

JUL 23, 2024

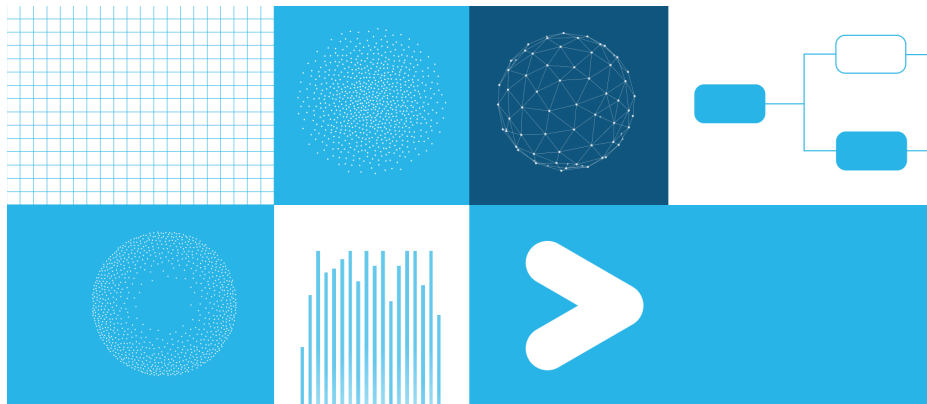
AUTHOR

Snowflake AI Research

SHARE

Achieve Low-Latency and High-Throughput Inference with Meta's Llama 3.1 405B using Snowflake's Optimized AI Stack

Gen AI



Meta's Llama 3.1 405B represents a groundbreaking milestone for open-weight large language models (LLMs), pushing the boundaries of what's possible in AI by bringing frontier model capabilities to everyone. However, the deployment and fine-tuning of such massive models come with a significant set of challenges:

Multi-node deployments. First, with more than 400 billion parameters, Llama 3.1 405B cannot fit on a single node with 8 x H100 80GB GPUs, even when using half-precision. This necessitates low-precision deployments, deploying on multiple nodes or both. Deploying on multiple nodes is not always accessible for everyone due to the cost of such infrastructure and the expertise needed to manage it.

Solving the speed vs. throughput challenge. Second, the massive compute and memory requirements make it difficult to achieve low latency for real-time use cases while sustaining high throughput for cost-effectiveness.

Managing maximal context lengths. Third, even though Llama 3.1 405B is trained to support long context lengths up to 128k, the open source framework support for actually running it is nascent.

AUTHOR

Snowflake AI Research

SHARE

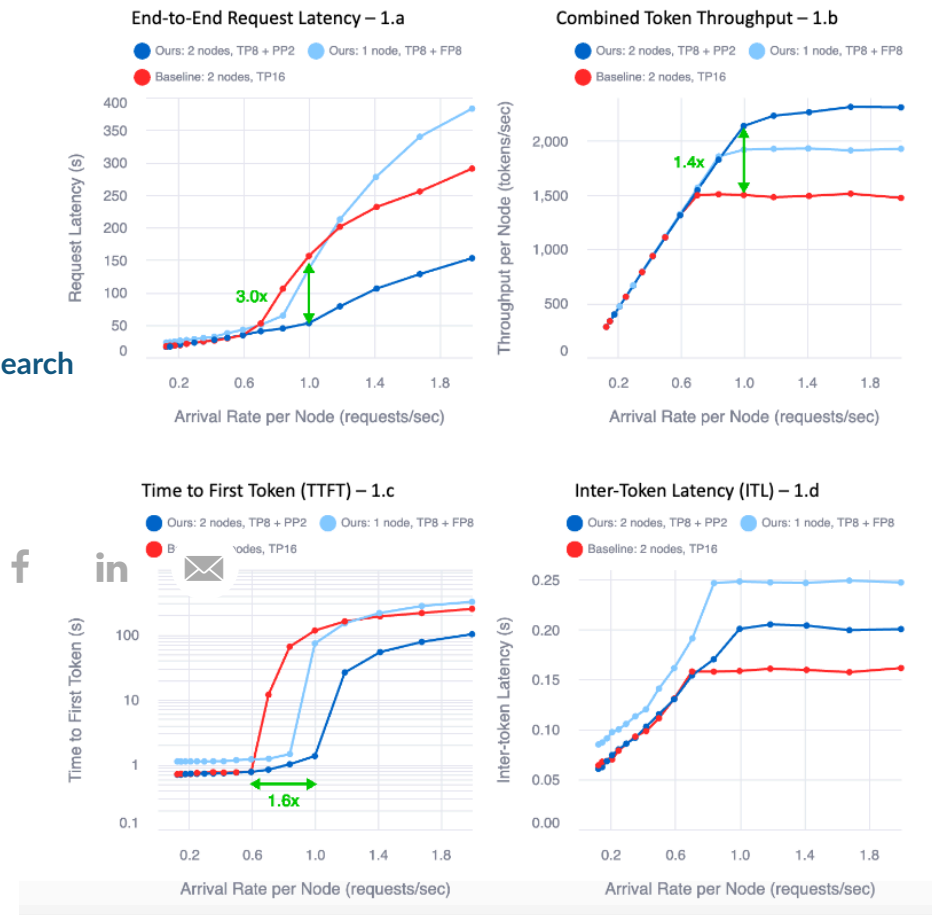
Snowflake AI Research has been proactively addressing these challenges by releasing our own open massive language model, Snowflake **Arctic**, in April 2024. Together with the OSS AI community, our work has been focused on building system solutions that enable efficient and cost-effective inference and fine-tuning of massive multi-hundred-billion-parameter models.

