Technical Report

21K Output Tokens Per Second DeepSeek Inference on AMD Instinct MI300X GPUs with Expert Parallelism

Moreh, Inc.

November 2025

MOREH

Contents

21K Output Tokens Per Second DeepSeek Inference on AMD Instinct MI300X GPUs with Expert Parallelism

Overview

One of the major breakthroughs in large language models (LLMs) is the adoption of the Mixture-of-Experts (MoE) architecture. Instead of a single large feed-forward network (FFN) layer, the model contains multiple smaller FFN layers, each referred to as an *expert*. For each token, a lightweight *gating network* dynamically selects only a small subset of experts to process each step. This conditional computation allows MoE models to scale to extremely large parameter counts while maintaining efficient compute usage. For example, the DeepSeek-R1 671B model that caused quite a stir when it was released in January 2025 and OpenAI's popular open-source GPT-OSS 120B model that was released in August 2025 both use the MoE architecture.

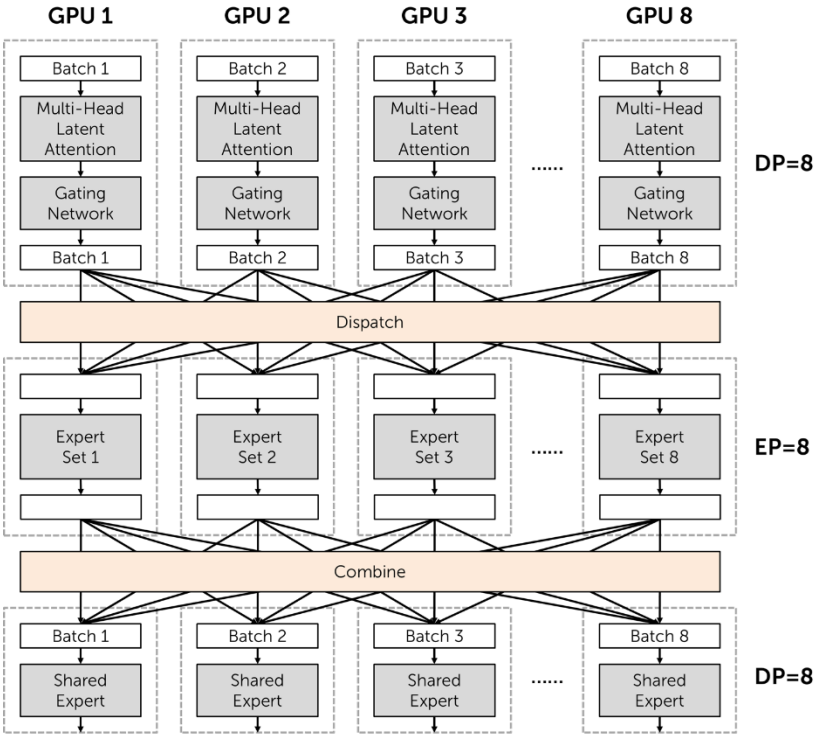
However, due to the large size and sparse design, models like DeepSeek-R1 require advanced optimization techniques to serve efficiently at scale. In particular, we can achieve high throughput (i.e., total output tokens per second) by applying expert parallelism (EP). This is because (1) only a subset of experts is stored in GPU memory, allowing for a larger batch size, and (2) the parameters of individual experts can be reused more once they are loaded, thereby improving FLOPS per byte. However, challenges arise in efficiently implementing the fine-grained communication pattern known as dispatch/combine — which transfers activations to the GPUs responsible for the activated experts — and in resolving the imbalance among experts.

Moreh, a software partner of AMD, recently demonstrated that DeepSeek-R1 inference can be executed at high throughput by implementing EP on the ROCm software stack. To actually attain such high performance in LLM inference, carefully optimized software is required across multiple layers of the stack. The Moreh team applies various optimizations at both the library and model implementation levels. As a result, they achieved a decoding throughput of >21,000 tokens/sec on a server equipped with 8x AMD Instinct MI300X GPUs.

Parallelization Method

In the DeepSeek R1 model, each decoder block contains 256 experts, among which 8 are activated for each token. In addition, there is a shared expert that is always executed regardless of routing, and a Multi-Head Latent Attention (MLA) layer, a new attention mechanism introduced by DeepSeek.

EP is typically implemented in combination with data parallelism (DP). Each GPU holds and executes a distinct subset of experts. Meanwhile, for components that are always executed — such as the shared expert and the MLA — each GPU processes a separate batch of requests in parallel. Accordingly, we applied a parallelization configuration of DP=8 and EP=8 per node. That is, each GPU is responsible for 32 experts.



