFE):** For rapid ML kernel development, a DSL compiler generates low-level optimized C++ code for MTIA using hardware-specific intrinsics. This helps build efficient custom kernels for critical operators.
- **LLVM Toolchain:** Converts LLVM IR into device executables. The LLVM stack is used due to its support for RISC-V and provides low-level optimizations such as register allocation, inlining, and hardware-specific codegen.
- **MTIA Kernel Operator Library:** Contains ML kernels for the most common operators (e.g., FC layers, EmbeddingBag). Some are hand-coded for maximum performance using exposed device-specific intrinsics.
- **Streaming APIs & Driver:** A CUDA-like streaming API supports scheduling, device memory allocation, and PyTorch operator launches.
- **Firmware:** Device firmware handles PE control, job scheduling, and resource management, based on the Zephyr RTOS.

Graphical stack overview (from the paper):

```
Application Layer (ML Serving Platform)
       |
PyTorch Framework & Runtime (Host)
       |
PyTorch FX MTIA Compiler --> KNYFE DSL Compiler --> LLVM Compiler
       |
MTIA Operator Library
       |
MTIA Streaming API (Host Driver)
       |
MTIA Firmware (Zephyr RTOS, Device Management)
```

Every layer is designed for **production efficiency and developer usability**, making model deployment on MTIA as seamless as possible for Meta's recommendation workloads, while leveraging well-supported open-source compiler infrastructure (LLVM, PyTorch) for flexibility and fast iteration.[5]

⁂

do they have specified matmal FU?

Yes, MTIA features a **dedicated fixed-function unit for efficient matrix multiplications**, which they call the **Dot-Product Engine (DPE)**. This unit is located within each Processing Element (PE) and handles the core computational bottleneck for deep learning inference—primarily GEMM and fully connected layers, which are central to recommendation models.

Key details from the paper:

- The **DPE** can perform large-scale dot-product operations between two input tensors and is optimized for INT8 and FP16/BF16 data types:
  - **INT8:** 1024 multiplications (32×32) per cycle
  - **FP16/BF16:** 512 multiplications (32×16) per cycle
  - Operations are fully pipelined, so multiplying two maximum-size matrices takes 32 cycles
  - Inputs can come from PE local memory or the previous pipeline stage
- The DPE outputs to dedicated reduction accumulators, enabling efficient accumulation of partial sums, broadcasting, and final result storage—tailored for matrix-multiply workloads.

This matrix-multiply (GEMM/FC) specialization is critical for MTIA's efficiency and performance with recommendation models, distinguishing it from GPUs, which typically rely on general-purpose ALUs and SIMT cores for matrix operations.[6]

⁂

compared to TPU it looks closer to a GPGPU than a work specific accelerator like TPU

Your intuition is correct: **MTIA sits somewhere between a classic domain-specific accelerator like TPU and a general-purpose GPU (GPGPU)** in terms of architectural philosophy.

How MTIA is "closer to a GPGPU" than TPU:

- MTIA includes **programmable RISC-V cores** (scalar + vector) in each PE, giving it broader programmability for ML workloads—not just fixed-function MLP/tensor operations (like ACU in TPU).
- It has general-purpose SIMD, Command Processor scheduling, and a flexible software stack (hosted on PyTorch with eager and graph modes). This is fundamentally unlike the "pure kernel-as-hardware" style of early TPUs, which are almost exclusively tailored to matrix multiply via systolic arrays and custom memory/control.
- MTIA supports multiple data types (INT8, FP16, BF16), operator fusion, and custom kernel development through compilers, blending flexibility with specialization.
- The memory hierarchy and interconnect (on-chip SRAM, NoC, multicast, reduction networks) are a hybrid—optimized for DLRM patterns, but not fixed exclusively for dense matrix-multiply (like TPU's systolic array).
- There's explicit focus on **developer productivity and software flexibility**, leveraging RISC-V and PyTorch, akin to the CUDA/GPGPU experience rather than "hardware-first, software-last" of early TPUs.

How MTIA is "domain-specific" (like TPU):

- Fixed-function units for dot-product, reduction, and memory layout are architected for recommendation models and deep learning inference workloads.
- On-chip features (large scratchpad, circular buffers, reduction logic) are geared toward sparsity, small batch, and producer-consumer model pipelines.
- The compiler stack is tailored for DLRM optimization, and the accelerator grid enables data-parallel scaling, reduction, and combination patterns that cohere closely to Meta's core workloads.

TPU Comparison:

- TPUs (especially earlier generations) center on systolic arrays for dense GEMM, fixed control logic, minimal general-purpose programmability, and architectures dictated by training workloads and kernel fusion—very "work-specific."
- MTIA *borrows* some of this specialization but gives up "fully fixed function" in favor of more programmable execution enabled by RISC-V, flexible functional units, and a grid of PEs with local control.