Snowflake Massive LLM Inference System Optimization Stack Minimizes Latency, while Maximizing Throughput and Arrival Rate per Node

AUTHOR

Snowflake AI Research

SHARE



Figures 1a-d. Arrival rate vs Latency (1.a), Throughput (1.b), Time to First Token (1.c), and Inter Token Latency (1.d) for Llama 3.1 405B for common Snowflake enterprise workload patterns (Input prompt: 2K; output: 256). Compared to baseline, our inference stack supports real-time inference¹ with up to 3x lower end-to-end latency and more than 1.4x higher token throughput (Fig 1.a and 1.b), and it supports 1.6x higher arrival rate for real-time inference (Fig. 1.c). At lower traffic, it can achieve TTFT (time to first token) in under 750 ms and ITL (inter-token latency) in under 70 ms.

To empower the broader AI community to efficiently use massive language models like Llama 3.1 405B, we are thrilled to make the following two releases:

1. Massive LLM Inference System Optimization Stack: Our

optimization stack combines hardware-agnostic FP8 ZeroQuant with highly optimized implementation of tensor and pipeline parallelism.

Low latency with high throughput. It simultaneously enables both low latency for real-time inference and high-throughput for cost-effective serving for Llama 3.1 405B,

with up to 3x lower latency and 1.4x higher token throughput than baseline (Fig. 1a, 1b).

Fast response for latency critical workloads. For latency critical workloads, it can achieve low latency of under 70ms per token generation and time to first token under 750 ms (Fig. 1).

AUTHOR

Snowflake AI Research

High throughput even on a single node. It supports single-node inference of Llama 3.1 405B on both legacy (A100) and current hardware (H100), while still achieving 1.25x higher throughput per node over baseline (Fig. 1).

SHARE



Maximum context length support. It supports a full context window of 128K for Llama 3.1 405B while achieving 1.25x higher throughput compared to baseline (Fig. 6).

- 2. Massive LLM Fine-Tuning System Optimization Stack:** This enables fine-tuning of the massive Llama 3.1 405B model on just a single node by combining hardware-agnostic FP8 ZeroQuant with highly optimized implementation of ZeRO-3 with CPU off-loading and LoRA.

With today's release, Snowflake AI Research, in collaboration with the open source AI community, establishes a new State-of-the-Art (SoTA) for open source inference and fine-tuning systems for multi-hundred-billion-parameter LLMs. Both optimization stacks have been upstreamed to vLLM and DeepSpeed, and are easily accessible via [GitHub repository](#).

We have also partnered with Meta to bring Llama 3.1 405B, 70B and 8B to Snowflake Cortex AI. All three Llama 3.1 models are generally available, via **COMPLETE function** with availability across various **cloud regions**. We are investing heavily in making fine-tuning easier with Llama 3.1 405B, with public preview coming soon for our **Cortex Fine-Tuning** customers, empowering the AI community to leverage and build on top of this powerful model.