Furthermore, applying prefill-decode disaggregation — that is, executing the prefill and decode stages on separate nodes — can further increase the overall throughput of the cluster system. The prefill stage processes the entire input sequence at once and is more compute-bound, whereas the decode stage generates the output tokens one by one in an autoregressive manner and is more memory-bound. By not only separating the execution but also tailoring parallelization and optimization configurations to the distinct computational characteristics of prefill and decode, GPU efficiency can be further improved. In this article, we focus our evaluation primarily on the performance of the decode stage, since in typical applications the majority of GPUs are devoted to decoding.

Performance Optimizations

Simply implementing a new parallelization method does not automatically yield the best performance. To fully unlock the hardware's potential, the entire software stack from libraries to model implementation must be optimized to accommodate the new computation and communication patterns. The Moreh team implemented several optimizations based on vLLM, as described below.

Load Balancing Between GPUs

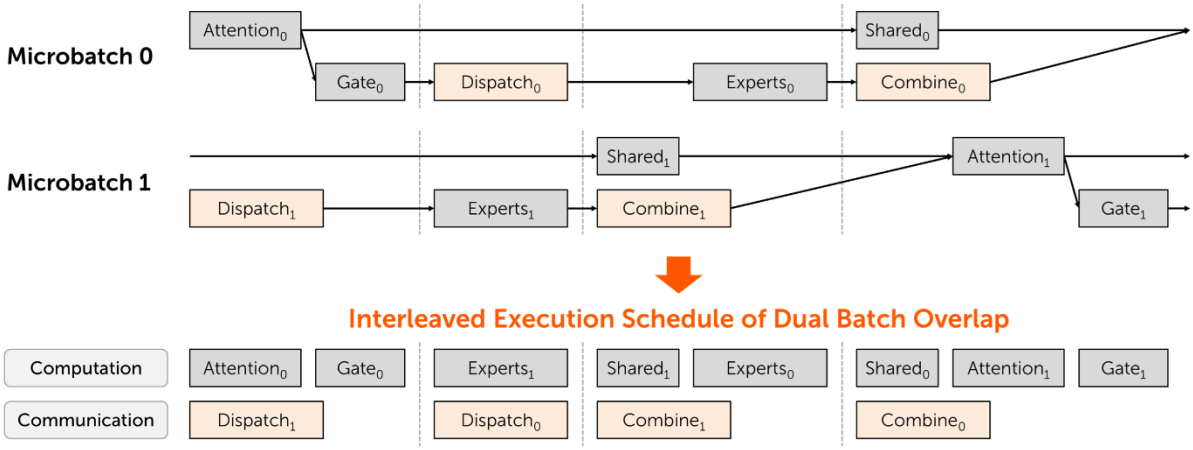
One of the major challenges in implementing EP is that the workload is not evenly distributed across GPUs and varies depending on the routing results. Although MoE models are trained to minimize this imbalance as much as possible, there are still inherent limitations. DeepSeek also acknowledged this issue and reported using their own load-balancing method called EPLB to address it.

Without applying any explicit load-balancing strategy, we observed up to a 2x workload imbalance across GPUs when experts were simply divided into eight consecutive chunks — that is, assigning experts 0-31 to the first GPU, 32-63 to the second, ..., and 224-255 to the last. We designed an algorithm that measures the activation frequency of each expert, and groups the 256 experts into eight sets of 32 such that the total activation frequency of each set is as balanced as possible. (This can be seen as a variant of the balanced number partitioning problem.) The algorithm then reorders the experts so that each set is placed contiguously and applies EP on that. Since the activation frequency of experts varies across decoder blocks, this process is performed separately for each decoder. The gating network must also produce routing results that reflect the new expert permutation.

As a result, we successfully reduced the workload imbalance across GPUs to within 5%. To the best of our knowledge, this is the first EP load balancing implementation on AMD GPUs. In real-world serving scenarios, continuous load balancing can be achieved by periodically re-measuring the activation frequencies and generating new expert permutations accordingly.

Computation-Communication Overlapping

Another challenge in applying EP is minimizing the overhead caused by dispatch/combine all-to-all communications. Dividing the batch into two microbatches and executing them concurrently allows the communication of one microbatch to overlap with the computation of the other. When the batch size is sufficiently large — that is, in high-throughput scenarios — this approach can effectively hide communication latency without sacrificing computational performance.



There are several implementation variations of this approach, among which we adopted vLLM's DBO (Dual Batch Overlap) system. We modified the DBO system to use MORI-EP, and in particular, we enhanced the MORI-EP library so that its communication operations can be invoked from the two worker threads of DBO and are properly synchronized with other computations.

Library Optimizations

The Moreh team has also applied several library optimizations to maximize GPU efficiency and improve both throughput (tokens/sec) and latency (time to first token and inter-token latency). Here are selective examples of optimizations.

