Meta's Second Generation AI Chip: Model-Chip Co-Design and Productionization Experiences

Joel Coburn[1], Chunqiang Tang[1], Adam Hutchin, Ajit Mathews, Alex Mastro, Amin Firoozshahian[2], Amit Nagpal, Aravind Sukumaran-Rajam, Arushi Sharma, Ashwin Kamath, Ashwin Narasimha, Bhasker Jakka, Brian Dodds, Cao Gao, David Reiss, Deboleena Roy, Eleanor Ozer, Emmanuel Menage, Eran Tal, Erum Kazi, Feixiong Zhang, Guoqiang Jerry Chen, Hangchen Yu, Harikrishna Reddy, Harish Dixit, Indu Kalyanaraman, Jack Montgomery, Jian Huang, Jinghan Yang, Jiyuan Zhang, Jongsoo Park[2], Junhan Hu, Kaustubh Gondkar, Mahesh Maddury, Maxim Naumov, Mike Tsai, Mohammed Sourouri, Neeraj Agrawal, Olivia Wu, Olusiji Medaiyese, Pankaj Kansal, Pavan Shetty, Poorvaja Ramani, Pritesh Modi, Raviteja Chinta, Richard Wareing, Roman Levenstein, Sameer Abu Asal, Saritha Dwarakapuram, Sathish Sekar, Satish Nadathur, Shreya Varshini, Sterling Hughes, Tanmay Zargar, Truls Edvard Stokke, Tyler Graf, Xiaolong Xie, Xun Jiao, and Zitong Zeng
Meta Platforms

Abstract

The rapid growth of AI workloads at Meta has motivated our in-house development of AI chips, aiming to significantly reduce the total cost of ownership and mitigate risks posed by unpredictable GPU supplies. At ISCA'23, we presented Meta's first-generation AI chip, MTIA 1. This paper describes its successor, MTIA 2i, now deployed at scale and serving billions of users. MTIA 2i significantly improves upon MTIA 1, reducing total cost of ownership by 44% compared to GPUs while delivering competitive performance per watt. A key differentiator is its memory hierarchy: instead of costly HBM, it uses large SRAM alongside LPDDR. Although there has been a proliferation of publications on AI chips, they often focus on architectural design and overlook three critical aspects: (1) co-designing and optimizing ML models to work effectively with the AI chip; (2) demonstrating sufficient flexibility to support a wide range of models; and (3) during the productionization process, addressing challenges unanticipated or decisions deferred at design time, such as dealing with memory errors, safe overclocking, reducing provisioned power, and implementing real-time firmware updates to mitigate silicon design defects. A key contribution of this paper is sharing our experience with these aspects, based on our journey of productionizing MTIA 2i at scale.

CCS Concepts

• **Computer systems organization** → **Special purpose systems**; *Parallel architectures*; *Systolic arrays*; • **Hardware** → *Application specific integrated circuits*; *Fault tolerance*; *Power and energy*; • **General and reference** → **Performance**; **Empirical studies**; *Design.*

Keywords

Accelerators, Artificial intelligence, Inference, Machine learning, Memory hierarchy, Productionization, Deep learning recommendation models, Total Cost of Ownership

ISCA '25, June 21–25, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/2025/06
https://doi.org/10.1145/3695053.3731409

1 Introduction

AI workloads have been growing rapidly across the industry. At Meta, our products, such as Facebook, Instagram, and Threads, critically depend on deep learning recommendation models (DLRM) [22] to deliver personalized content to users, including advertisements, short videos, and friend posts. Additionally, Llama-based generative AI powers various product features, such as image and text generation for advertisers [2, 3].

The rapid growth of Meta's AI workloads has motivated our in-house development of AI chips. Previously, we presented Meta's first-generation experimental AI chip, MTIA 1 [10]. This paper describes its successor, MTIA 2i, now deployed at scale and serving billions of users. Among Meta's four major categories of AI workloads—combinations of training or inference for generative AI or recommendation models—MTIA 2i is optimized for inference for recommendation models, which currently represent the majority of inference workloads at Meta.

MTIA 2i has two major goals: (1) significantly lowering total cost of ownership (TCO) compared to GPUs, and (2) offering sufficient flexibility to support a wide range of production models. While ensuring flexibility, our goal is not to stretch MTIA 2i to accommodate every model at Meta, as doing so could move it away from its optimal design point. For models not supported by MTIA 2i, we can still leverage GPUs available on the market.

We have successfully achieved both goals. For the models we have launched into production, MTIA 2i reduces the TCO by an average of 44% compared to GPUs. Additionally, the flexibility of MTIA 2i is evident in its support for Meta's latest models, such as DHEN [30] and HSTU [28]. Developed after MTIA 2i's design freeze, these models exhibit significantly greater complexity than traditional DLRM models [22].

Without delving into details, we summarize several notable aspects of MTIA 2i. First, MTIA 2i features a unique memory hierarchy, incorporating a large amount of SRAM and utilizing LPDDR DRAM instead of HBM. Second, it provides hardware support for PyTorch's eager mode, enabling job launches in less than 1 µs. In contrast, most non-GPU AI chips lack eager mode support and

[1]Contributions: Joel and Chunqiang drafted the paper. The other authors made major contributions to developing the MTIA 2i features highlighted in the paper. MTIA 2i is a large effort, with many aspects not covered in this paper due to space constraints. We are grateful to the hundreds of contributors who made MTIA 2i possible.
[2]This work was done while Amin and Jongsoo were at Meta.

Model Type	Description	Model Size	Model Complexity
Retrieval	Rank initial 1M candidates.	50-100 GB	0.001 - 0.01 GFLOPS/sample
Early stage ranking	Rank 10K candidates.	100-300 GB	0.01 - 0.1 GFLOPS/sample
Late stage ranking	Final ranking of top 100 candidates.	100-300 GB	0.2 - 2 GFLOPS/sample
HSTU retrieval	Rank initial hundreds of Ms candidates.	1TB	10 GFLOPS/request
HSTU ranking	Similar to ranking above.	2TB	80 GFLOPS/request

Table 1: Examples of some production models at Meta. Note that 90% of model size is embeddings.

rely on statically compiled graphs, which limits flexibility. Third, while MTIA 2i was designed as an efficient, low-power, small form-factor accelerator, its system-level performance rivals that of GPU-based systems. Specifically, our MTIA 2i-based production server, equipped with 24 MTIA 2i chips, achieves total performance comparable to that of our GPU-based production server equipped with eight GPUs. Finally, compared to its predecessor, MTIA 1 [10], MTIA 2i delivers more than 3x peak FLOPS, over 3x SRAM bandwidth, greater than 3x network-on-chip bandwidth, 2x DRAM capacity, and approximately 1.4x DRAM bandwidth.

When exploring other AI chips for comparison with MTIA 2i, we observe that the trend of developing in-house AI chips has gained significant momentum among major IT companies, including Google [15], Amazon [11], Huawei [18], IBM [19], Microsoft [1], and Alibaba [14]. Additionally, many startups, such as Cerebras [20], Groq [5], and SambaNova [23], offer their AI chips as alternatives to GPUs from established vendors [8, 16, 24].

Each of these AI chips adopts a design point tailored to its target workloads and makes distinctive architectural choices. For instance, Cerebras and Groq forego DRAM and HBM, relying solely on SRAM, while SambaNova employs reconfigurable dataflow. Rather than focusing the comparison solely on these architectural differences, our experience of productionizing MTIA 2i has taught us that the success of an AI chip crucially depends on several aspects beyond architectural design: (1) co-designing and optimizing ML models to work effectively with the AI chip; (2) demonstrating sufficient flexibility to support a wide range of models; and (3) during the productionization process, addressing challenges unanticipated or decisions deferred at design time, such as dealing with memory errors, safe overclocking, reducing provisioned power, and implementing real-time firmware updates to mitigate silicon design defects.

Despite the proliferation of publications on AI chips [1, 5, 8, 11, 14–20, 23, 24], prior works often focus on architectural design while overlooking these critical aspects. Therefore, a key contribution of this paper is sharing our experience with these aspects, based on our journey of productionizing MTIA 2i at scale.

The remainder of the paper is organized as follows. Section 2 summarizes Meta's recommendation models. Section 3 provides an overview of MTIA 2i. Section 4 discusses our experience co-designing and optimizing models for MTIA 2i. Section 5 shares our experience productionizing MTIA 2i at scale. Section 6 presents a case study on porting one of Meta's critical models to MTIA 2i, and section 7 shows results for MTIA 2i across a range of models in production. Section 8 discusses some of the limitations and challenges with MTIA 2i. Finally, Section 9 concludes the paper.