In the remainder of this blog we will do a deep dive on the inference optimization stack, for more details about fine-tuning visit [this blog](#).

Massive LLM Inference System Optimization

Overview

AUTHOR

Snowflake AI Research

Real-time and efficient serving of massive LLMs, like Meta's Llama 3.1 405B, has three key requirements:

i) sufficient memory to accommodate the model parameters and the KV caches during inference;

SHARE

f a l i n e i ✉,h batch size to achieve good hardware efficiency; and

iii) adequate aggregate memory bandwidth and compute to achieve low latency.

While a basic multi-node solution using tensor parallelism can meet the demand for aggregate memory, memory bandwidth and compute — as well as support large batch sizes — it often incurs significant communication overhead, due to the large amount of data that needs to be shared across nodes combined with slow interconnect across nodes. To address those challenges, we developed four key optimization techniques:

Hardware-agnostic **FP8 quantization** and dequantization

Tensor parallelism within each node to aggregate memory, memory bandwidth and compute for low latency

Pipeline parallelism across multiple nodes to aggregate more memory to support high throughput and longer context

Dynamic SplitFuse to enable large batch per each forward pass

FP8 Quantization. We leverage hardware-agnostic FP8 quantization to reduce the memory required to store the model parameters. This technique, combined with 8-way tensor parallelism within a node, aggregates enough GPU memory to accommodate the model parameters and KV caches required during inference.

Tensor Parallelism. Using FP8 and 8-way tensor parallelism offers sufficient aggregate memory bandwidth and compute throughput to achieve low latency for real-time inference. However, this combination still does not provide enough aggregate memory to support the large batch size needed for high-throughput or the extensive context windows that enterprise workloads demand.

AUTHOR

Snowflake AI Research

Pipeline Parallelism. To address aggregate memory shortage, we combine tensor parallelism with pipeline parallelism, which has a very low communication overhead. This two-dimensional (2D) parallelism approach allows us to scale inference across multiple nodes, leveraging additional aggregate GPU memory to support

SHARE

f gh in ro ✉ out and larger context windows.

SplitFuse Scheduling. The Dynamic SplitFuse scheduling technique combines prefill and sampling together within a single batch. This allows for consistently large batch size during each forward pass, eliminating pipeline bubbles and improving compute efficiency.

We developed this system in collaboration with the vLLM OSS community, including contributions from Murali Andoorvedu @ CentML, UC Berkeley, UCSD and Anyscale. All of our optimizations have been upstreamed to benefit the broader community. In the following sections, we discuss our optimizations in detail and present a thorough evaluation of our system for massive LLM inference.

Hardware-Agnostic Quantization and Dequantization

Quantization primarily aims to reduce the memory footprint of large-scale models, enabling them to be deployed on fewer devices. However, this process introduces challenges regarding system performance, particularly in efficiently executing matrix multiplications (GeMM) when inputs are in half-precision formats (BF16 or FP16) and weights are quantized to FP8. We can take two approaches to perform the GeMM using FP8 weights:

1. FP8-format GeMM by quantizing the activation input.

2. Run GeMM in high-precision by dequantizing the weight and keeping the activation in the original format.

Even though the first approach doubles the GeMM throughput, it comes with specific hardware requirements — such as FP8 tensor cores, which are provided by some hardware such as NVIDIA H100 — whereas the latter is a more generic solution supported by many hardware. On the other hand, the second way of

AUTHOR

Snowflake AI Research

running GeMM maintains model quality due to preserving the input's precision and using higher-precision MMA (tensor cores), while the first one may lose some accuracy due to quantizing input and using the FP8, lower-precision MMA cores. Here, we take the latter approach, as it provides a more generic solution with large adoption and maintains model quality.

SHARE

In terms of the overhead compared to pure BFP16/FP16 GeMM, both methods introduce either quantization or dequantization overhead. Regarding the dequantization, we can efficiently fuse it with the GeMM and dequantize the weight on-the-fly. This requires careful implementation to fuse dequantization with GeMM to maximize the teraflops achieved during these operations. In this section, we explore the performance challenges and implementation strategies necessary to make on-the-fly dequantization a viable solution, while minimizing resource usage for large-model deployment.