Summary Table:

Feature	MTIA	TPU	GPGPU
Programmable Cores	Yes (RISC-V)	Minimal (scalar)	Yes (SIMT)
Fixed-function Units	Yes	Yes (systolic array)	No (mostly ALUs/FPU)
Target Workload	DLRM/inference	Dense matmul/training	General ML/compute
Flexibility	Moderate	Low	High
Software Stack	PyTorch-integrated	XLA/JAX + libraries	CUDA/Rocm

In conclusion: MTIA is a **hybrid**—more programmable and developer-oriented than classical TPUs, with domain specialization for inference efficiency, but incorporating key GPGPU-like programmability and architectural flexibility.[7]

⁂

compared to GPU tensor cores

Here's a technical comparison of **MTIA** and **GPU tensor cores** for matrix multiplication and recommendation model inference:

MTIA Dot-Product Engine (DPE):

- Dedicated fixed-function block in each PE for dot-products (the main operation in matrix multiplication).
- Capable of 1024 INT8 multiplications (32×32) or 512 FP16/BF16 multiplications (32×16) per cycle, fully pipelined.
- Data types supported: INT8, FP16/BF16; outputs in INT32/FP32 respectively.
- Optimized for the smaller, irregular, and latency-sensitive matmuls typical in DLRM inference, not only for large dense GEMMs.
- Works in concert with local accumulators (Reduction Engine) and circular buffers for producer-consumer patterns and efficient data movement.
- Data often comes from on-chip SRAM/local memory, enabling ultra-low-latency access if tensor placement is optimal.

GPU Tensor Cores (e.g., NVIDIA Ampere):

- Fixed-function hardware blocks embedded in each SM (Streaming Multiprocessor) dedicated to high-throughput matrix math.
- Typically support 4×4, 16×16, or even larger tiles at once (Ampere tensor cores: up to 8×8 INT8 and 16×16 FP16 per core per clock).
- Massive aggregate throughput: e.g., A100 can sustain 312 TFLOPS FP16, with high parallelism across many cores.
- Optimized for training workloads (deep/dense GEMMs), large batches, and regular compute strips. Excellent for convolution and large FC layers, but less so for sparse/small batch workloads.

- Memory feeds from shared memory, L1 cache, L2 cache, and HBM—high bandwidth but sometimes higher latency for small/irregular jobs.

Key Differences:

Attribute	MTIA DPE	GPU Tensor Core
Data Types	INT8, FP16, BF16	FP16, BF16, INT8, TF32
Tile Size/Op	32×32 (INT8), 32×16 (FP16/BF16)	4×4, 8×8, 16×16, up to 1024×1024 (whole GPU)
Peak Throughput/Card	102.4 INT8 TOPS, 51.2 FP16 TOPS	624 INT8 TOPS, 312 FP16 TOPS (A100)
Data Movement	Optimized for on-chip/local memory accesses	Fed from cache/HBM, shared memory
Workload Optimization	Latency-sensitive, low-batch, DLRM inference	Large batch, dense matmul, training
Execution Model	Custom producer-consumer via circular buffers, multicasting, reduction net	SIMT model, warps, block/thread launches
Programming Model	PyTorch-first, custom kernels, operator fusion	CUDA, TensorRT, cuBLAS, kernel fusion

Summary:
MTIA's DPE is **domain-specific for recommendation inference**: efficient, low-latency, and able to deeply exploit on-chip data placement and parallelism for meta's targeted models. GPU tensor cores are **maximally general, high-throughput engines** for matrix ops, optimized for dense compute workloads and large batch sizes. MTIA sacrifices raw peak throughput for tailored efficiency, making it better for certain classes of inference workloads (small/sparse, latency critical) compared to GPU tensor cores.[8]

⁂

why do they have much lower than TDP than GPU

MTIA's **much lower TDP (thermal design power) compared to a GPU** is an intentional architectural decision, rooted in several factors highlighted in the paper:

- **Domain-Specific Specialization:** MTIA is optimized mainly for inference workloads (especially DLRM), rather than being a general-purpose compute engine like a GPU, which must support training, dense compute, large memory bandwidth, graphics workloads, and high parallelism across many different tasks. This lets MTIA trade off raw peak throughput for tailored, lower-power circuitry.
- **Efficient Data Movement:** MTIA's grid-based architecture and use of on-chip SRAM, large local memories, circular buffers, multicasting, and reduction networks are designed to minimize off-chip memory traffic—a huge source of power consumption in GPUs.
- **Simplified Functional Units:** Instead of vast arrays of powerful SIMT CUDA cores and massive on-chip cache hierarchies, MTIA has a smaller number of highly specialized fixed-

function units (e.g., dot-product engines) in each PE. These are lower-power and more energy efficient for target workloads, leveraging parallelism and locality rather than brute-force high-frequency/high-power execution.