2 Production Models

Before the rise of Transformer-based models, Deep Learning Recommendation Models (DLRMs) were, and remain, a dominant ML workload at Meta. Their canonical architecture includes embeddings for sparse features (e.g., categorical inputs like post IDs), a multilayer perceptron (MLP), also known as a Fully Connected (FC) network, for dense features (e.g., continuous values like ages), and a final MLP to process interactions between sparse and dense components. Recently, some recommendation models have adopted Transformer-like architectures to capture sequence information, significantly increasing complexity and parameter size. These trends have guided our co-design of models and MTIA 2i. Table 1 summarizes some of our current models. We explain each model below.

Retrieval: Retrieval models, positioned at the front of the recommendation funnel, process millions of candidates to narrow the list to 10K–100K, which are then passed to early-stage ranking. Retrieval models are low complexity and must operate at very large batch sizes to maintain efficiency, and they can spend a significant amount of time on feature preprocessing. Communication between CPU and accelerator and front end network can become bottlenecks. Also, unlike early and late stage models, retrieval requires processing both the user embeddings and ad embeddings on the same host machine, so this puts pressure on host DRAM, device DRAM, or both.

Early-stage ranking: Early-stage models further refine thousands of candidates passed from the retrieval phase, selecting only hundreds for final ranking in the late stage. Some preprocessing of ad embeddings is required but to a much lesser extent than retrieval. Memory bandwidth tends to be the bottleneck due to the low complexity and high batch sizes needed to saturate an accelerator.

Late-stage ranking: Late-stage models rank a few hundred candidates prepared by early-stage ranking and present the final top results to users. Due to the importance of late-stage ranking, efforts to improve model quality have led to increases in model size and complexity. New architectures, such as DHEN [30] and Wukong [29], show continued improvements in model quality with greater computational power, reaching up to 2 GFLOPS per sample. DHEN's hierarchical design enables high-order interactions, converting FLOPS into model quality. Wukong extends DHEN by scaling models across two orders of magnitude. With effective modeling of high-order interactions, more sparse features enabled by larger embedding tables improve model quality. However, significant diversity in complexity and size remains among late-stage ranking models in production, with over 60x variation.

Transformer-like models for recommendation: There is an increasing trend toward incorporating Transformer-like architectures in recommendation models. For instance, HSTU [28], developed at Meta, enhances prediction quality by processing user history in a generative, sequential manner. HSTU employs ragged attention to effectively manage the skewed distribution of user history sequences. Consequently, it requires jagged tensor support and can specialize across varying sequence lengths. Techniques like HSTU are sequence models working with large effective batch sizes introducing a 10x–100x complexity increase per request compared to the most demanding recommendation models. They also tend to have much larger embeddings, which places significant demands on memory capacity and bandwidth.

		MTIA 2i	MTIA 1
Technology		TSMC 5nm	TSMC 7nm
Frequency		1.35GHz	800MHz
Instances		2.35B gates, 103M flops	1.12B gates, 65M flops
Area		25.6mm x 16.4mm	19.3mm x 19.1mm
Package		50mm x 40mm	43mm x 43mm
Voltage		0.85V	Dual rail: 0.67V (logic), 0.75V (memories)
TDP		85W (65W typical)	35W (25W typical)
Host connection		8x PCIe Gen5 (32 GB/s)	8x PCIe Gen4 (16 GB/s)
GEMM TOPS		354 TFLOPS/s (INT8)	102.4 TFLOPS/s (INT8)
		177 TFLOPS/s (FP16/BF16)	51.2 TFLOPS/s (FP16)
GEMM TOPS (Sparsity)		708 TFLOPS/s (INT8)	N/A
		354 TFLOPS/s (FP16/BF16)	N/A
SIMD TOPS	RISC-V vector core	5.5 (INT8), 2.8 (FP16), 1.4 (BF16/FP32)	3.2 (INT8), 1.6 (FP16), 0.8 (FP32)
	SIMD Engine	5.5 (INT8/FP16/BF16/FP32)	3.2 (INT8), 1.6 (FP16)
Memory capacity	Per-PE local memory	384 KB	128 KB
	On-chip SRAM shared across PEs	256 MB	128 MB
	Off-chip LPDDR5	64-128 GB	32-64 GB
Memory bandwidth	Per-PE local memory	1 TB/s	0.4 TB/s
	On-chip SRAM	2.7 TB/s	0.8 TB/s
	Off-chip LPDDR5	204.8 GB/s	176 GB/s

Table 2: Specifications of MTIA 2i versus MTIA 1.

Figure 1: MTIA 2i's overall architecture.

3 MTIA 2i Overview

This section provides an overview of MTIA 2i, focusing on its differences from its predecessor, MTIA 1, while directing readers to the MTIA 1 paper [10] for details on shared components. Additional details on the MTIA 2i architecture can be found in [21].

3.1 MTIA 2i Architecture Overview

Table 2 compares the specifications of MTIA 2i and MTIA 1. While MTIA 2i retains many features of MTIA 1, its design incorporates significant enhancements that triple overall performance, with only a 1.13x increase in silicon die area. Figure 1 shows the overall architecture of MTIA 2i. It consists of an 8×8 array of processing elements (PEs) connected via a custom network-on-chip.

Network-on-chip (NoC): The NoC connects, through crossbars located on each side of the die, to a set of on-chip SRAMs shared by the PEs and to off-chip memory controllers. It delivers 3.3x the bandwidth compared to MTIA 1. It ensures high-speed, high-bandwidth data transfer between the 64 PEs, the host, the Control Core, and the memory subsystem. The NoC features a non-blocking architecture, minimizing interference among different initiators. To handle NoC congestion, flow control is enforced at the sources. Leaky-bucket traffic shaping and packet fragmentation are used to smooth traffic and prevent sudden bursts and congestion.

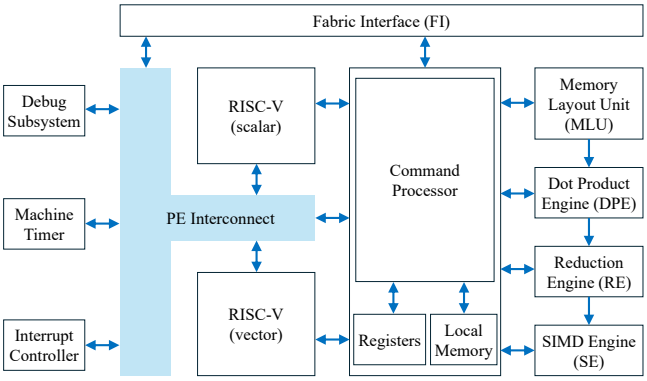

Figure 2: Internal architecture of Processing Element (PE).

Host Interface: Compared to MTIA 1, MTIA 2i incorporates a faster Host Interface, featuring PCIe, DMA, and a secure boot processor. Moreover, MTIA 2i introduces a new host-to-accelerator decompression engine to enhance the effective bandwidth over PCIe.

Control Core: It is a RISC-V quad-core processor coordinating operations across the 64 PEs.

3.2 Processing Element (PE)

Figure 2 shows the internal architecture of a PE. Each PE comprises two RISC-V processor cores and their associated peripherals (on the left), as well as a set of fixed-function units specialized for specific computations or data movements (on the right).

Local Memory: Each PE includes 384 KB of fast Local Memory (LS), representing a 3x increase over MTIA 1.

PE Interconnect: It provides connectivity among the RISC-V cores, peripherals, Local Memory, and custom hardware blocks. Without going through the PE Interconnect, the fixed-function units can directly access data from Local Memory within the PE and can form a coarse-grained pipeline, where data is passed from one unit to the next for successive operations.

RISC-V Cores: Within a PE, the RISC-V cores execute the application's code and issue commands to the Command Processor to offload computations to fixed-function units. One of the RISC-V cores is augmented with the RISC-V vector extension (64B wide), enabling it to execute in a Single-Instruction-Multiple-Data (SIMD) fashion, providing an alternative to using the fixed-function units.

Execution on a PE follows an asynchronous dataflow model. The programmer writes code for the RISC-V cores to generate a series of custom instructions that execute on the fixed-function units in a dataflow manner, with DMA transfers and computations occurring as their dependencies are resolved. The RISC-V cores properly implement the desired kernel to keep the fixed-function

units efficiently utilized.

