Technical Report

# Moreh-Tenstorrent AI Data Center Solution System Architecture

Moreh, Inc.

October 2025



MOREH

# Contents

# Moreh-Tenstorrent AI Data Center Solution System Architecture

## Overview

Moreh's mission is to provide alternative options to NVIDIA GPUs for AI data centers through advanced software technologies. As part of this effort, we have been working closely with Tenstorrent and will be launching a data center solution in Q4 2025. Tenstorrent, led by legendary semiconductor architect Jim Keller, delivers scalable hardware through network-integrated AI chips. On top of that, Moreh adds its unique cluster architecture and software for efficiently utilizing many chips, completing a full-stack solution. We are confident that this is the best option to minimize the total cost of ownership (TCO) of AI data centers.

This article describes the architecture of the Tenstorrent solution we provide. Our approach, chip architecture, cluster architecture, and software architecture are fundamentally differentiated from conventional NVIDIA GPUs and DGX systems. We explain how this enables us to optimize large-scale AI infrastructure. Below is a summary of our differentiators:

- **Approach**
  - We employ a larger number of lighter chips compared to GPUs, achieving high performance and efficiency at the cluster level rather than at the individual chip level.
  - To realize this, scalable network architecture and software capable of efficiently leveraging such many chips are essential.
  - Since individual chips do not require extremely high performance, they can be built on older process nodes (e.g., 6 nm or 12 nm) and use GDDR memory instead of HBM, thereby maximizing overall cost efficiency.
  - The chips are not limited to inference but can be used for both training and inference. This is a crucial factor for large-scale AI data centers when adopting a new type of processor.
- **Chip architecture**
  - Large software-managed SRAMs (approximately 1.5 MB per core) are adopted instead of a complex hardware-managed memory hierarchy such as coherent shared caches. With proper software support, this can minimize off-chip memory bandwidth requirements.

  - Intra-chip inter-core communication is performed explicitly through a 2D torus Network-on-Chip (NoC), rather than indirectly via shared memory or caches. This allows direct data exchange between cores without consuming bandwidth from off-chip memory or shared caches, while giving the software more room to optimize data movement.
  - A block floating-point format is supported, where 16 adjacent elements share a common exponent. This reduces memory footprint and bandwidth requirements by approximately half, without causing significant impact on accuracy.
- **Cluster architecture**
  - Each chip is equipped with built-in Ethernet interfaces, enabling direct data transfer between two linked chips with low latency and without CPU intervention.
  - Multiple chips are interconnected through a torus network, without requiring a complex switch network (similar to Google's TPU clustering approach). A torus network is beneficial for communication patterns of typical AI workloads.
- **Software architecture**
  - We provide an inference framework that performs distributed inference across multiple nodes and chips, presenting them as a single unified endpoint, and a training framework that allows multiple nodes and chips to operate as a single PyTorch device.
  - Data distribution, task allocation, and inter-chip communication are automated by software. Consequently, although the number of chips increases compared to a GPU cluster, the overall infrastructure becomes easier to utilize, with workloads distributed to enable efficient communication over the torus network.

## Overall System Architecture

Figure 1 conceptually illustrates the overall hardware hierarchy of the Tenstorrent AI data center solution. Two chip generations – *Wormhole* and *Blackhole* – are currently available. Although they differ in detailed specifications, their architecture is largely similar.

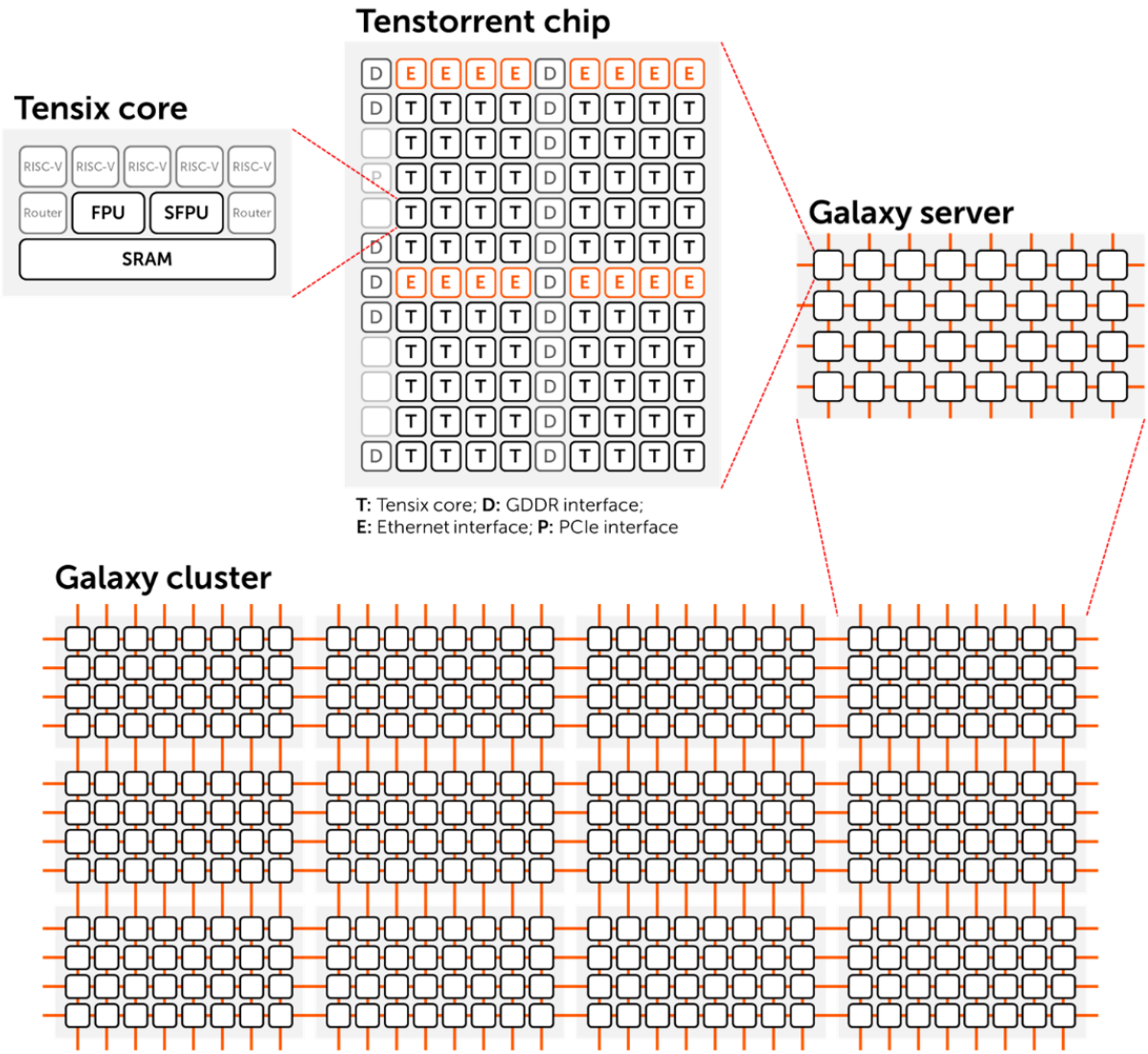


Figure 1. Hardware hierarchy from Tensix cores to Tenstorrent chips, Galaxy servers, and Galaxy clusters.