Quantization Format and Scale

For our quantization strategy, we adopt the widely used float8 format, consisting of a 4-bit exponent and a 3.1-bit mantissa. To enhance accuracy, we implement groupwise quantization scaling, which effectively maps higher-precision values to ranges that can be more accurately represented in the FP8 format. Since quantization occurs offline (during checkpoint loading) and does not affect the model's inference-critical path, we don't focus on optimizing this part. Conversely, the performance of GeMM is heavily dependent on the speed of the dequantization process.

Dequantization Process

Accurate dequantization involves three key operations:

1. **Masking:** Extract the different components of a floating-point number — the exponent, mantissa and sign.
2. **Reconstruction:** Reassemble the higher-precision value using the extracted components, including checks for NaN/inf values and adjusting the exponent with bias.
3. **Scaling:** Adjust the dequantized data to map from the lower-precision range back to a higher-precision range.

AUTHOR

Snowflake AI Research

SHARE

To simplify the dequantization process, we precompute masks and fixed-point differences for the exponent between BF16/FP16 and FP8 formats. We also skip checks for NaN/inf, assuming that weights do not contain such values. Consequently, we can convert from FP8 to FP16/BF16 using just four bitwise operations (AND and shift) along with a single addition. The figure below illustrates the relationship between the components of the two formats and details the conversion process. Ultimately, the only computationally expensive operation left is the scaling of the converted data, which requires a single-precision multiplication.

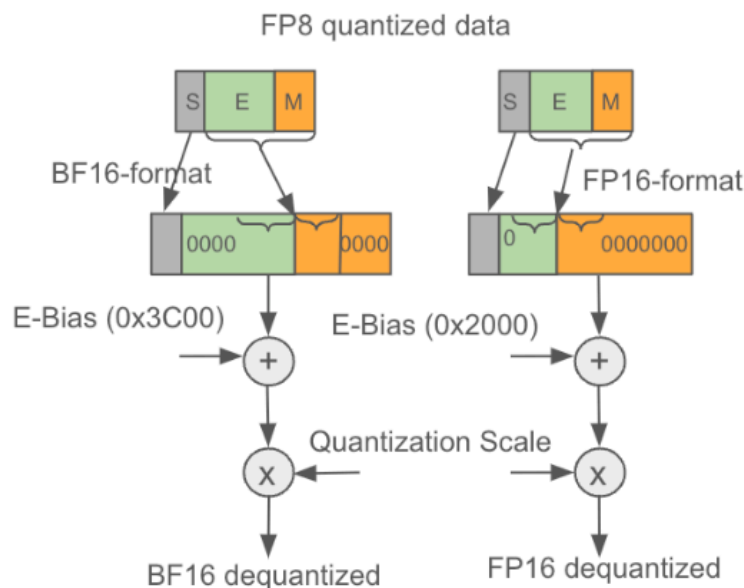


Figure 2. FP8 dequantization: 1) mapping the exponent and mantissa data to the right location based on FP16/BF16 formats; 2) adding the exponent-bias; 3) scale the data back using the quantization scale.

To further enhance GeMM performance, we use the loop unrolling technique to prefetch the scale and weight matrices. This approach allows us to overlap the computation of the current

MMA operation with data fetching for the subsequent block. We utilize double-buffered memory for loading weights in a ping-pong manner, resulting in an approximately 15% performance improvement.

Implementation Details and Hardware Independence

AUTHOR

Snowflake AI Research

SHARE

For the implementation of our quantization scheme, we utilize the Triton programming language, which facilitates a high-level description of GeMM and related operations. To optimize performance across various GeMM shapes, we conduct a sweep of different sizes (block_m, block_n and block_k) to identify the configuration that maximizes GeMM throughput. These configurations are then combined to select optimal settings for each GeMM during runtime. It is important to note that we refrain from using Triton's auto-tuning feature during runtime due to its potential cost, especially given the dynamic nature of inference, where input lengths can vary significantly.

By employing FP16/BF16 tensor-core operations, our FP8-fused GeMM kernel is compatible with a range of hardware platforms, including NVIDIA V100, A100 and H100. Additionally, we include the number of warps as a configurable parameter, allowing for kernel optimization across different hardware SKUs, such as AMD and Intel.