Memory Layout Unit (MLU): It performs memory-layout transformation such as transpose, concatenate, and reshape.

Dot Product Engine (DPE): It performs General Matrix Multiplication (GEMM) operations, which are often the most intensive computations in kernels. It operates on two tensors. The first tensor is read and cached in the DPE, while the second tensor is streamed from Local Memory to perform the dot product using all the rows of the first tensor. The DPE-to-Local Memory bandwidth is doubled compared to MTIA 1. The DPE incorporates two 32 × 32B × 32 Multiply-Accumulate (MAC) tiles, providing a combined throughput of 2.76 TFLOPS/s per PE with inputs in FP16/BF16 and output in FP32. It also supports 2:4 sparsity for weights, which can deliver an additional 2x throughput.

Reduction Engine (RE): It stores matrix multiplication results as they are accumulated. A dedicated reduction network allows the RE to receive and accumulate results, forward them to the next neighboring PE, or transfer them to the SIMD Engine for further computation, such as applying an activation function.

SIMD Engine (SE): It performs various operations on a vector, including quantization and nonlinear functions. In addition to its floating-point ALUs, which can receive data from the Reduction Engine or read directly from Local Memory, it also contains a set of lookup tables for approximating nonlinear functions. For vector computation, MTIA 2i can utilize either the SIMD Engine or the RISC-V vector extension with 64B vector registers, and the SIMD Engine offers 2x and 4x higher throughput for FP16 and BF16/FP32, respectively. As a result, MTIA 2i improves the FP16/BF16 GEMM to FP32 SIMD ratio compared by MTIA 1 by decreasing it to 32:1, providing more acceleration for non-GEMM operations.

Command Processor (CP): This custom logic orchestrates execution across the fixed-function units, handling dependency checking, scheduling, and tracking of the RISC-V cores' custom instructions. It also arbitrates access to Local Memory between the RISC-V cores and the fixed-function units. The CP provides the programmer with an easy-to-use Circular Buffer (CB) abstraction from Local Memory, while offloading the CB's management and dependency tracking to hardware.

Fabric Interface (FI): It acts as a DMA engine to transfer data in and out of the PE's Local Memory through the network-on-chip (NoC), and to or from on-chip or off-chip memory. The FI-to-NoC bandwidth is doubled compared to MTIA 1.

3.3 New Features in MTIA 2i

Building on our experience with MTIA 1 and projections of future model trends, we introduced several new hardware features in MTIA 2i, as described below.

Dynamic INT8 quantization: MTIA 2i supports dynamic INT8 quantization by leveraging the Reduction Engine to identify the minimum and maximum values per batch, which are then used to derive scaling factors for row-wise quantization. This approach enables channel-wise symmetric dynamic INT8 quantization for Fully Connected (FC) layers, minimizing the model quality gap compared to FCs using FP16. However, effectively using this feature in production has proven to be challenging, which will be discussed further in Section 4.4.

Compression: MTIA 2i supports lossless Asymmetric Numerical System (ANS) compression for weights, achieving up to a 50% compression ratio, thereby reducing the consumed memory space, memory bandwidth, and network-on-chip bandwidth. However, effectively leveraging this feature has been challenging, as our models have not widely adopted INT8 for weights and FP16 data does not compress efficiently. Additionally, MTIA 2i supports GZIP compression at rates up to 25 GB/s for PCIe communication between the host and device, alleviating PCIe and network congestion. This significantly benefits our early-stage retrieval models, which transfer large volumes of data between host and device.

Sparsity: MTIA 2i supports 2:4 weight sparsity in the Dot Product Engine, which could potentially double effective FLOPS. However, our production experience indicates that exploiting sparsity is challenging due to potential quality loss in our recommendation models. To be effective, sparsity must apply to the largest weight matrices, which are often used in the most critical layers that impact model quality. Many of our models lack sufficient sparsity in these matrices, leading to accuracy degradation. Therefore, this feature is not yet widely used in production.

Fast eager mode: To better support PyTorch's eager mode, MTIA 2i provides hardware features that accelerate job launches. Eager mode executes operations immediately as they are called, rather than first compiling them into a static computation graph. We support eager mode for several reasons. First, ML training often requires eager mode, and with MTIA 2i supporting it, we can prototype the software ecosystem for future training chips. Second, many complex models in PyTorch cannot be fully compiled into a static graph, necessitating eager mode support even for ML inference. Third, eager mode accelerates host-bound operations during inference that may not fit into the graph mode, such as merge, remote, or local networks. Fourth, it enables real-time weight updates, improving model freshness.

To support eager mode, MTIA 2i includes several improvements over MTIA 1. The Control Core is upgraded from an ARM CPU with a single core to four cores. Additionally, the Control Core can broadcast Work Queue (WQ) descriptors for eager mode jobs to PEs, and the PEs are equipped with a Work Queue Engine (WQE) to DMA WQ requests from the Control Core. These improvements reduce PE job launch time by as much as 80%, launching jobs in under 1 µs and replacing jobs in less than 0.5 µs.

Addressing bottlenecks in issuing instructions: With the increased GEMM FLOPS and SIMD FLOPS in MTIA 2i, more powerful custom instructions are required to fully utilize the computation engines. This became evident in our initial kernel implementations, which did not utilize these advanced custom instructions and were bottlenecked by the custom-instruction issue rate. Our RISC-V scalar cores could not issue custom instructions fast enough, resulting in lower out-of-the-box efficiency, particularly for smaller GEMM shapes.

For GEMMs operations, we introduced new custom instructions to support multiple contexts, which helps avoid unnecessarily duplicating writes to custom registers. We also added an auto-increment offset feature, which is crucial for issuing matrix multiplication instructions in a tight loop. When activations or weights are too large

to fit in SRAM, we leverage a prefetch feature added to DMA_IN instructions, enabling data to be read directly from DRAM into the SRAM in advance of loading it into the Local Memory. With these enhancements, we can achieve >92% of peak FLOPS for GEMM shapes such as 2K x 2K x 2K.

For sparse operators, particularly Table Batched Embedding (TBE), we needed to increase the rate of DMA reads for embedding rows from tables in memory and improve the rate of row-wise accumulations requested from the SIMD Engine to avoid being instruction-bound. We implemented a new version of our DMA_IN instruction, which takes an index as input and automatically calculates the address. Additionally, we added support for handling unaligned addresses, which was absent in MTIA 1. For accumulations in the SIMD Engine, we introduced a new instruction capable of handling up to 128 rows, compared to the 32 rows previously supported, significantly reducing the number of instructions needed for embedding pooling.

3.4 Server Design

We use the open-source Grand Teton [7] platform as the shared server platform for both our MTIA 2i-based servers and GPU-based servers, which helps reduce costs through platform reuse.

Each MTIA 2i server has two CPU sockets, each connected to a PCIe switch that links to six accelerator modules. Each module houses two MTIA 2i chips, connected via two 8x Gen5 PCIe links. The PCIe bandwidth supports both host-to-device and peer-to-peer (P2P) communication between modules. In total, each server contains two CPUs and 24 MTIA 2i accelerators. Each CPU has 96 cores and is connected to 12 x 96GB DDR5 DRAM (totaling 1.15 TB per CPU), offering 460 GB/s of bandwidth. Additionally, each CPU is equipped with 2 x 200Gbps Ethernet NICs. We treat each CPU socket and its 12 attached MTIA 2i chips as a single logical system. This configuration results in 8 CPU cores, 96 GB of host DRAM with 38 GB/s bandwidth, and 4.17 GB/s Ethernet bandwidth per accelerator. Our container management system [25] allocates accelerators to ML models at the granularity of one or more accelerators, along with the corresponding cores, DRAM, and NIC bandwidth. The scheduling is NUMA-aware, ensuring that sharded models are placed on one or more modules within the same PCIe switch.

Densely packing 24 accelerators into one server helps amortize the cost of non-accelerator components (e.g., CPU, DRAM, NIC, and motherboard), but also introduces challenges. Although CPU cores and PCIe bandwidth are sufficient, host DRAM bandwidth becomes a bottleneck when running low-complexity models on all 24 accelerators at the same time. Beyond resource-allocation optimizations (e.g., NUMA-aware memory allocation, thread core pinning, and transparent huge pages), we eliminate unnecessary memory copies of input tensor data and offload certain cast operations to the accelerator, halving data transfer by converting FP32 to FP16. Additionally, we implement a userspace driver for MTIA 2i, streamlining software releases and improving the predictability of multi-instance serving.