Each chip integrates numerous *Tensix* cores, general-purpose RISC-V cores (available only in Blackhole), GDDR memory interfaces, Ethernet interfaces, and PCIe interfaces, all connected through a unified 2D torus NoC. The Tensix cores are specialized for tensor operations in AI workloads and include an FPU for matrix multiplication, an SFPU for vector operation, and local SRAM.

A *Galaxy* server is equipped with 32 Tenstorrent chips. We expect that roughly four Tenstorrent chips will deliver performance comparable to a single GPU. Specifically, four Wormhole chips correspond to one NVIDIA A100, and four Blackhole chips correspond to one NVIDIA H100. Therefore, a single Galaxy server with 32 chips is equivalent in performance to a conventional 8-GPU server.

Each chip's Ethernet interfaces can be directly connected to those of other chips without passing through network switches. According to our cluster architecture, Wormhole is connected to neighboring chips in four directions – *east*, *west*, *north*, and *south* – while Blackhole adds two more connections in the *up* and *down* directions, for a total of six. Within a Galaxy server, the 32 chips are pre-connected in hardware in an 8×4 mesh. The Ethernet interfaces located at the "edges" of the mesh are exposed as the server's Ethernet ports, which are cabled to those on neighboring servers. As a result, chips across multiple servers are interconnected in a flat torus network, seamlessly linking the inside and outside of each server without any hierarchical distinction.

From the software perspective, Tenstorrent provides an open-source low-level SDK called *TT-Metalium*. It includes a runtime system, kernels (TT-LLK) and a compiler for the Tensix cores, and an inter-chip Ethernet communication stack (TT-Fabric). On top of that, we have added our own proprietary software, including tensor operation and communication libraries (independent of Tenstorrent's TT-NN and CCL) as well as inference and training frameworks.

# Wormhole and Blackhole Processors

## Hardware Specifications

Table 1 summarizes the hardware specifications of Wormhole and Blackhole. Note that the chips operate with different specifications when used on PCIe cards versus in Galaxy servers (e.g., available cores, theoretical performance, memory bandwidth, TDP, and interconnect configuration). All figures in the table represent the specifications for the Galaxy server configuration.

Table 1. Specifications of Wormhole and Blackhole

| Items | Wormhole | Blackhole |
|---|---|---|
| **Basic facts** | | |
| Lithography | GlobalFoundries 12 nm | TSMC 6 nm |
| AI core architecture | Tensix | Tensix |
| # AI cores | 80 (72 available) | 140 |
| General-purpose core architecture | - | RISC-V |
| # general cores | - | 16 |
| Clock frequency | 1.0 GHz | 1.35 GHz |
| TDP | 250 W | 300 W |
| **Theoretical peak performance** | | |
| TF32 matrix | 73.7 TFLOPS[1] | 193.5 TFLOPS |
| FP16 matrix | 73.7 TFLOPS[1] | 193.5 TFLOPS |
| BF16 matrix | 147.5 TFLOPS[1] | 387.1 TFLOPS |
| BLOCKFP8 matrix | 147.5 TFLOPS[1] | 387.1 TFLOPS |
| FP8 matrix | 294.9 TFLOPS[1] | 774.1 TFLOPS |
| BLOCKFP4 matrix | 294.9 TFLOPS[1] | 774.1 TFLOPS |
| **Network-on-Chip** | | |
| Individual link bandwidth | 32 B/cycle | 64 B/cycle |
| Aggregated bandwidth | 15.36 TB/s | 70.50 TB/s |
| **On-chip SRAM** | | |
| Total SRAM | 114 MB | 210 MB |
| SRAM per core | 1.4 MB | 1.5 MB |
| **Off-chip memory** | | |
| Technology | GDDR6 | GDDR6 |
| Capacity | 12 GB | 32 GB |
| Peak bandwidth | 336 GB/s | 512 GB/s |
| **Connectivity** | | |
| Host-to-chip interface | PCIe Gen4 x8 or x1 | PCIe Gen5 x8 or x1 |
| Host-to-chip bandwidth | 32 GB/s | 64 GB/s |
| Chip-to-chip interface (physical) | 16x 100 Gbps Ethernet | 10x 400 Gbps Ethernet |
| Chip-to-chip interface (aggregated) | 4x 400 Gbps | 4x 800 Gbps + 2x 400 Gbps |
| Chip-to-chip total bandwidth | 400 GB/s | 1,000 GB/s |

[1] Due to the limitations in the runtime implementation of TT-Metalium, only 72 out of 80 Tensix cores on Wormhole can be used for computation. Therefore, the theoretical peak performance of Wormhole shown here represents the performance of these 72 cores, rather than that of all 80 cores.

## Comparison with NVIDIA GPUs

Table 2 compares the specifications between NVIDIA GPUs and Tenstorrent chips. As mentioned earlier, we consider four Wormhole chips to correspond to one NVIDIA A100 80 GB SXM, four Blackhole chips to one NVIDIA H100 SXM, and eight Blackhole chips to one NVIDIA B200. Based on these equivalences, we compare computational performance, SRAM (or L2 cache) capacity, DRAM capacity and bandwidth, and inter-chip communication bandwidth – NVLink for NVIDIA and Ethernet for Tenstorrent. The values in parenthesis indicate the relative levels of Tenstorrent chips compared to NVIDIA GPUs; higher is better.

Roughly speaking, four or eight Tenstorrent chips offer higher computational performance, more than 10x larger SRAM capacity compared to GPU's L2 cache, and higher inter-chip communication bandwidth compared to NVIDIA's NVLink. On the other hand, their memory bandwidth is only about half that of GPUs, but this limitation can be mitigated through the effective use of large on-chip SRAM and the block floating-point data format.