- **Optimal GEMM and Attention Kernel Selection:** dynamically selects the optimal GEMM and Attention kernels without the need for online profiling and manual tuning.
- **Fused MoE Kernel Optimization:** implements a highly optimized fused MoE kernel that delivers better performance than AMD's AITER library, particularly for small batch sizes.
- **FP8 KV Cache Support:** implements Multi-head Latent Attention (MLA) kernels that enable the KV cache to be stored and loaded in FP8 format, to improve performance especially in long-context scenarios.
- **Vertical and Horizontal Kernel Fusion:** employs both vertical fusion (e.g., fused RoPE kernels) and horizontal fusion (e.g., merging multiple GEMMs in shared experts) to reduce kernel overhead and improve computational efficiency.

For details on these library-level optimizations and their performance improvements independent of EP, please refer to Moreh's technical report.

Leveraging HIP Graphs

HIP Graphs is a technique that reduces CPU runtime overhead by capturing multiple GPU operations into a static graph and issuing them all at once. This is particularly

essential in the decode stage, where individual operations are relatively short, making it critical to fully utilize the GPU. However, constructing a static graph requires that the tensor sizes passed between operations should be fixed. In EP, however, the input size for each expert is inherently dynamic and determined by the routing results, making it highly challenging to represent this as a static graph.

We modified the DeepSeek model implementation in vLLM to make tensor sizes as static as possible while still supporting EP. This enabled us to take full advantage of HIP Graph, thereby minimizing CPU overhead. Incorporating MoRI library operations into the graph posed several technical challenges, but we finally succeeded in including MORI-EP's dispatch/combine operations as part of the graph execution.

Experimental Method

Our experiments were conducted on an MI300X server with the following specifications.

- CPU: 2x AMD EPYC 9474F 48-core 3.6 GHz
- Main memory: 2,304 GB
- GPU: 8x AMD Instinct MI300X OAM GPU 192 GB
- Server: Gigabyte G593-ZX1-AAX1
- Operating system: Ubuntu 22.04.4 LTS
- ROCm version: 6.4.1

We designed the evaluation methodology by referring to the LMSys blog post. In that post, the key metrics for evaluating EP performance — assuming prefill/decode disaggregation — are the total throughput (tokens/sec) per decode node and the inter-token latency. While LMSys implemented prefill/decode disaggregation across 12 nodes, we aimed to measure decode performance even in a single-node environment. To achieve this, we made several modifications to vLLM as follows:

- On the server side, we excluded the prefill time and accumulated only the decode time and token count to measure decode throughput separately. Inter-token latency can still be measured on the client side using the standard benchmarking approach.
- The scheduler of the vLLM V1 engine originally executes prefill and decode requests in a single batch (referred to as mixed prefill), making it impossible to exclude the prefill time. We modified it so that prefill and decode requests are issued to separate queues and executed independently.

The following command can be used to send requests to the (modified) vLLM server and measure performance.

```
vllm bench serve \
    --backend vllm \
```

```
    --model "deepseek-ai/DeepSeek-R1" \
    --metric-percentiles "10,25,50,75,90" \
    --percentile-metrics "itl,tps,ttft,e2el" \
    --port 8001 \
    --num-prompts 2048 \
    --max-concurrency 2048 \
    --ignore-eos \
    --dataset-name sharegpt \
    --dataset-path /app/dataset/ShareGPT_V3_unfiltered_cleaned_split.json \
    --sharegpt-input-len 2000 \
    --sharegpt-output-len 100
```

Experimental Results

We measured the throughput (per node) and inter-token latency for four different concurrency levels (1024, 2048, 3072, and 4096) to observe the trade-off between latency and throughput. The results are as follows.

Table 1. Experimental results

Concurrency	Throughput (tokens/sec)	Inter-token latency (ms)					
		P10	P25	P50 (median)	P75	P90	Mean
1024	10,030.0	49.49	100.15	102.69	104.94	112.26	102.94
2048	18,348.6	81.22	87.85	91.48	96.06	126.70	116.92
3072	19,412.7	105.52	121.03	145.08	167.47	229.79	166.10
4096	21,224.6	145.13	158.41	160.75	167.25	216.61	197.13

We achieved a throughput of 21,224.6 tokens/sec at the highest concurrency level. This is nearly equivalent to the 22,282 tokens/sec reported by the SGLang team on an 8x NVIDIA H100 GPU server, confirming that our implementation delivers state-of-the-art EP performance. Meanwhile, even in the configuration optimized for minimal latency (with a median inter-token latency of 91.48 ms), throughput decreased by less than 15%, demonstrating that the system can flexibly adjust its maximum concurrency according to the target SLOs (Service Level Objectives).

Conclusion

Moreh successfully implemented high-throughput inference with EP on AMD Instinct MI300 series GPUs by applying and optimizing a variety of techniques for MoE models. We also provides the MoAI Inference Framework, which automatically applies various distributed inference techniques including the efficient EP

implementation explained in this article, on AMD Instinct GPU clusters. For more information, please visit the Moreh website.

MOREH

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