3.5 Software Stack: Evolving with PyTorch

At Meta, the ML software stack is centered around PyTorch, providing consistent support for GPU, CPU, and MTIA to ensure a unified development experience across hardware platforms. Unlike many AI chips that rely on static computation graphs and are incompatible with PyTorch's eager mode, MTIA adopts a PyTorch-first design, offering proper support for eager mode and seamless integration with PyTorch. Figure 3 provides a high-level overview of MTIA's software stack. For further details, see the MTIA 1 paper [10]. Below, we briefly summarize the software stack's evolution since MTIA 1.

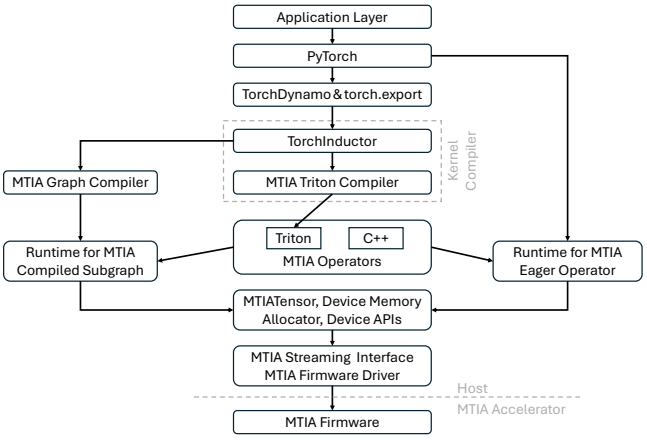

Figure 3: MTIA software stack.

From MTIA 1 to MTIA 2i, the software stack has evolved from PyTorch with the FX compiler to PyTorch 2.0 [6] with TorchDynamo, TorchInductor, and Triton [26]. TorchDynamo enables symbolic tracing to capture models with dynamic shapes, while TorchInductor generates Triton code for PyTorch operators and automatically detects operator patterns for fused operator generation. Triton offers a uniform framework for writing custom kernels across MTIA, GPU, and CPU. We have optimized Triton kernel compilation to efficiently utilize MTIA's fixed-function units. Additionally, we added eager mode support via the PyTorch runtime, which has proven invaluable in both training and inference, as well as for debugging.

3.6 Discussion: Unique Memory Hierarchy

Compared to GPUs and other accelerators, the memory hierarchy of MTIA 2i is unconventional. It uses a large SRAM (256 MB) backed by LPDDR DRAM, avoiding HBM to reduce cost and power consumption. The larger SRAM is chosen to meet the stringent latency requirements of our recommendation models. Moreover, unlike large Transformer models [27], these relatively smaller models exhibit significant locality, allowing us to maximize data reuse in SRAM. This design shares similarities with those of Cerebras [20] and Groq [5], which rely exclusively on SRAM and omit DRAM and HBM. However, MTIA 2i incorporates LPDDR DRAM to complement the large SRAM, rather than eliminating DRAM entirely.

MTIA 2i's SRAM provides 2.7 TB/s of bandwidth, whereas LPDDR offers just 204 GB/s—a 13x difference. Compared to MTIA 1, we doubled the SRAM capacity and tripled its bandwidth, while increasing DRAM bandwidth by only 1.4x. Each Processing Element (PE) provides 3x Local Memory for DMAing input and output tensor data to and from memory. We also doubled the bandwidth of Local Memory to the Dot Product Engine (DPE) compared to MTIA 1. This is critical for feeding the DPE with sufficient data, as it now contains two 32 x 32 MAC arrays, along with increased input

caches to accommodate the 2x larger effective tile size for matrix multiplications.

Achieving high effective FLOPS in the DPE requires careful optimization to keep the working set in SRAM. In particular, we need to choose the right model batch size to strike a balance between (1) keeping the batch size small enough for the activation buffer to fit in SRAM and (2) increasing the batch size to enhance the computational intensity and efficiency of GEMMs, which, in turn, makes it easier to hide the latency of blocking weight tiles in SRAM as they are read from DRAM.

Using LPDDR instead of HBM reduces costs but introduces limitations that make MTIA 2i unsuitable for certain models. With limited off-chip bandwidth, performance drops sharply as models reach a complexity and size that exceeds the SRAM capacity. We believe that 2 GF/sample is unattainable for MTIA 2i because GEMMs become DRAM bandwidth-bound, and the latency of dense and sparse networks becomes prohibitive even at small batch sizes. For example, running LLMs efficiently on MTIA 2i is challenging due to limited LPDDR bandwidth. For the Llama2-7B model, our evaluation shows that the prefill stage meets the time-to-first-token requirement of 600ms, but the decode stage fails to generate each additional token within the latency requirement of 60ms.

4 Model-Chip Co-design

Model-chip co-design is essential for the success of AI chips, yet it is often overlooked in prior publications. This section highlights our experience in co-designing models with MTIA 2i.

4.1 Autotuning Models

Optimizing each model requires carefully determining the configuration of various hardware knobs that we introduced in MTIA 2i to give us flexibility in handling model evolution over time. Given the large number of unique models we have at Meta, hand-optimizing each model obviously would not scale. Therefore, we have built an autotuning framework to pick optimal hyperparameters for both the hardware and the model.

Data placement: The placement of data in SRAM or DRAM has a significant impact on performance. The 256MB global SRAM shared across PEs is partitioned, at a granularity of 32MB, regions into hardware-managed cache (LLC) and software-managed scratch memory (LLS), with data in LLS not being automatically evicted by hardware. The autotuning framework automatically sizes the LLS and LLC using a simple approach: configure the LLS to hold the entire activation buffer and use the remaining SRAM for LLC. When the activation buffer is too large to fit, compare the performance of the nearest lower batch size where activations do fit in LLS with the current batch size with activations in LLC and pick the winner.

This works well in practice for the following reasons. The activation buffer is reused throughout model execution, i.e. the same memory can be used to back multiple activation tensors whose lifetimes do not overlap. But inputs, outputs, and weights on the other hand have a limited lifetime, often for a small number of operations, and therefore tend to waste LLS space because the capacity is unused during the rest of model execution. Also, the eviction of data in the activation buffer would trigger a writeback to DRAM which slows down execution whereas weights, being constant, can be evicted from LLC without a writeback. We are exploring algorithms for fine-grained data placement tuning improve those cases when the activation buffer does not fit in LLS.

Kernel tuning: Kernel tuning is one of the most important optimizations because different variants of a kernel can optimize for specific inputs with certain shapes, tensor data placement (SRAM versus DRAM), and data types. We built a kernel generator for Fully Connected (FC) layers to customize kernel variants based on input, output, and weight stationary. These kernel variants can adjust block sizes, DMA scheduling, and the usage of circular buffers. Initially, we ran exhaustive tests to cover all FC shapes in a model with different data placements, which proved to be too time-consuming. Consequently, we created a performance database and used approximate nearest neighbor search to pick FC kernel variants, which reduced FC tuning time by up to 1000x while achieving kernel performance within 5% of exhaustive FC tuning.

Batch size: To autotune a model's batch size, we build multiple snapshots of the model with different batch sizes and select the best performing one using traffic-replay tests.

Request coalescing: To autotune request coalescing, we run experiments to identify the optimal time window for coalescing requests and the number of windows that can be supported in parallel. We found that a model's throughput at its P99 latency SLO is highly sensitive to these parameters. With effective autotuning, we typically achieve >95% requests per batch.

Model sharding: To determine model sharding, we measure whether a model and its runtime buffers exceed the size of DRAM for a single device. If so, autotuning automatically explores how to shard the model across multiple devices.

Summary: We have successfully used autotuning to completely optimize models launched to production, with performance per total cost of ownership (Perf/TCO) and performance per unit of power consumption (Perf/Watt) matching or exceeding those of prior models that were manually optimized. We expect to autotune more parameters as new optimization techniques are developed.

4.2 Exploit Locality Across the Stack

Given MTIA 2i's large SRAM alongside LPDDR DRAM without HBM, optimizing for locality is expected. However, what's surprising is the extent of locality we can exploit, even with sparse networks [22] (i.e., embedding table lookups) in recommendation models that have irregular memory accesses across large tables. As models scale, sparse networks take up a smaller fraction of the overall execution time, and caching allows us to keep 40-60% of their accesses in SRAM. For dense networks, we can achieve over a 95% SRAM hit rate, thanks to optimizations such as buffer placement, graph optimizations (including fusion to reduce working set size), and an extra tiling level in the LLC for GEMM kernels.