Table 2. Comparison between NVIDIA GPUs and the corresponding four or eight Tenstorrent chips

| Items | A100 | 4x Wormhole | H100 | 4x Blackhole | B200 | 8x Blackhole |
|---|---|---|---|---|---|---|
| TF32 matrix[2] (TFLOPS) | 156 **(1.00x)** | 295 **(1.89x)** | 495 **(1.00x)** | 774 **(1.56x)** | 1,100 **(1.00x)** | 1,548 **(1.41x)** |
| BF16 matrix[2] (TFLOPS) | 312 **(1.00x)** | 590 **(1.89x)** | 989 **(1.00x)** | 1,548 **(1.56x)** | 2,200 **(1.00x)** | 3,097 **(1.40x)** |
| FP8 matrix[2] (TFLOPS) | - | 1,180 | 1,979 **(1.00x)** | 3,097 **(1.56x)** | 4,500 **(1.00x)** | 6,193 **(1.38x)** |
| SRAM or L2 cache capacity (MB) | 40 **(1.00x)** | 456 **(11.4x)** | 50 **(1.00x)** | 840 **(16.8x)** | 50 **(1.00x)** | 1,680 **(33.6x)** |
| DRAM capacity (GB) | 80 **(1.00x)** | 48 **(0.60x)** | 80 **(1.00x)** | 128 **(1.60x)** | 192 **(1.00x)** | 256 **(1.33x)** |
| DRAM peak bandwidth (GB/s) | 2,039 **(1.00x)** | 1,344 **(0.66x)** | 3,352 **(1.00x)** | 2,048 **(0.61x)** | 7,700 **(1.00x)** | 4,096 **(0.53x)** |
| Chip-to-chip interconnect bandwidth (GB/s) | 600 **(1.00x)** | 1,000[3] **(1.67x)** | 900 **(1.00x)** | 2,800[3] **(3.11x)** | 1,800 **(1.00x)** | 5,200[3] **(2.89x)** |

[2] All performance figures are based on dense matrix computations.

[3] The aggregate Ethernet bandwidth of four Wormhole chips is 1,600 GB/s, not 1,000 GB/s. However, some of their Ethernet links must be used to interconnect them, which is unnecessary for a single NVIDIA GPU. To ensure a fair

## Tensix Core Architecture

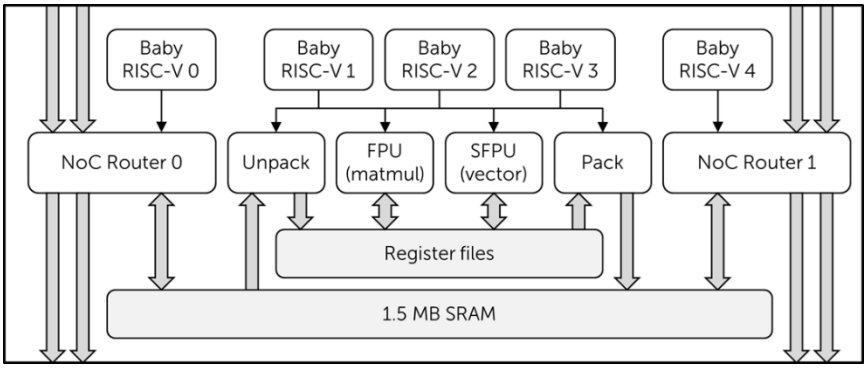


Figure 2. The architecture of a Tensix core.

The Tensix core is the key building block of Tenstorrent processors. Figure 2 shows the overall structure of a Tensix core. Each core contains two computation units: an FPU for matrix multiplication and an SFPU for vector operations. It also includes 32 KB of destination registers and approximately 1.5 MB of local SRAM.

In general, data are first transferred from off-chip memory or from other cores/chips through the NoC (to be described later) and stored in the SRAM. The data are then fetched from the SRAM into the registers (called *unpacking*), processed by the FPU or SFPU, written back from the registers to the SRAM (called *packing*), and finally sent back through the NoC to off-chip memory or to other cores/chips. During this process, data are typically handled in tiles of 32×32 elements.

Each Tensix core contains five “baby” RISC-V sub-cores to control these components simultaneously. Although the roles of each sub-core is not strictly defined, typically one handles data movement from the NoC to the SRAM, one manages unpacking, one issues FPU/SFPU instructions, one manages packing, and one handles data movement from the SRAM to the NoC.

In other words, instead of implementing out-of-order execution at the hardware level, five baby RISC-V sub-cores run different kernel codes to explicitly achieve instruction-level parallelism (ILP). This approach simplifies the hardware design but requires specialized expertise to implement high-performance Tensix kernels. As Tenstorrent’s leading software partner, Moreh possesses this expertise and has developed and delivered optimized tensor operation libraries.

The SRAM of a Tensix core is much larger than the L1 cache of a GPU (for example, the L1 cache of the NVIDIA A100 is 192 KB). At the same time, as mentioned earlier, the total SRAM capacity – calculated as per-core SRAM size × the number of cores –

comparison, the bandwidth figures presented here exclude the links required to interconnect four or eight Tenstorrent chips in a linear topology.

is also much greater than the L2 cache of a GPU. Because SRAM has its own address space, separate from off-chip memory that is explicitly allocated and accessed by software, we can precisely track which data are stored in each core's SRAM across multiple kernel executions. As a result, more intermediate results (e.g., activations) can reside in SRAM, thereby reducing off-chip memory accesses without requiring explicit kernel fusion as in GPUs. This is one of the reasons Tenstorrent chips can adopt cost-effective GDDR instead of HBM.

## Chip Layout and the Network-on-Chip

All components in a Tenstorrent chip, including Tensix cores, general-purpose RISC-V cores, and various interfaces are arranged in a grid structure. Figure 3 illustrates the logical grid layouts[4] of the Wormhole and Blackhole chips.

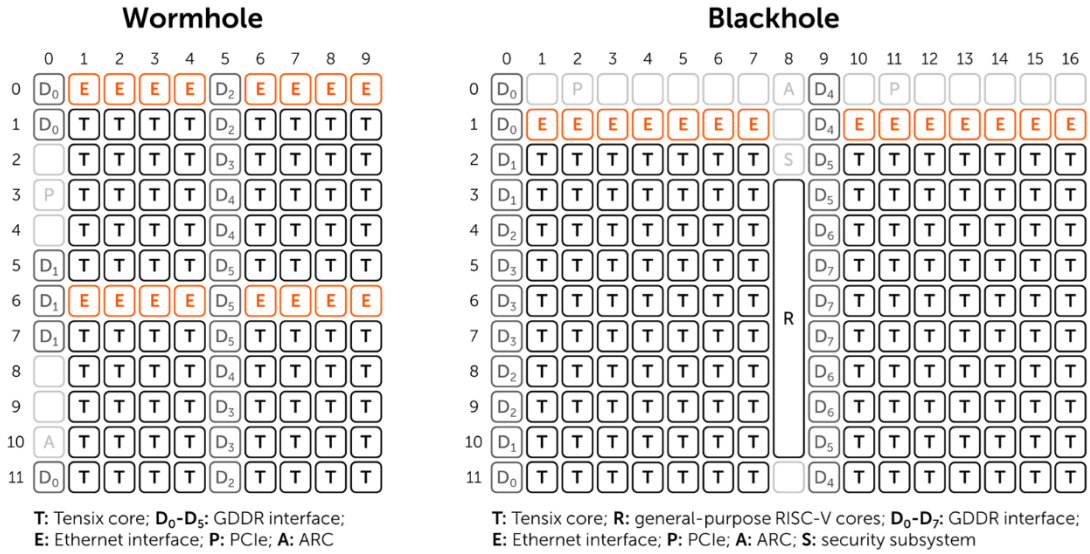


Figure 3. Logical placement of different tiles (Tensix, general-purpose RISC-V, GDDR, Ethernet, PCIe, ARC, and security subsystem tiles) of Wormhole and Blackhole chips.