Tensor Parallelism

Tensor parallelism splits both model parameters and computation across multiple GPUs, allowing it to leverage both the parallel computation powers and the aggregate bandwidth from all GPUs in parallel for low latency. However, it requires significant communication bandwidth between participating GPUs. Within a node, the NVSWITCH offers the necessary bandwidth to reduce the communication overhead associated with tensor parallelism. However, as we go across nodes, the slow interconnect bandwidth results in massive communication overhead. Therefore, we limit the use of tensor parallelism to within a node.

Pipeline Parallelism

Llama 3.1 405B has 405 billion parameters, requiring roughly 800 GB memory to be served in its original BF16 precision, exceeding the total GPU memory capacity of a single AWS P4 or P5 instance with 8 x 80GB A100/H100 (640GB memory capacity). Even in FP8 precision, it requires 405 GB of memory, leaving limited space for KV caches, which limits the batch size and context window we can run with.

AUTHOR

Snowflake AI Research

Therefore, to leverage the aggregate GPU memory across multiple nodes, we use pipeline parallelism, as it has a much lower communication overhead compared to tensor parallelism. For instance, we can leverage 2D parallelism – 2-way pipeline parallelism and 8-way tensor parallelism – on two nodes.

SHARE

Specifically, we first partition the model equally into two parts by separating its layers and placing half on each node; the two nodes only transfer layer activations at the partitioned boundary in order to minimize internode communication. We then apply 8-way tensor parallelism inside each node, leveraging the high-bandwidth NVLink inside a node, to minimize request latency. Fig. 3 illustrates this parallelism architecture.

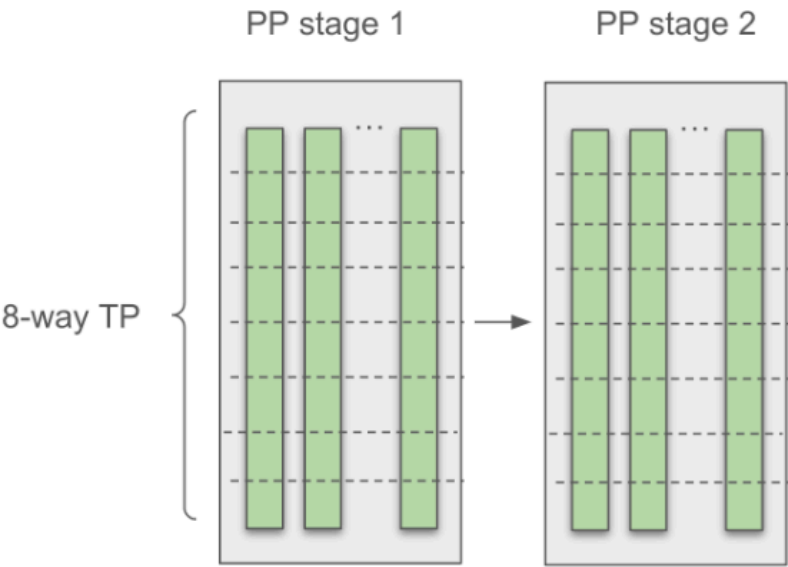


Figure 3. Two stage pipeline parallelism combined with 8-way tensor parallelism.

Communication optimizations for Pipeline Parallelism





In addition to the vanilla pipeline parallelism, we have also implemented several additional communication optimizations:

Early-fusing residual and hidden states. The original Llama model architecture passes two intermediate states between adjacent layers: hidden states and residual. Due to pipeline parallelism partitioning, this requires transfer of two tensors of the same size ($\text{batch_size} \times \text{hidden size}$) at the pipeline layer boundary between two nodes, which causes communication latency. We early-fuse these two tensors before communicating them between boundaries, which reduces the communication volume by half.

AUTHOR

Snowflake AI Research

SHARE

    **Further communication.** When applying 8-way tensor parallelism together with pipeline parallelism, each GPU out of an 8-GPU tensor parallel group needs to communicate with its next rank in its pipeline parallel group via the slow internode connections. Since oftentimes the internode connections have limited bandwidth — as compared to fast, intranode NVLink — we apply a scatter-allgather optimization, originally developed in DeepSpeed, Megatron-LM and Alpa, to account for a potential bottleneck: We let each of the two GPUs in a pipeline parallel group to only send a $\frac{1}{8}$ slice of the activation, via the slower internode connect, and let all GPUs within a node perform an allgather in the destination tensor parallel group to recover the full tensor. This reduces the internode communication further by $\frac{7}{8}$. Since the allgather happens now via the high-bandwidth NVLink, it has negligible overheads.