Autotuning partitions the SRAM between LLS and LLC to maximize the SRAM hit rate. Additionally, we maximize data reuse by selecting the best operator scheduling algorithm for a model to minimize the liveness range required for activations. As a result, the LLC is primarily used for loading weights for FCs, minimizing performance loss from evicting dirty data. In cases where activations cannot fit entirely in LLS, we rely on memory hints supported by the hardware to skip the write-back to DRAM when we know

the tensor data will not be reused.

Graph optimizations, including fusions, were the most effective way to reduce activation buffer size. Fusions moved much of a sub-graph's working set into the distributed Local Memory of the PE grid by combining back-to-back (vertical), parallel (horizontal), or other operator combinations that would otherwise load and store inputs and outputs from LLS/LLC. One example is the sibling transpose FC fusion, where a transposed output is used as input for multiple FC layers; fusing these improved cache locality, resulting in up to a 15% performance gain for some models. We also observed that inputs with a low batch size, which had to be broadcasted to the model batch size, caused extra runtime and duplicated activation data. We modified the model publish flow to delay this broadcast, reducing the memory footprint of some models by up to 2x.

As recommendation models grew more complex, GEMMs with large weight tensors became a bottleneck on MTIA 2i, often DRAM bandwidth-bound. Increasing batch size to make them compute-bound was not viable, as it increased the live range of activations and risked overflowing SRAM. While we achieved 93% or higher efficiency for compute-bound models with tensors fitting in SRAM, more complex models shifted the challenge to saturating DRAM bandwidth. We found that activations typically fit in the distributed Local Memory of the PEs, so we implemented an algorithm to pre-load activations from LLS and broadcast weights across PE columns. This decoupled activation and weight loading, reducing PE contention. By leveraging hardware broadcast read support, we eliminated contention in the network-on-chip when reading weights. We also prefetched weight tiles into LLC to hide DRAM read latency. This approach improved latency by 45% and achieved over 95% DRAM bandwidth for shapes like 512 x 26592 x 2048 with a 109MB weight tensor. For larger batch sizes, where activations may not fit in LLS, we added an extra tiling level on the first dimension and prefetched activations as well.

4.3 Keeping Up with Model Evolution

As MTIA 2i was being developed based on the initial requirements, the field of AI evolved rapidly, introducing new modeling techniques such as sequence embeddings and HSTU [28]. These advancements presented challenging new kernels and shapes which were not always a natural fit for MTIA 2i's SIMD Engine, given its somewhat limited ISA. While GPUs could more easily address these challenges using their general-purpose Single-Instruction-Multiple-Threads (SIMT) cores, we found that MTIA 2i's heterogeneous PE architecture offered sufficient flexibility to handle these evolving workloads by utilizing the RISC-V vector core and employing creative approaches to use the SIMD Engine.

Sequence embeddings require jagged tensor [4], where each consecutive row may have a different length. Unlike pooled embeddings, which are dense with shape T x B x D (where T is the number of tables, B is the batch size, and D is the embedding dimension), sequence embedding lookup results are inputs to more complex compute operations rather than simple pooling. These complex operations include linear transformations, Hadamard products, or those found in Transformer. A variety of jagged tensor operators are needed, particularly for converting between sparse and dense formats and performing mathematical operations on jagged tensors. Using the SIMD Engine would be cumbersome, while the RISC-V vector core offers more flexibility given its shorter vector length and data-level parallelism of jagged tensor ops is more limited compared to dense operators.

Another model evolution requiring support is HSTU [28], which relies on a fused ragged attention operator. While quite similar to the multi-headed attention used in Transformer, it relies on a bias calculated from positional weights and timestamps. This bias calculation involves table index computations, which are then used to gather entries from these tables. Both steps can be parallelized with vector instructions. Additionally, we repurposed the lookup table (LUT) support in the SIMD Engine for the gather operation by performing it piecewise, loading each segment of the weights and timestamp tables into the limited LUT memory.

Finally, we faced challenges in handling LayerNorm and SoftMax as we onboarded new models with different shapes. LayerNorm requires three distinct steps to process the data: row-wise mean, row-wise variance, and element-wise result. This process involves a mixture of fixed-function commands and RISC-V vector instructions, balanced across the two cores in the PE. SoftMax was even more challenging because it involves five distinct steps, requiring careful pipelining across the RISC-V scalar and vector cores to balance data fetch and computation from the DMA engine, as well as between the SIMD Engine and vector instructions. Additional steps were needed to transpose the input when the inner dimension was small, ensuring full throughput for SIMD computation. While a simple, low-performance implementation of these kernels was relatively straightforward, the various acceleration engines in the PE enabled much higher performance with an efficient pipelined computation.

4.4 Limited Use of Quantization in Production

Despite the potential efficiency gains from quantization, FP16 remains the preferred choice for most of our recommendation models, even though MTIA 2i supports dynamic INT8 quantization. Dynamic INT8 quantization computes activation parameters on-the-fly, while static INT8 quantization uses fixed parameters from offline calibration. As a result, dynamic quantization adapts to different inputs and achieves better accuracy. MTIA 2i supports dynamic quantization parameter computation, with the reduction engine (RE) outputting min and max values per row after Matmul computation, and the SIMD engine computing row-wise quantization.

We evaluated per-tensor quantization, per-batch-item quantization (i.e., row-wise quantization with M as the batch dimension), and per-N batch-item quantization. We found that row-wise quantization of activations, combined with static INT8 quantization of weights, achieves model quality comparable to FP16. However, while the DPE performs 2x faster with INT8 compared to FP16, the overhead of quantization and dequantization for FC layers using this feature reduces the speedup to around 1.6x for large, compute-bound shapes (e.g., 2048 x 2048 x 2048).

Given the significant model development effort required, quantization is best suited for high-usage models deployed on many accelerators where efficiency gains are critical. For low-usage models, the effort may not be justified. As models grow in complexity, achieving a significant end-to-end performance gain with dynamic quantization becomes challenging. Additionally, layers closest to

the input and output of the dense compute network have the greatest impact on model accuracy and are therefore sensitive to quantization. Typically, only a few large layers show performance gains due to quantization, and even then, end-to-end improvements are often marginal (a few percent), unless layers causing quality degradation are quantized for more substantial gains (>5%). In practice, quantizing only the largest FC layers to amortize the overhead is most effective.

Despite these challenges, opportunities exist to increase quantization adoption by automating the process, including model quality analysis, and by enhancing future MTIA chip generations to support larger, more complex models with bigger FC layers that would benefit more from quantization.

5 Productionization Experience at Scale

During the productionization process of an AI chip, addressing challenges that were unanticipated or decisions deferred at the design stage due to ambiguity is critical for the chip's success. However, productionization experience is often overlooked in prior publications. This section shares our lessons learned from productionizing MTIA 2i at scale.

5.1 Trade-offs in Handling Memory Errors

We designed MTIA 2i as an efficient, low-power, small form-factor accelerator, opting for LPDDR DRAM, which lacks built-in support for Error-Correcting Code (ECC). As a result, ECC must be computed by the memory controller rather than at the memory itself. During the design phase, due to the lack of large-scale production data on LPDDR DRAM's error rate, we were uncertain whether to enable the controller-based, inefficient ECC in production. An alternative was to operate without ECC and address occasional memory errors through other means, given that inference results are inherently statistical. To make an informed decision, we adopted a multi-pronged approach to assess the impact of forgoing ECC.

First, as we scaled up the deployment of MTIA 2i in our datacenters, we collected memory error data at scale. From an initial sample of 1,700 servers, we found that 24% exhibited ECC errors, typically on a single MTIA card per server. This error rate was too high, and we lacked an efficient way to reliably detect them.

Second, we developed a memory error injection tool to identify which parts of a model (e.g., weights, activations, inputs, or outputs) are most sensitive to errors and how to mitigate them. We found that bit flips in Table Batched Embedding (TBE) indices, TBE table rows, or specific bits in floating-point representations of dense weights can cause NaNs or output corruptions, with some failures occurring with high probability. We considered using region-based ECC to protect these memory regions while leaving others unprotected, but it proved to be a difficult trade-off between performance and protection. We also prototyped software-based memory integrity checking, such as using hashing to detect corruptions, but found the overhead too high.