The Wormhole chip is organized as a 10×12 grid, in which 80 tiles are used for Tensix cores, 18 for GDDR interfaces, 16 for Ethernet interfaces, 1 for a PCIe interface, 1 for an ARC core, and the remaining 4 tiles are left empty. The Blackhole chip is organized as a 17×12 grid, in which 140 tiles are used for Tensix cores, 8 for general-purpose RISC-V cores, 24 for GDDR interfaces, 14 for Ethernet interfaces, 8 for PCIe interfaces, 6 for SerDes modules, 2 for an ARC core and a security subsystem, and the remaining 2 tiles are left empty.

[4] This figure illustrates the logical layout defined by the NoC, not the actual physical placement. The discrepancy between the two is due to the physical implementation of the 2D torus NoC, which is explained on the following page: https://tenstorrent.com/vision/community-highlight-tenstorrent-wormhole-series-part-1-physicalities

All tiles are interconnected through a 2D torus topology NoC. Each tile can exchange data bidirectionally with its neighboring tiles in the four directions – east, west, north, and south – at 32 bytes/cycle in Wormhole and 64 bytes/cycle in Blackhole. Each tile contains two NoC routers, which relay communication between non-adjacent tiles.

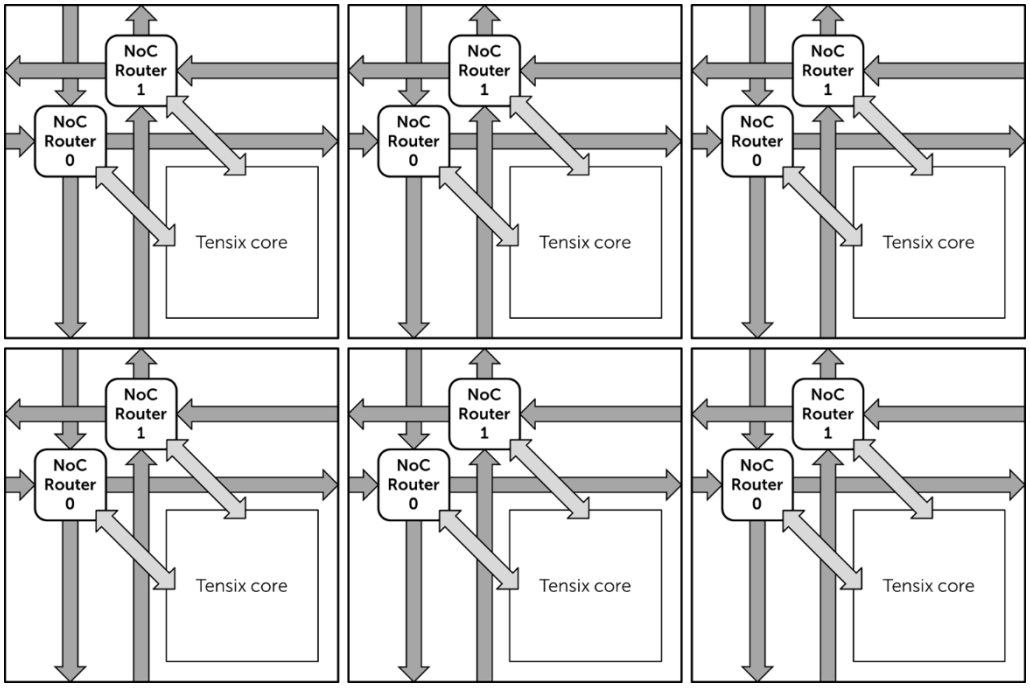


Figure 4. A logical diagram illustrating the NoC connections between adjacent tiles. NoC Router 0 handles data movement in the +X and +Y directions, while NoC Router 1 handles data movement in the -X and -Y directions.

All on-chip data movement – for example, between two Tensix cores, between a Tensix core and GDDR memory, between a Tensix core and an Ethernet interface, or between GDDR memory and an Ethernet interface – occurs through this unified NoC. Specifically, the NoC provides a single address space that encompasses all Tensix core SRAMs, off-chip GDDR memory, and Ethernet RX/TX queues. Arbitrary data transfers can be implemented by programming the Tensix cores to read from and write to specific NoC addresses.

This not only simplifies the chip architecture but also enables efficient and flexible data movement that is not possible in conventional GPUs. For instance, each core can directly send data from its own SRAM to Ethernet, bypassing GDDR as an intermediary. This reduces communication latency and saves memory bandwidth.

The NoC supports multicast, allowing the same data to be efficiently broadcast to multiple tiles arranged in a row or a column. This capability enables efficient loading of off-chip memory data during matrix multiplications.

## Block Floating-Point Types

The block floating-point format is one of the approaches used to store real numbers in a compressed format. It is based on the observation that adjacent values belonging to the same array (tensor) generally share similar scales. A recent example is the Microscaling (MX) format, which has been adopted in some modern AI accelerators.