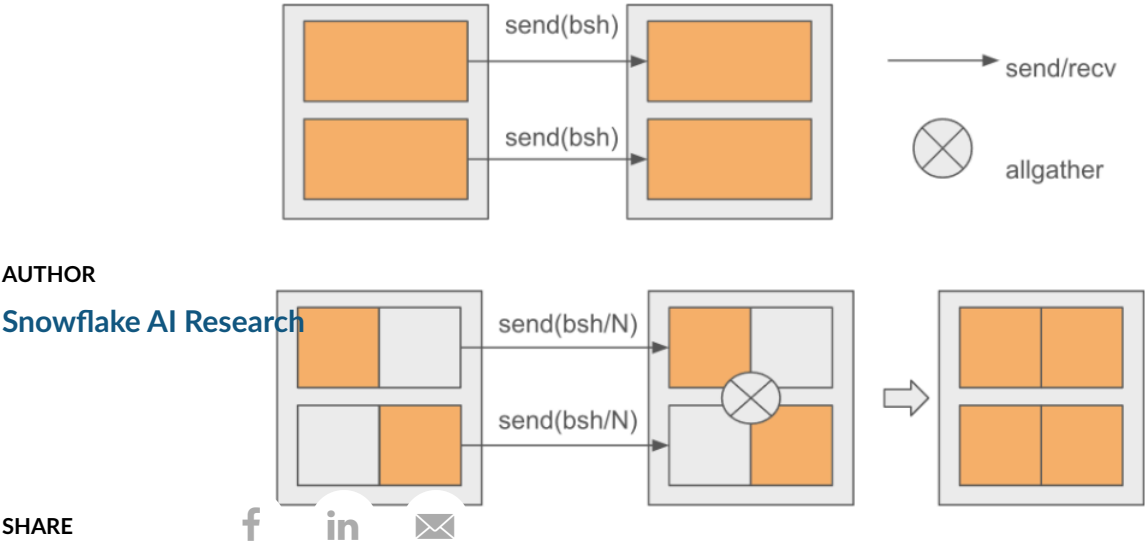


Figure 4. Scatter and allgather communication optimization to reduce pipeline communication across nodes.

SplitFuse

LLM inference uniquely has two phases of computation: prefill and decoding. These two phases feature very distinct computational characteristics – a prefill often takes much longer than a decoding batch.

Pipeline parallelism alone can cause significant GPU idle time (dubbed “bubbles”) in LLM serving, as shown in the following figures. This is due to the unequal amount of time spent on each phase when we pipeline them together.

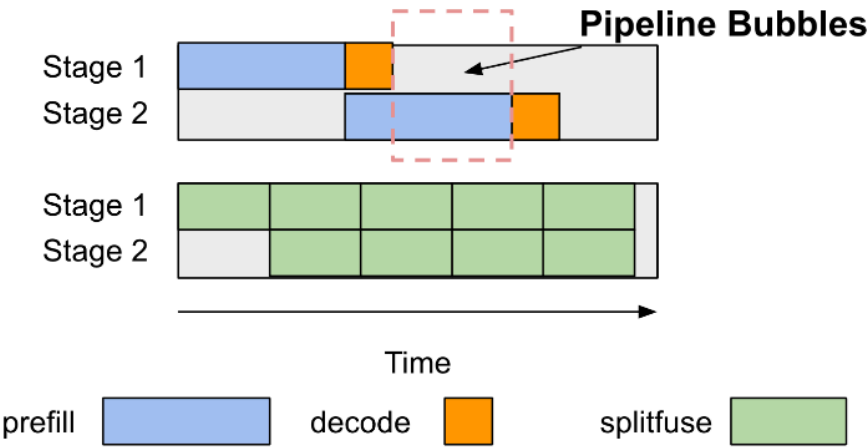


Figure 5. Two-stage pipeline parallelism scheduling and bubble overhead – with and without Dynamic SplitFuse.

We apply the SplitFuse optimization, originally developed by the DeepSpeed team (and already implemented in the open source

vLLM by the community), to combat these bubbles. In particular, prompts are prefilled in smaller chunks and batched with decoding tasks to form a fixed number of tokens to be processed as a batch per step. SplitFuse yields two substantial improvements: (1) It reduces the amount of time the GPUs are only processing decode batches and ensures the GPU utilization is constantly high; and (2) It avoids pipeline bubbles (above, top), as now each pipeline step will process a batch deliberately created to take an equal amount of time (bottom).