Third, we evaluated whether the software products using MTIA 2i could simply absorb the negative impact of memory errors, since the result of inference is inherently statistical. Through collaboration with the product teams, we learned that while the software products have mechanisms to detect product-level anomalies, such as a drop in advertisement revenue due to incorrect ranking outputs, the high volume of memory errors would overwhelm the operators, especially given the high risk of revenue or user engagement loss.

Ultimately, we concluded that enabling ECC is necessary for production-scale operation, despite the 10-15% throughput penalty associated with the inefficient memory-controller-based ECC. However, even with this penalty, MTIA 2i still delivers significant Perf/TCO gains over GPUs. All reported numbers in this paper already account for this penalty.

5.2 Overclocking at Scale

To further optimize MTIA 2i, we evaluated overclocking the chip to improve performance. The evaluation results led to an increase in the chip's operating frequency from the initial design specification of 1.1GHz to 1.35GHz. This increase surpassed our expectations, indicating sufficient frequency safety margins in the chip's design and manufacturing. With this 23% frequency increase, we observed end-to-end throughput improvements ranging between 5% and 20% in offline replayer tests for the models we evaluated. Overall, our experience highlights the importance of conducting a detailed overclocking study pre-production to maximize performance, despite careful consideration of clock frequency during the design phase.

Concretely, to assess the impact of overclocking, we conducted a large-scale study on the correlation between clock frequency, performance, and reliability, involving approximately 3,000 chips. For each chip, we conducted 10 tests, including performance tests, power tests, memory tests, kernel tests, module manufacturing tests, and functional PCIe tests, with each test focusing on specific aspects of chip functionality and performance. We compared the test results at three different frequencies (1.1GHz, 1.25GHz, and 1.35GHz) and observed negligible decreases in the test pass rate as the frequency increased from 1.1GHz to 1.35GHz. In production, we ensure stability through continuous monitoring at every stage, from manufacturing runs to deployment across the fleet.

5.3 Reducing Provisioned Power

When deploying new hardware, we initially set the rack power budget based on stress test and application load testing results at a small scale. We then adjust the budget as we gain experience from large-scale deployment. For MTIA 2i, after six months in production, we reduced the rack power budget by nearly 40% compared to initial estimates—a significantly larger reduction than with previously deployed mature hardware.

We attribute this higher-than-usual reduction to two factors. First, the initial power estimates were based on out-of-the-box models unoptimized for MTIA 2i. Second, the smaller chip size and more chips per server allowed for more granular resource allocation while maintaining adequate buffers for load spikes, reducing the likelihood of power capping due to overdraw.

After gaining sufficient production, we derived the new power budget using power data from the two largest and most demanding models. In one experiment, each of the 24 accelerators in a server was subjected to the P90 of peak throughput that the accelerators handled for these models in production. The likelihood of all 24 accelerators in a server simultaneously running the largest model and handling P90 peak traffic is very low. In another analysis, we determined the P90 power consumption of fully utilized servers

in production. The higher value from the experiment and analysis was selected as the new rack power budget. Although this approach led to a drastic reduction in the rack power budget by 40%, it has proven robust in production.

5.4 Advantages of Smaller Chips for Inference

Meta's recommendation models vary widely in size and complexity, resulting in differing demands for FLOPS, memory bandwidth, and I/O bandwidth. Some small models require only 10M FLOPS/sample and several GB of device memory, while large models demand over 1G FLOPS/sample and up to 100 GB of memory.

In the AI chip industry, there is a common belief that "bigger is better," with expectations for increasingly powerful devices each year to handle the latest models with ever-growing demands. However, we find that while the largest devices can support all of our models, they are not the most efficient. Smaller chips, with lower power consumption and peak FLOPS, provide distinct advantages, offering greater versatility across the range of models we serve and enabling higher device utilization.

When designing MTIA 2i, we did not expect it to handle the most complex or largest models, given its small device size. We anticipated that beyond a certain level of model complexity, meeting the P99 latency requirements for production serving while maintaining a sufficiently large batch size for reasonable throughput would become infeasible. However, throughout the process of launching models on MTIA 2i, we were pleasantly surprised to find that it could handle more complex models than initially expected. This was made possible by exploiting data locality to effectively leverage the large SRAM, which significantly boosts effective FLOPS.

Even more surprisingly, we found that MTIA 2i delivered significant efficiency gains compared to our GPU platform. Our testing showed an additional gain of 5% to 90% in Perf/TCO and Perf/Watt in production compared to offline traffic replay. In addition to MTIA 2i's inherent efficiency, the highly variable user load in production is also a factor in these gains. As a result, we must reserve substantial buffer capacity for peak demand, often leading to underutilization of devices. Larger, underutilized devices result in greater inefficiencies and waste, thus favoring smaller devices.

5.5 Real-time Firmware Updates for Mitigating Silicon Issues

An advantage of developing an AI chip in-house is the ability to frequently update firmware, drivers, and runtime libraries to enhance performance. Collectively termed the *firmware bundle*, these low-level software components are deployed atomically to ensure consistency and version compatibility. Furthermore, updates can be deployed immediately to address production issues caused by silicon design flaws—a task that is challenging with third-party silicon. The following example illustrates this.

In 2024, we enhanced the MTIA 2i stress test suite and found that when PE utilization was driven to 100% under certain conditions, approximately 1% of the servers under test experienced a loss of PCIe connectivity to the MTIA 2i chip. Further investigation revealed that about 0.1% of production servers serving certain models encountered a similar issue under high load. A detailed analysis identified the root cause as a subtle deadlock involving multiple components: the Control Core, the network-on-chip (NoC), and the host. Specifically, when the Control Core issued a request to read host memory under certain conditions, and the PCIe controller already had a queue of transactions in flight, PCIe transaction ordering rules required the response to the Control Core's request to wait for earlier transactions to complete. However, those transactions faced back-pressure blocking from the NoC, which serialized certain transactions and waited for the Control Core to complete an operation. Meanwhile, the Control Core was waiting for the host to complete the memory read transaction, resulting in a deadlock. To mitigate the issue, we deployed a firmware update that relocated the specific memory required by the Control Core from host memory to the device's SRAM. This quick change eliminated the need for the Control Core to access host memory under the specific condition, effectively preventing the deadlock. In contrast, debugging silicon reliability issues and implementing firmware mitigations typically take much longer with third-party GPUs.

In addition to instant firmware-bundle updates for production issues, we use Meta's continuous deployment tool [12] to regularly test and deploy firmware across the fleet. The tool builds firmware three times daily and subjects each build to stress testing on Meta's testing platform [9], where the issue described above was automatically detected. Not all builds are deployed to production. A typical rollout takes 18 days, starting in the staging environment and gradually scaling to the entire fleet. During this process, our cluster manager [25] enforces product team policies on server restarts to maintain service health. This incremental approach helps identify subtle issues, such as the 0.1% server impact noted earlier, while minimizing fleetwide disruptions. In emergencies, updates can be deployed fleetwide within three hours, adhering to safety policies to limit simultaneous server restarts. In extreme cases, the entire fleet can be updated within one hour by overriding these policies.

In 2024, we deployed 23 firmware-bundle releases fleet-wide, enabling timely production enhancements. In contrast, we were only able to deploy one or two firmware updates for third-party GPUs each year.

5.6 Large-Scale A/B Testing in Live Production

To establish MTIA 2i as a viable alternative to GPUs, we evaluated model quality and runtime efficiency at scale in live production. We developed an inference software stack that can serve the same model on either MTIA 2i or GPUs, flexibly shifting live production traffic and systematically comparing their results. This capability also enables incremental replacement of GPUs with MTIA 2i without disrupting production. Below are the details.

During post-training processing, our automation pipeline applies inference-optimized transformations, some accelerator-specific, to the same trained model to ensure an apples-to-apples comparison, generating runtime models suitable for serving on MTIA 2i and GPUs. During online serving, we split user traffic between MTIA 2i and GPUs in a controlled manner for fair A/B testing. For example, both MTIA 2i and GPUs are deployed globally within the same datacenter regions, ensuring they receive comparable user requests. We compare results holistically, including business metrics (e.g., ads revenue), system-level metrics (e.g., latency, throughput, error rate), high-level model metrics (e.g., normalized entropy [13] for evaluating prediction accuracy), and low-level model metrics (e.g.,

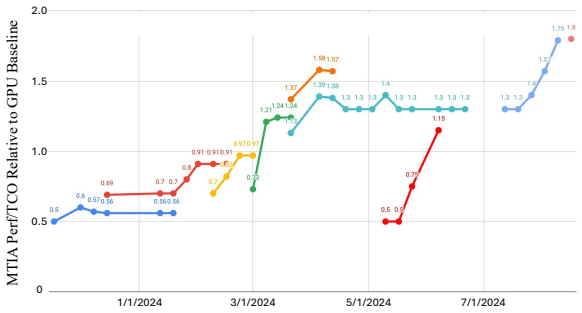

Figure 4: Continuous optimization improves MTIA 2i's Perf/TCO for a key ranking model used at Meta. The lines represent various variants of the model.

numerical accuracy and prediction value distributions).