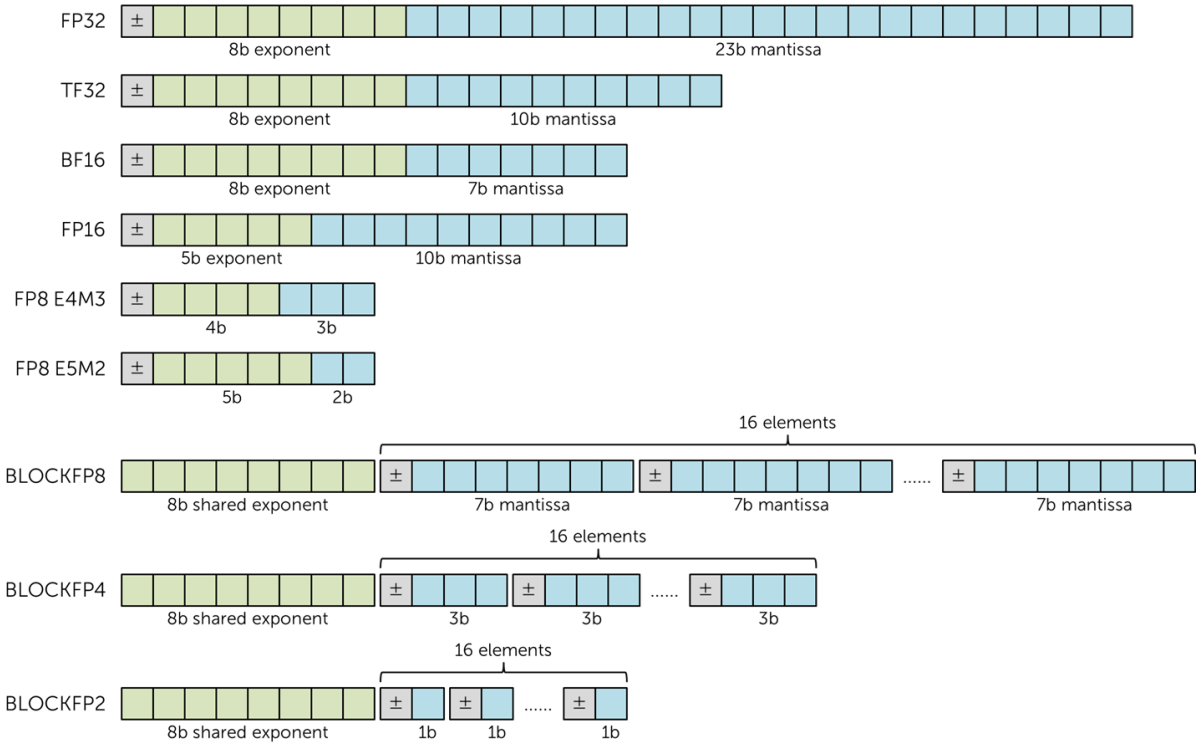


Figure 5. Comparison among FP32, TF32, BF16, FP16, FP8 (E4M3 and E5M2), and Tenstorrent's BLOCKFP8, BLOCKFP4, and BLOCKFP2 formats.

Tenstorrent processors support their own block floating-point formats – BLOCKFP8, BLOCKFP4, and BLOCKFP2. Each element has its own sign and mantissa, while every group of 16 adjacent elements shares a single exponent. Figure 5 illustrates several floating-point formats for comparison. BLOCKFP8 uses a 7-bit mantissa and an 8-bit shared exponent, occupying an average of 8.5 bits per element, while providing the same dynamic range and precision as the standard BF16 type. BLOCKFP4 uses a 3-bit mantissa and an 8-bit shared exponent, resulting in 4.5 bits per element, yet offering a higher dynamic range and the same precision as FP8 E4M3.

As a result, by replacing the commonly used BF16 and FP8 types with BLOCKFP8 and BLOCKFP4, the same representational precision can be achieved while reducing memory footprint, memory bandwidth requirements, and Ethernet bandwidth requirements by nearly half. Our software provides the capability to execute various LLM inference and training workloads using the block floating-point format.

## Galaxy Server


Figure 6. A picture of the Galaxy server.

Galaxy is Tenstorrent's data center-class server product, equipped with 32 Wormhole processors. Its detailed specifications are shown in Table 3. A model equipped with 32 Blackhole processors is planned to be released.

Table 3. Specifications of the Tenstorrent Wormhole Galaxy server

| Items | Description |
|---|---|
| Form factor | 6U rackmount |
| Chassis dimensions | W 447 mm × H 266.7 mm × D 884.5 mm |
| Server weight | 119 kg |
| AI accelerators | **32x** Wormhole |
| CPU | **1x** AMD EPYC 9004 series with TDP ≤ 280 W |
| Main memory | 576 GB (**6x** 96 GB) DDR5-4800 ECC RDIMM |
| PCIe connections | **4x** PCIe Gen4 x8 lanes, **28x** PCIe Gen4 x1 lane |
| BMC management network | **1x** 1 GbE RJ-45 |
| Host network | **2x** 100 GbE QSFP-DD |
| Chip-to-chip network (internal) | **52x** 400 Gbps Ethernet 8×4 pre-connected |
| Chip-to-chip network (external) | **24x** 400 Gbps Ethernet QSFP-DD |
| Host storage | **2x** 960 GB NVMe M.2, **4x** 4 TB or 8 TB E1.S |
| Cooling | Air-cooling |
| Fan | **8x** hot-swappable 9276, **2x** hot-swappable 6056 |
| Power | 12 kW |


Figure 7. A side cross-sectional view of the Galaxy server.

A Galaxy server contains four AI accelerator modules, each equipped with eight Wormhole chips mounted on a UBB (Universal Baseboard). Each module has a height of 1.25U and can be replaced independently. The motherboard, CPU, and main memory occupy an additional 1U, bringing the total server height to 6U. The eight chips in each module are interconnected on the UBB in 4×2, while chips on different UBBs are connected through the server's backplane, with the remaining network interfaces at the edges of the mesh exposed as the server's 400 Gbps QSFP-DD ports, as shown in Figure 8.

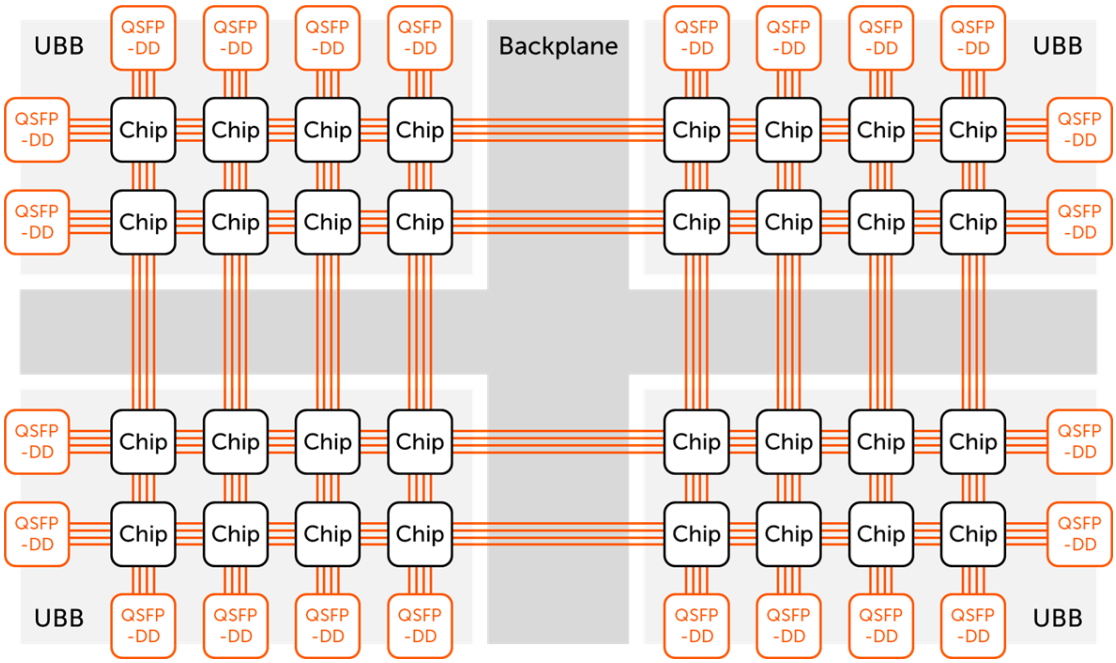


Figure 8. The pre-connected network topology inside the Galaxy server. Ultimately, the chips are interconnected in an 8×4 mesh and 24 QSFP-DD ports are exposed.