- **Lower Frequency and Smaller Die:** MTIA runs at lower frequency and uses a smaller die (19.3 × 19.1mm, 25W typical TDP), whereas modern GPUs (like NVIDIA A100) run at much higher frequencies and have far larger dies (often >400W TDP per card).
- **LPDDR5 Instead of HBM:** MTIA uses LPDDR5 memory, which has much lower power and bandwidth than the HBM used in GPUs but is sufficient for the inference memory footprints targeted.
- **Single-purpose Design:** Unlike GPUs, which support both compute and graphics (necessitating higher power budgets for peak graphics operations), MTIA omits graphics logic and any unnecessary functionality, further reducing energy requirements.

Summary Table:

Attribute	MTIA	GPU (A100 example)
TDP/Card	~25–35W	~330W
Memory	LPDDR5, 176GB/s	HBM2, 1.5TB/s
Die Size	~373mm²	~826mm²
Frequency	Lower	Higher
Main Target	Inference/DLRM	General ML/graphics
Power Design	Specialized PE grid, efficient dataflow	High parallelism, wide SIMT cores, massive bandwidth

In essence: MTIA achieves lower TDP by focusing only on what is strictly necessary for its target inference workloads—enabling better perf-per-watt and total cost of ownership performance for Meta's use case, while GPUs maintain higher TDP to maximize peak general-purpose throughput for broader ML and graphics workloads.[9]

⁂

what is the most power hungry part of the GPU

The **most power-hungry part of the GPU** is typically the **core compute array** itself—specifically, the **SIMT (Single Instruction, Multiple Threads) cores and specialized units like tensor cores**. These units perform massive amounts of parallel computation at high frequency and require substantial energy for both switching and data movement. Depending on the architecture, tensor cores—when fully utilized for high-throughput matrix multiplications—can draw a significant share of the total chip power, especially during deep learning workloads and training.

Other major contributors include:

- **Memory subsystem** (especially HBM/GDDR controllers and DRAM chips): High-bandwidth memory interfaces are also very power-hungry, sometimes drawing 35+ watts per card just for VRAM.
- **Data movement/interconnect**: Crossbar, NoC, and PCIe interfaces can consume meaningful power during large-scale data transfers.
- **Power regulation and cooling**: VRMs and fans together can use 10–15W or more.

However, the dominant power draw (often over half the TDP) under full deep learning or graphics load comes from the **compute cores (CUDA + tensor cores)** running at high utilization, followed by memory bandwidth consumption during large model execution.[10] [11] [12]

⁂

why are people still using GPUs rather than these accelerators

GPUs are still widely used rather than dedicated inference accelerators like MTIA for several important reasons:

- **Flexibility and Universality:** GPUs are highly general-purpose, capable of running a vast array of deep learning models—including training and inference, for computer vision, NLP, generative AI, scientific workloads, and more. They are not restricted to specific architectures or workloads, making them ideal for both experimentation and production across domains.[31] [32] [33]
- **Mature, Robust Software Stack:** NVIDIA GPUs (and to a lesser extent AMD) have decades of investment and market share in developer tooling (CUDA, cuDNN, TensorRT, PyTorch, TensorFlow, ROCm), making development and deployment much easier and faster. Domain-specific accelerators often require custom toolchains, which can slow developer adoption and model migration.[33] [31]
- **Cloud and Ecosystem Availability:** GPUs are available across every major cloud provider and colocation facility—AWS, Azure, Google Cloud, Oracle, etc.—making them the default for scalable AI deployments. Custom silicon like MTIA, TPU, or AWS Inferentia is typically limited to in-house use at hyperscalers or specific clouds, restricting broader accessibility.[34] [31] [33]
- **Training Capability:** Most domain-specific accelerators (like first-generation MTIA) are inference-only, whereas GPUs are the standard for model training at scale. Research, prototyping, and experimentation all rely on the flexibility of GPU parallelism for training novel architectures.[35] [31]
- **Performance Headroom:** Even if accelerators offer superior perf/W for specific tasks, GPUs often deliver much higher raw throughput, making them better for batch processing, large models, and rapid scaling—critical in academic/research and production settings.[32] [36] [35]
- **Hardware Churn and Risk:** Custom chips are often tied to rapidly evolving workloads—GPUs, being general-purpose, remain useful through many generations and use cases, protecting investment in hardware acquisition.[34]

Summary Table:

Reason	GPU Advantage
Flexibility	Supports wide range of ML/AI workloads
Software Ecosystem	Well-supported frameworks/toolchains
Availability	Universal across clouds/datacenters
Training Capability	Full support for training/inference
Raw Performance	Higher throughput for large batch/model sizes
Longevity/Churn	Less risk of rapid obsolescence

MTIA, TPUs, and other accelerators are increasingly used for cost-efficient, hyperscale inference, but **GPUs remain dominant because of versatility, accessibility, developer ecosystem, and their central role in both research and training**. [31] [32] [33] [35] [34]

⁂

1. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
2. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
3. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
4. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
5. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
6. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
7. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
8. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
9. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
10. https://www.tomshardware.com/features/graphics-card-power-consumption-tested
11. https://www.wevolver.com/article/tensor-cores-vs-cuda-cores
12. https://massedcompute.com/faq-answers/?question=How+do+Tensor+Cores+affect+the+power+consumption+of+large+language+models+during+training%3F
13. https://learn.microsoft.com/en-us/answers/questions/4073781/gpu-crash-(-)
14. https://www.reddit.com/r/hardware/comments/wgqre7/what_is_the_breakdown_of_power_consumption_by/
15. https://acecloud.ai/blog/cuda-cores-vs-tensor-cores/
16. https://hothardware.com/news/amd-predicts-gpus-scaling-to-700w-by-2025
17. https://www.videocardbenchmark.net/power_performance.html
18. https://www.reddit.com/r/nvidia/comments/1hxmvm4/what_exactly_can_cuda_cores_do_that_tensor_cores/
19. https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/
20. https://rafay.co/ai-and-cloud-native-blog/gpu-metrics-power
21. https://stackoverflow.com/questions/47335027/what-is-the-difference-between-cuda-vs-tensor-cores

22. https://chipsandcheese.com/p/analyzing-video-card-efficiency-part-i-power
23. https://www.beam.cloud/blog/cuda-cores-vs-tensor-cores
24. https://www.osti.gov/servlets/purl/1326472
25. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
26. https://www.reddit.com/r/buildapc/comments/1clаxmq/you_guys_know_which_gpu_is_most_powereffici ent/
27. https://gtg.benabraham.net/whats-going-on-with-nvidia-gpus/
28. https://www.tomshardware.com/desktops/servers/a-single-modern-ai-gpu-consumes-up-to-37-mwh-of-power-per-year-gpus-sold-last-year-alone-consume-more-power-than-13-million-households
29. https://news.ycombinator.com/item?id=38775309
30. https://www.forbes.com/sites/bethkindig/2024/06/20/ai-power-consumption-rapidly-becoming-missio n-critical/
31. https://www.cloudoptimo.com/blog/tpu-vs-gpu-what-is-the-difference-in-2025/
32. https://www.liquidweb.com/gpu/vs-ai-accelerators/
33. https://www.gigenet.com/blog/gpu-vs-tpu/
34. https://www.atlantic.net/gpu-server-hosting/ai-accelerator-vs-gpu-5-key-differences-and-how-to-cho ose/
35. https://www.radiofreemobile.com/meta-vs-nvidia-no-competition/
36. https://www.delloro.com/evolution-of-accelerated-computing-for-ai-applications/
37. https://www.reddit.com/r/computers/comments/1ilqotn/can_someone_smarter_than_me_explain_like_im _five/
38. https://encord.com/blog/meta-ai-chip-mtia-explained/
39. https://nwai.co/what-is-the-difference-between-ai-accelerators-and-gpus/
40. https://arxiv.org/html/2511.06565v1
41. https://www.nextplatform.com/2024/04/10/with-mtia-v2-chip-meta-can-do-ai-training-as-well-as-infe rence/
42. https://www.deeplifelearning.com/p/challenges-on-accelerating-ml-inference
43. https://blogs.sw.siemens.com/hardware-assisted-verification/2024/08/22/the-rise-of-custom-accelerati on/
44. https://cacm.acm.org/research/domain-specific-hardware-accelerators/
45. https://www.reuters.com/technology/artificial-intelligence/meta-begins-testing-its-first-in-house-ai-trai ning-chip-2025-03-11/
46. https://www.byteplus.com/en/topic/448374
47. https://blog.purestorage.com/purely-educational/gpus-vs-fpgas-whats-the-difference/
48. https://engineering.fb.com/2025/09/29/data-infrastructure/metas-infrastructure-evolution-and-the-adv ent-of-ai/
49. https://dl.acm.org/doi/pdf/10.1145/3579371.3589348
50. https://www.kearney.com/industry/technology/article/breaking-the-gpu-stronghold-emerging-competiti on-in-ai-infrastructure
51. https://www.ibm.com/think/topics/ai-accelerator-vs-gpu