Overall, these rigorous A/B tests in live production have confirmed that MTIA 2i meets SLOs, achieves comparable model quality, and significantly reduces Perf/TCO.

6 Case Study: Porting a Model to MTIA 2i

Existing models are primarily optimized for GPUs. Naturally, their performance on new accelerators may initially be inferior due to a lack of optimization. Our experience demonstrates that the success of an accelerator critically depends on optimizing and co-designing models to fully leverage its hardware features. In this section, we share our journey of improving a key model's Perf/TCO from an initially inferior 50% to a final, superior 180% compared to the GPU baseline, while achieving 2% higher Perf/Watt, as shown in Figure 4.

This model is among the top five recommendation models at Meta, responsible for generating a significant percentage of Meta's advertisement revenue. It is deployed across Meta's global datacenter regions and must meet strict latency requirements. Given the risks of negatively impacting advertisement revenue, launching this model required not only improvements in Perf/Watt and Perf/TCO compared to the GPU baseline but also high software reliability and stability in model quality.

Due to the model's significance for Meta, many ML engineers continuously fine-tune the model structure and features to enhance ranking quality. As a result, the model evolved significantly over the eight-month period from its initial test on MTIA 2i to its final production launch. During this process, the model's complexity increased from 140 MFLOPS/sample to 940 MFLOPS/sample and incorporated new modeling techniques, such as a network of multi-headed attention (MHA) blocks like those in traditional transformers [27]. It requires tens of gigabytes for the embedding tables, necessitating sharding across two accelerators. The model architecture is based on DHEN [30]; it contains stacked layers with skip connection and layer normalization where each layer contains an ensemble of a Factorization Machine Block and a Linear Compression Block [29].

Multiple co-design optimizations helped improve the model's Perf/TCO. First, we identified that certain graph patterns in the model can be transformed into more efficient sub-graphs or fused into a single kernel to best leverage MTIA 2i's hardware. For example, we fused multiple parallel FC layers along with a common input transpose into a single operator, which shrunk the activation

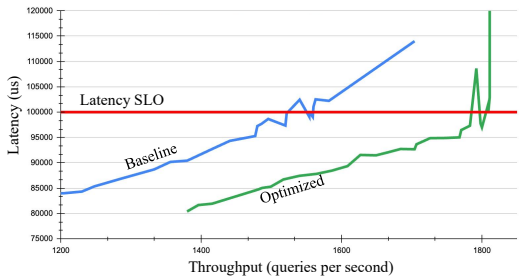

Figure 5: Consolidating TBE instances leads to significant throughput improvement.

size and improved the cache hit rate. We also identified hundreds of LayerNorm layers that can be batched together horizontally to amortize the kernel launch overhead. In the MHA blocks, we replaced a sequence of operators (i.e., Slice, Reshape, Concat) with a single custom Transpose kernel on MTIA 2i. Finally, we reduced the merge network latency by 50% by selecting the FC kernel variant most optimized for each shape, especially for some of the largest FCs with weight tensors which required an additional level of blocking by prefetching into SRAM.

Second, we rejected model changes that would undermine SRAM's effectiveness and identified an alternative that achieves similar results while being more SRAM-friendly. The initial change, which would have tripled the remote embedding inputs to the merge network, caused a 90% drop in throughput because the increased activation buffer size could no longer be pinned in SRAM and had to be fetched from LPDDR. We achieved similar improvements in model quality by adding two DHEN layers to deepen the computational portion of the merge network. These additional layers were small enough to keep the activations pinned in SRAM.

Third, we performed a model transformation to better accommodate a new model change. We had previously developed an optimization called In-Batch Broadcast (IBB) that broadcasts (i.e., expands) the user-side inputs to the merge network, aligning user-ad pairs for the interaction and prediction layers. However, due to a new model change, the early stage of the merge network now requires only the user-side inputs. Accordingly, we deferred the user-side broadcast to a later stage, reducing the user-side inputs processed in the early stage and increasing throughput by 17%.

Fourth, we improved job scheduling on MTIA 2i to increase throughput while still meeting the latency SLO. The problem we initially encountered was rooted in how we serve models. We typically partition our models into remote (sparse) and merge (dense) networks, where one or more remote networks produce the accumulated sparse feature output, which is then input to a single merge network. For models with large embeddings that cannot fit in DRAM, we shard them across multiple accelerators. We may also end up with multiple shards per device because TBE operators can be either unweighted or weighted, with a weight matrix applied to the accumulated embeddings to produce the final output.

In this setup, we observed low device utilization due to the inefficient ordering of jobs for the merge and remote networks. For each batched request, we ideally want the job ordering on a single device to be remote-merge-remote-merge to minimize latency. However, in the traces, we observed that jobs were sometimes scheduled in the order of remote-remote-merge-merge, meaning that a remote

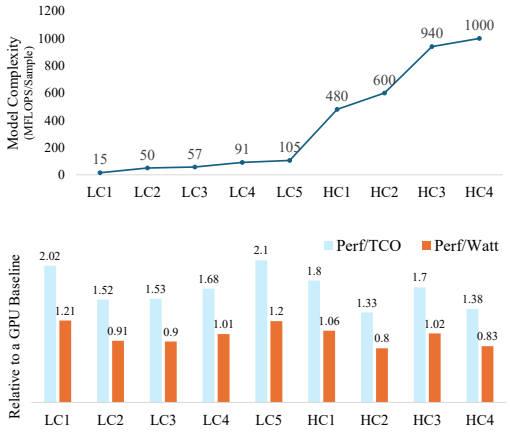

Figure 6: Complexity and efficiency of models on MTIA 2i.

for a subsequent request was scheduled before the merge of the preceding request, which increased its overall latency. The overall impact was low device utilization while still meeting the 99th percentile (P99) latency SLO of 100ms.

To improve device utilization, we decided to consolidate the weighted and unweighted TBE instances into a single job, thereby reducing the number of remote jobs by half. This resulted in a significant improvement in throughput, as shown in Figure 5. Note that the execution time of the merge and remote jobs on the MTIA 2i PE grid remains the same in both cases, so the gains were realized higher in the serving stack, where the scheduling benefits became visible. The measured P99 request latency decreased by 13ms, from 99ms to 86ms. Breaking this down, we observe a similar 13ms improvement in the merge request latency, while the remote request latency remains unchanged, indicating that the gains were derived from higher occupancy of merge jobs on the devices.

7 Supporting Diverse Models

Our production experience has given us confidence that MTIA 2i offers advantages across a wide range of models. To demonstrate this, we show in Figure 6 the Perf/Watt and Perf/TCO of nine production models. We classify the first five models as Low Complexity (LC) ranging from 15 to 105 MFLOPS/sample, and the last four models as High Complexity (HC) ranging from 480 to 1000 MFLOPS/sample. Each of these models runs on one or two accelerators. Although there is no clear trend for efficiency as a function of complexity, the highest efficiency was achieved on LC models, namely LC1 and LC5, while the lowest efficiency was observed on HC models, namely HC2 and HC4. This is consistent with the fact that a model's complexity and size determine how well it fits into the global SRAM and, therefore, how well it can utilize the Dot Product Engine and SIMD Engine.

Although Figure 6 highlights model complexity in FLOPS, a model's efficiency also depends on other factors, such as the structure of the model graph, the operators, their shapes, and the batch size. For example, HC3, the model described in Section 4, shows high efficiency because it is co-designed with additional DHEN layers that increase computation intensity without significantly increasing the memory footprint. Similarly, HC1 achieves even higher efficiency because its small memory footprint allows its batch size to be pushed to 2K, the largest of any model with more than 100 MFLOPS/sample. LC1 is another interesting example; it is optimized to run at a 4K batch size and thus achieves higher efficiency than Model2, which only runs at a batch size of 512, despite both having very low complexity.