# Galaxy Cluster

## Inter-Chip Communication Mechanism

Wormhole has 16x 100 Gbps Ethernet interfaces, while Blackhole has 10x 400 Gbps Ethernet interfaces. Each interface basically operates in TT-Link mode, which implements the standard OSI Layer 1 and the Logical Link Control (LLC) sublayer of Layer 2. In addition, each interface is paired with a RISC-V-based Ethernet core that is also connected to the NoC and runs the TT-Fabric software stack, implementing the upper Layers 3 to 5.

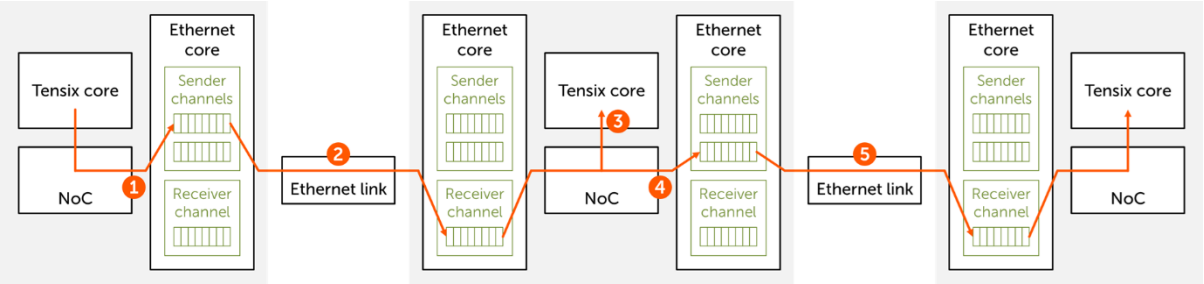


Figure 9. The process of chip-to-chip data transfer.

Figure 9 illustrates the actual process of data transfer between two chips. A Tensix core can push data into the TX queue (referred to as a *sender channels* in TT-Fabric) located in the SRAM of the Ethernet core via the NoC (①). The data is then transmitted to the neighboring chip via the direct link, and pushed into the RX queue (referred to as a *receiver channel* in TT-Fabric) of the receiving Ethernet core (②). The core determines whether the packet's destination is itself or another chip. If it is the destination, the packet is delivered via the NoC to its target – either GDDR memory or a Tensix core (③). If the destination is another chip, the packet is forwarded to the TX queue of the Ethernet core corresponding to the interface that leads to that chip (④). The packet is then sent onward to the next chip (⑤), repeating this process until it reaches its final destination. As will be described later, since the chips are connected in a torus network, packets can be efficiently delivered using simple static routing tables.

In the Wormhole chip, the 16 Ethernet interfaces are grouped into sets of four, each group connected to one of four neighboring chips. They cannot be connected to more than four chips individually. Similarly, in the Blackhole chip, eight of the ten Ethernet interfaces are paired to connect to four neighboring chips, while the remaining two interfaces can each connect to a different chip. That is:

- Wormhole: 16x 100 Gbps → exposed as 4x 400 Gbps
- Blackhole: 10x 400 Gbps → exposed as 4x 800 Gbps + 2x 400 Gbps

This is already reflected in the configuration of the Galaxy server. Load balancing among multiple physical links between two adjacent chips is handled by software.

Although this mechanism allows communication between distant chips, it is desirable to ensure that communication primarily occurs between adjacent or nearby chips to fully leverage the benefits of the Tenstorrent architecture. Fortunately, since Moreh's inference and training frameworks handle this automatically, users can run models and applications without concern for the underlying network topology.

## Inter-Chip Network Topology

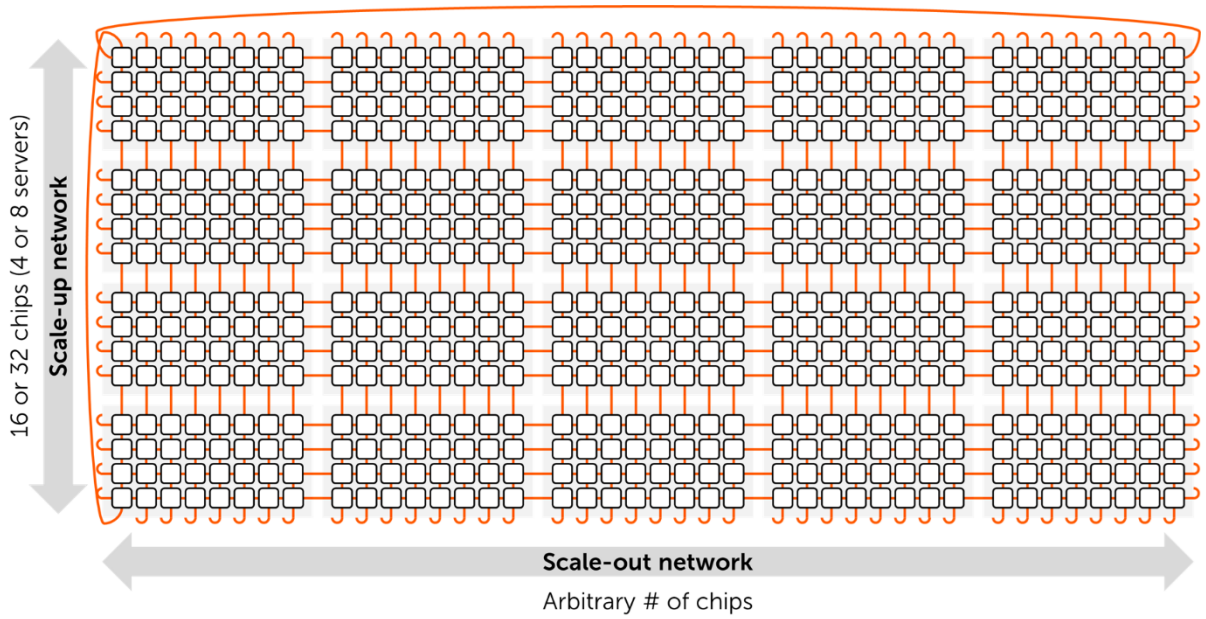


Figure 10. Example of a 2D torus network topology with 16 chips vertically and an arbitrary number horizontally.

We arrange Galaxy servers vertically in stacks of four or eight and horizontally in an arbitrary number to form a 2D torus network, as shown in Figure 10. Since each Galaxy server contains Wormhole chips arranged in 8×4, the resulting cluster of $N$ Wormhole chips is organized into a torus with 16 or 32 chips vertically and $N/16$ or $N/32$ chips horizontally, respectively. For example, a cluster of 8,192 chips (from 256 Galaxy servers) is interconnected in a 256×32 torus network. All chips are seamlessly interconnected without any distinction between connections inside and outside the server.