AUTHOR

Snowflake AI Research

It is critical to tune the maximum number of tokens allowed to be processed in a single forward pass. The Snowflake inference stack f ll : in : tu ✉ : depending on the nature of the traffic to ensure high utilization and minimized bubbles.

SHARE

Other performance tuning

We have identified many other places where we can further improve the pipeline parallelism efficiency, including reducing the sampler overhead; making the sampler asynchronous, to be out of the critical path; and vectorizing the prepare_input in vLLM. We are working actively with the vLLM community (including Anyscale, UC Berkeley and UCSD) to ship these optimizations to vLLM main. We project these optimizations will yield another 25% improvement on the throughput of serving Llama 3.1 405B.

Performance Benchmarks

In this section, we share the performance profile for various production workloads shown in Table 1, and compare it with the existing OSS baseline when applicable.

	<i>Snowflake Average Production Workloads</i>	<i>Long Context Workloads</i>
<i>Input Prompt Length</i>	2K	8K-128K
<i>Output Length</i>	256	256

Table 1. Different workload profiles, for which we measure latency and throughput using our inference system stack.

Hardware

Our benchmark cluster consists of AWS p5.48x large instances, each with 8 H100 GPUs — totaling 640 GB of GPU memory, 192 vCPUs and 2TB of memory. We ran our FP8 benchmarks using both single-node and two-node setups, and BF16 benchmarks using two nodes.

AUTHOR

[Snowflake AI Research](#)

Baseline

Previous to our work, inference of a massive model like Llama 3.1 405B could require using 16-way tensor parallelism across multiple nodes. Therefore, we use vLLM with 16-way TP as our baseline².

SHARE

Results

Snowflake Average Production Workloads

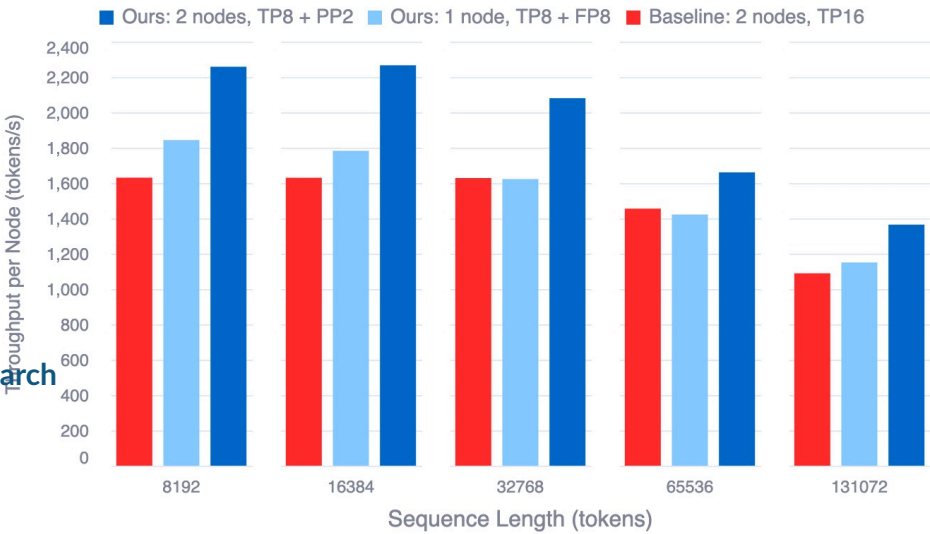
On the one hand, many interactive workloads in production demand low latency, while on the other hand, achieving high throughput is vital to lowering inference cost. Our system can achieve low-latency real-time inference of Llama 3.1 405B while achieving high throughput for average Snowflake production workloads.

Figure 1 shows that compared to the baseline, our inference stack supports real-time inference with up to **3x lower e2e latency**, and over **1.4x higher token throughput** (Fig 1.a and 1.b), and supports **1.6x higher arrival rate** for real-time inference (Fig. 1.c).

At lower traffic, it achieves TTFT under 1 second, and ITL under 70ms. At high traffic it can support a max throughput of about 2.5K tokens/sec/node or 250 TFlops/GPU.