The achieved performance of models also depends on the amount of optimization effort invested. The time we spend optimizing models is proportional to their importance, such as their impact on revenue, user experience, or the amount of capacity they consume. Specifically, we spent more time optimizing HC1 compared to HC2 and HC4, as it accounts for a much higher fraction of revenue. It is also worth noting that HC2 has a more challenging set of serving features to manage, which resulted in more host-side overhead and lower overall efficiency. Finally, when working with an in-house custom AI chip, it is easier to outperform GPUs in Perf/TCO than in Perf/Watt, due to GPUs having a much longer history in power optimization.

8 Challenges and Limitations

While MTIA 2i has been highly successful at Meta, it also faces challenges and limitations. Our intentional divergence from GPUs in the architecture, programming model, and capabilities has introduced a set of consequences.

Unsuitable for large language models (LLMs). As MTIA 2i was designed prior to the boom of LLMs sparked by ChatGPT, it was optimized for ranking and recommendation models. Given its limited FLOPS and memory bandwidth, as well as the lack of a low-latency, high-bandwidth communication network, it is unsuitable for running large models such as Llama3 70B or 405B. However, we evaluated MTIA 2i for Llama3 8B and found that, while we could achieve acceptable performance during the prefill phase, the decode phase failed to meet the latency requirement. During decode, both the MHA and FFN portions of each transformer layer are limited by the LPDDR bandwidth.

Unstuitable for highly complex models. Given the latency requirements of serving in production, we will run into a limit on model complexity where it is no longer cost effective to use MTIA 2i and we will hit this limit sooner than a contemporary GPU given it has higher peak FLOPS and HBM. However, we believe there is significantly more performance headroom and we should be able to handle at least 2 GFLOPS/sample models. We also have shown that MTIA 2i can handle HSTU-based ranking models (>10 GFLOPS/sample) efficiently at low batch sizes. For future generations of MTIA, we plan to increase their peak FLOPS to handle more complex models.

Maturing MTIA software ecosystem. While a small number of models consume a large fraction of our datacenter capacity, and we can afford dedicated manual optimizations for these models due to their importance, Meta has many models with small to medium capacity demands, making manual optimization intractable. Although we have designed MTIA software to scale with new models—enabling functional support, numerics debugging, validation, and performance autotuning—this remains an ongoing challenge,

especially given the rapid evolution of our models and developer environment. In comparison, GPUs have undergone a longer evolution and have a more mature software ecosystem.

9 Conclusion

We presented an overview of MTIA 2i, highlighting our experience with model-chip co-design and the large-scale productionization of MTIA 2i. Encouraged by MTIA 2i's significantly lower total cost of ownership compared to GPUs, we are accelerating the development of Meta's next-generation AI chips. We expect the industry-wide shift among major IT companies toward in-house AI chip development to gain further momentum, driving innovative designs with distinctive features. Finally, given the tension between the lengthy development cycles of AI chips and the rapid evolution of the AI field, we believe that a deep understanding of AI workloads, reasonable projections of AI technology trends, and maintaining flexibility in chip design without overly adding complexity are essential for the success of future AI chips.

References

[1] 2024. Azure Maia for the era of AI: From silicon to software to systems. https://azure.microsoft.com/en-us/blog/azure-maia-for-the-era-of-ai-from-silicon-to-software-to-systems/

[2] 2024. Introducing Enhanced Gen AI Features and Other Tools to Help Build Your Business. https://www.facebook.com/business/news/Introducing-Enhanced-Gen-AI-Features-and-Other-Tools-to-Help-Build-Your-Business

[3] 2024. Meta's AI Products Just Got Smarter and More Useful. https://about.fb.com/news/2024/09/metas-ai-product-news-connect/

[4] 2024. PyTorch Jagged Tensor Operators. https://pytorch.org/FBGEMM/fbgemm_gpu-overview/jagged-tensor-ops/JaggedTensorOps.html

[5] Dennis Abts, John Kim, Garrin Kimmell, Matthew Boyd, Kris Kang, Sahil Parmar, Andrew Ling, Andrew Bitar, Ibrahim Ahmed, and Jonathan Ross. 2022. The groq software-defined scale-out tensor streaming multiprocessor: From chips-to-systems architectural overview. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 1–69.

[6] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. 2024. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 929–947.

[7] Alexis Bjorlin. 2022. OCP Summit 2022: Open hardware for AI infrastructure. https://engineering.fb.com/2022/10/18/open-source/ocp-summit-2022-grand-teton/.

[8] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[9] Mike Chow, Yang Wang, William Wang, Ayichew Hailu, Rohan Bopardikar, Bin Zhang, Jialiang Qu, David Meisner, Santosh Sonawane, Yunqi Zhang, Rodrigo Paim, Mack Ward, Ivor Huang, Matt McNally, Daniel Hodges, Zoltan Farkas, Caner Gocmen, Elvis Huang, and Chunqiang Tang. 2024. ServiceLab: Preventing Tiny Performance Regressions at Hyperscale through Pre-Production Testing. In *Proceedings of the 28th Symposium on Operating Systems Principles*.

[10] Amin Firoozshahian, Joel Coburn, Roman Levenstein, Rakesh Nattoji, Ashwin Kamath, Olivia Wu, Gurdeepak Grewal, Harish Aepala, Bhasker Jakka, Bob Dreyer, et al. 2023. Mtia: First generation silicon targeting meta's recommendation systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[11] Xinwei Fu, Zhen Zhang, Haozheng Fan, Guangtai Huang, Mohammad El-Shabani, Randy Huang, Rahul Solanki, Fei Wu, Ron Diamant, and Yida Wang. 2024. Distributed training of large language models on AWS Trainium. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 961–976.

[12] Boris Grubic, Yang Wang, Tyler Petrochko, Ran Yaniv, Brad Jones, David Callies, Matt Clarke-Lauer, Dan Kelley, Soteris Demetriou, Kenny Yu, and Chunqiang Tang. 2023. Conveyor: One-Tool-Fits-All Continuous Software Deployment at Meta. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation*.

[13] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9.

[14] Yang Jiao, Liang Han, and Xin Long. 2020. Hanguang 800 NPU—The Ultimate AI Inference Solution for Data Centers. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 1–29.

[15] Norman P Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, and Cliff Young. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–14.

[16] Roman Kaplan. 2024. Intel Gaudi 3 AI Accelerator: Architected for Gen AI Training and Inference. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE, 1–16.

[17] Hanjoon Kim, Younggeun Choi, Junyoung Park, Byeongwook Bae, Hyunmin Jeong, Sang Min Lee, Jeseung Yeon, Minho Kim, Changjae Park, Boncheol Gu, et al. 2024. TCP: A Tensor Contraction Processor for AI Workloads Industrial Product. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 890–902.

[18] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. 2021. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 789–801.

[19] Cedric Lichtenau, Alper Buyuktosunoglu, Ramon Bertran, Peter Figuli, Christian Jacobi, Nikolaos Papandreou, Haris Pozidis, Anthony Saporito, Andrew Sica, and Elpida Tzortzatos. 2022. AI accelerator on IBM Telum processor: Industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 1012–1028.

[20] Sean Lie. 2023. Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning. *IEEE Micro* 43, 3 (2023), 18–30.

[21] Mahesh Maddury, Pankaj Kansal, and Olivia Wu. 2024. Next Gen MTIA - Recommendation Inference Accelerator. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. 1–27. https://doi.org/10.1109/HCS61935.2024.10665192

[22] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[23] Raghu Prabhakar. 2024. SambaNova SN40L RDU: Breaking the Barrier of Trillion+ Parameter Scale Gen AI Computing. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE, 1–24.

[24] Alan Smith and Vamsi Alla. 2024. AMD Instinct MI300X Generative AI Accelerator and Platform Architecture. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 1–22.

[25] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matt Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A Unified Cluster Management System for Shared Infrastructure. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*. 787–803.

[26] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 10–19.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Łukasz Kaiser. 2017. Attention Is All You Need. *Advances in Neural Information Processing Systems* (2017).

[28] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152* (2024).

[29] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. 2024. Wukong: Towards a Scaling Law for Large-Scale Recommendation. *arXiv preprint arXiv:2403.02545* (2024).

[30] Buyun Zhang, Liang Luo, Xi Liu, Jay Li, Zeliang Chen, Weilin Zhang, Xiaohan Wei, Yuchen Hao, Michael Tsang, Wenjun Wang, et al. 2022. DHEN: A Deep and Hierarchical Ensemble Network for Large-Scale Click-Through Rate Prediction. *arXiv preprint arXiv:2203.11014* (2022).