The vertical and horizontal networks have identical physical performance but serve different logical roles. The vertical network functions as a replacement for the NVLink interconnect in a GPU server (i.e., a scale-up network). All the 16 or 32 Wormhole chips in a column are connected in a bidirectional ring network with an aggregate bandwidth of 100 GB/s.

Considering the relative ratio between compute and communication performance, this allows for lower collective communication overhead compared to eight NVIDIA A100 GPUs connected via NVLink, which together form ring networks with an aggregate bandwidth of 300 GB/s. In addition, by leveraging the block floating-point format, we can reduce bandwidth requirements by half, while also benefitting from the low latency of CPU-free, chip-to-chip communication.

The horizontal network serves as a replacement for the InfiniBand interconnect between GPU servers (i.e., a scale-out network). In the vertical direction, it is assumed that all 16 or 32 chips participate in collective communication. However, in the horizontal direction, where a larger number of chips are connected, this assumption is not always valid. Nevertheless, multiple horizontally adjacent chips can still form a ring network with a bandwidth of 50 GB/s, which is significantly higher than the 25 GB/s achievable with InfiniBand HDR used in NVIDIA DGX A100 systems.

In the case of Blackhole, we can configure either a 3D torus or a 2D torus with a bypassing network, enabling scalable interconnection of up to one million chips. For even larger clusters, a torus-switch hybrid network is required, and the specific implementation details can be discussed with Moreh.

The torus network assumes that appropriate parallelization and task allocation are properly implemented to enable efficient collective and point-to-point communication. For example, the most fine-grained level of parallelization should be applied among the vertically aligned chips. This is achieved through our inference and training frameworks.

## Resource Allocation Across Jobs

In real-world data centers, it is rare for the entire cluster to be dedicated to a single user and a single job (a training job or an inference server). In practice, multiple jobs run concurrently, each assigned to a different set of chips.

Our basic unit of resource allocation is a column within the cluster. Each job is allocated to all 16 or 32 chips vertically, and an arbitrary number of consecutive chips horizontally, forming a 1D ring $\times$ 1D line. This ensures that every job can utilize a complete ring in at least one dimension, since the presence or absence of a loopback (i.e., ring vs line) results in a 2x difference in collective communication bandwidth. Figure 11 shows an example.

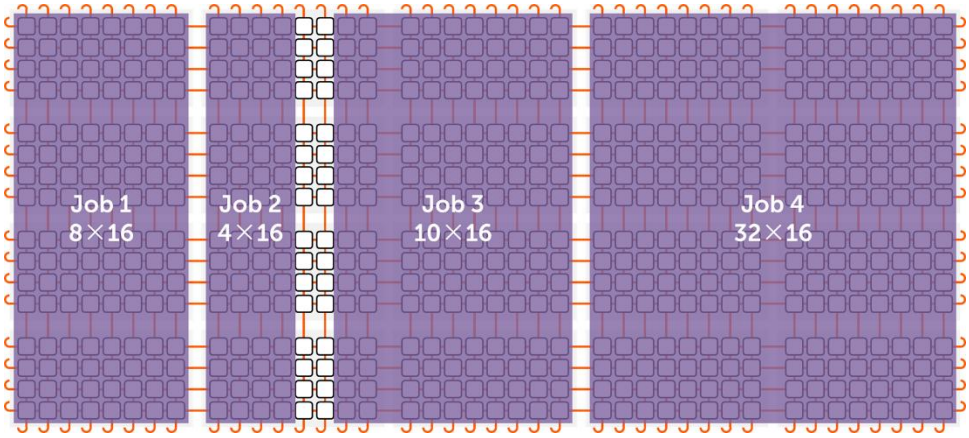


Figure 11. Example of four jobs in a cluster, each assigned to a different set of chips. Each job uses all 16 chips vertically, while the number of chips allocated horizontally varies.

### Host Network

Separate from the switchless torus network interconnecting multiple Tenstorrent chips, each Galaxy server includes a host CPU, and these hosts are interconnected via a 100 Gbps RoCE (RDMA over Converged Ethernet) host network. This network is used for the following purposes:

- Remote access to individual servers
- Kubernetes control and data-plane networking
- Access to a distributed file system – for example, training data are transferred to each server via the host network and then delivered to the chips through PCIe
- Transmission of inference requests (e.g., input sequences) and return of responses (e.g., output sequences)
- Control of multiple servers and chips by the runtime systems of our inference and training frameworks

## Software Stack

### TT-Metalium

TT-Metalium is a low-level programming model for Tenstorrent chips and clusters, serving as the bridge between our higher-level software (frameworks and libraries) and Tenstorrent hardware. In particular, the following components are primarily used in our software stack:

- A runtime system for controlling each chip and transferring data between the host and the chips

- A compiler and low-level library (TT-LLK) for writing kernels executed on the baby RISC-V sub-cores within each Tensix core
- The TT-Fabric software stack for enabling inter-chip Ethernet communication

TT-Metalium is an open-source project, and Moreh is one of its largest external contributors. By continuously improving TT-Metalium, we aim to achieve seamless integration and end-to-end optimization between our software and Tenstorrent hardware.

## Tensor Operation Library

We provide optimized implementations for a wide range of tensor operations required for both inference and training of generative AI models. These serve as the fundamental building blocks of our inference and training frameworks. Although Tenstorrent also offers its own operation library, TT-NN, we have developed a separate library with the following objectives:

- To support a broader range of operations, ensuring compatibility with various PyTorch-based models – particularly by covering diverse backward operations needed for training.
- To ensure consistent behavior and high performance across diverse input/output patterns (e.g., tensor shapes) and operation variants. Operations in TT-NN are often optimized and tested only for specific cases.
- To support various tensor layout combinations, thereby enabling the compiler to apply optimized data placement and tiling schemes.
- To support multiple numerical precisions, and especially to provide numerically stable implementations that ensure training convergence comparable to that of GPUs.
- To provide additional specialized operations (e.g., retiling) required by our inference and training frameworks.

## Communication Library

Our inference and training framework also incorporates built-in collective and point-to-point communication libraries. These libraries are developed as extensions of Tenstorrent's CCL library and provide the following capabilities:

- Deliver high performance across a wide range of data size, chip counts, and mesh dimensions.
- Minimize end-to-end latency through *spatial fusion*, that is, by executing communication concurrently with the preceding and succeeding tensor operations on different Tensix cores within a chip.
- Minimize the number of Tensix cores dedicated to communication tasks.

- Support emerging communication patterns used for model disaggregation and KV cache transfer in distributed inference (e.g., dispatch and combine), in addition to conventional collective communication.

## Moreh vLLM

*Moreh vLLM* is our optimized version of vLLM designed to efficiently run various open-source or even proprietary AI models on Tenstorrent Galaxy systems. Build upon our tensor operation and communication libraries, it consistently delivers strong performance across diverse model architectures and sizes, input/output lengths, concurrency levels, precisions, generation methods, the number of chips, and other configurations. It continuously stays up to date with the latest version of open-source vLLM and expands its model support.

Moreh vLLM can be used either as a standalone inference server or as the backend for the MoAI Inference Framework described later. For the latter use case, Moreh vLLM is optimized not only to run full model inference but also to execute partial inference tasks, such as prefill-only, decode-only, or expert-specific execution.

## Inference Framework

The *MoAI Inference Framework* is a distributed inference framework designed for cluster systems and Kubernetes environments. It enables users to deploy inference API endpoints for various generative AI models on Galaxy clusters at different scales. Simply put, it can be regarded as a solution analogous to llm-d or Dynamo for NVIDIA GPU clusters, though MoAI provides additional automation capabilities beyond those.

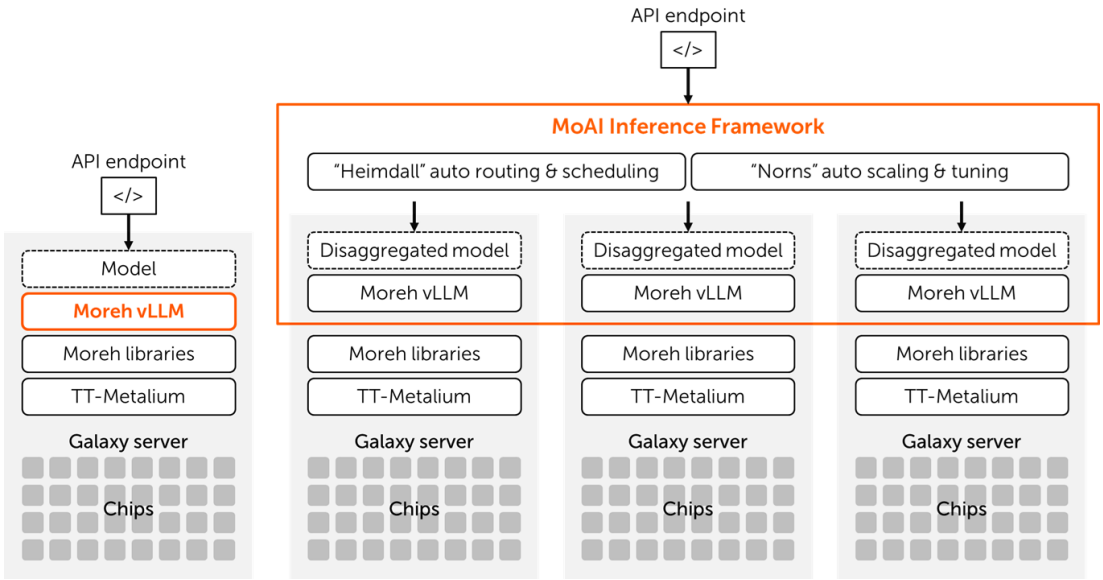


Figure 12. Inference software stack including Moreh vLLM running as a standalone server and the MoAI Inference Framework for distributed inference.

It runs multiple Moreh vLLM instances concurrently on the cluster and distributes incoming requests among them. It also incorporates various distributed inference techniques – such as prefill-decode disaggregation, expert parallelism, and prefix-cache-aware routing. Furthermore, it can identify, apply, and dynamically adjust the optimal combination of these techniques, as well as the appropriate scaling, to meet specific service level objectives (SLOs).

## Training Framework

The *MoAI Training Framework* is a PyTorch-compatible framework that presents a Galaxy cluster as a single virtual device. That means, users can write their models and training scripts as if only one large and powerful PyTorch device exists, while the framework automatically parallelizes and optimizes execution across multiple Tenstorrent chips within the cluster.

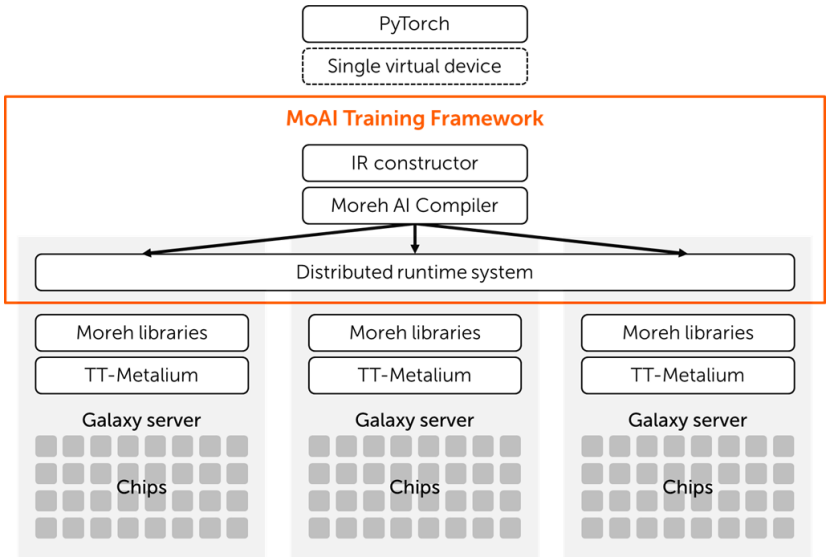


Figure 13. Training software stack providing a single virtual PyTorch device.

Users can use existing model implementations (such as those included in Hugging Face Transformers) and write their own training algorithms in PyTorch. The framework preserves PyTorch's eager programming model while internally generating a graph-level intermediate representation (IR), transforming it through a graph-to-graph compiler called the *Moreh AI Compiler*, and executing it via a distributed runtime system (similar to LazyTensor for Google TPUs). The Moreh AI Compiler performs not only parallelization, but also a range of optimizations to achieve peak performance on Tenstorrent chips, including automatic tiling, SRAM allocation, and spatial fusion.

## Conclusion

We combine Tenstorrent's lightweight and scalable hardware with our proprietary software stack to deliver an efficient and flexible solution for large-scale AI data centers. This approach minimizes total cost of ownership (TCO), provides an alternative to dominant hardware vendors, and supports a wide range of AI workloads from inference to training.

This is not merely a replica of a GPU cluster; it is made possible through differentiation and vertical integration across the entire stack, from unique core and NoC architectures to a low-latency flat torus network, optimized libraries, and frameworks that enable efficient and seamless orchestration of large numbers of chips.

## Disclaimer

The information presented in this article pertains to Moreh's integrated solution, which combines Tenstorrent hardware with our cluster architecture and software, and may differ from information provided by Tenstorrent.

This article includes registered trademarks of Tenstorrent. Please refer to the page at the following URL for a list of Tenstorrent's trademarks:
https://tenstorrent.com/trademarks

MOREH

To learn more, please visit our website (https://moreh.io) or contact us (contact@moreh.io).