AUTHOR

Snowflake AI Research



SHARE

Figure 6: Throughput per node for different context windows, ranging from 8K to 128K, comparing our inference stack with baseline.

Supporting long context windows is both memory and memory-bandwidth intensive. At large context windows, the memory required to store KV caches increases linearly, requiring more memory and limiting the batch size that can be used. At the same time, significant latency is spent in reading the increased KV caches affecting throughput.

Despite these challenges, as shown in Fig. 6, our inference stack well supports 128K context windows even on a single node, while achieving 1.25-1.4x higher throughput per node compared to the baseline when using multi-node setup.

Learn More:

More details on how to get started with Meta’s Llama 3.1 405B and Snowflake Cortex AI can be found in this [quickstart guide](#).

For the most up-to-date resources on how to run Llama 3.1 405B with Snowflake AI Research’s open source inference and fine-tuning pipelines, see our [GitHub repo](#). This repo includes both example code and documentation for running Llama 3.1 and Snowflake Arctic models.

Learn more about the continued innovation coming out of Snowflake’s AI Research team, and meet the experts driving the

future of AI forward in the [AI Research hub](#).

For enterprises interested in distilling Llama 3.1 405B for their own domain-specific use cases and getting additional support from Snowflake's AI Research team, fill out [this form](#).

Contributors

AUTHOR

Snowflake AI Research

Snowflake AI Research: Aurick Qiao, Reza Yazdani, Hao Zhang, Jeff Rasley, Flex Wang, Gabriele Oliaro, Yuxiong He, Samyam Rajbhandari (Tech Lead)

SHARE

OS Community Collaborators: Murali Andoorvedu @CentML, [f](#) [t](#) [in](#) [le](#) [✉](#) CSD and Anyscale

Acknowledgements

We would like to thank Meta for their wonderful partnership, the Open Source community for their collaboration, and our leadership at Snowflake for their continued support.

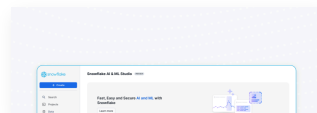
¹ Real-Time inference implies little or no wait time before the generation of the first token and token generation roughly at the rate that humans can read. We define real-time inference as TTFT < 2 seconds and ITL < 250 ms. We configured VLLM to maximize throughput while achieving ITL < 250 ms by limiting the max tokens in a batch with dynamic SplitFuse.

² Other baselines: Very recently, an alternate fp8 solution was upstreamed to VLLM by Neural Magic that we hope will further improve VLLMs performance for these massive models. Unfortunately, we could not get it to work with Llama 3.1 405B in the short timeframe. We are excited to leverage it in future as it is complementary to the optimizations discussed here.

SHARE



RELATED CONTENT





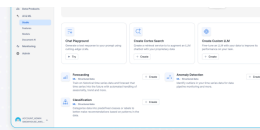
Product and Technology AI & ML

MAR 05, 2024

Easy and Secure LLM Inference and Retrieval Augmented Generation (RAG) Using Snowflake Cortex

Because human-machine interaction using natural language is now possible with large language models (LLMs), more data teams and developers can bring AI to their daily workflows. To do this efficiently...

[Full Details](#)



Product and Technology AI & ML

JUN 04, 2024

Snowflake Announces State-of-the-Art AI to Talk to your Data, Securely Customize LLMs and Streamline Model Operations

Generative AI presents enterprises with the opportunity to extract insights at scale from unstructured data...

[More to follow](#)

READ OUR EBOOK

GENERATIVE AI IN PRACTICE

DOWNLOAD HERE

AUTHOR

Snowflake AI Research



SHARE



FORM

SOLUTIONS

RESOURCES

EXPLORE

ABOUT

Cloud Data Platform

Pricing

Marketplace

Security & Trust

Snowflake for Financial Services

Snowflake for Advertising, Media, & Entertainment

Snowflake for Retail & CPG

Healthcare & Life Sciences Data Cloud

Snowflake for Marketing Analytics

Resource Library

Webinars

Documentation

Community

Procurement

Legal

News

Blog

Trending

Guides

Developers

About Snowflake

Investor Relations

Leadership & Board

Snowflake Ventures

Careers

Contact

Sign up for

Snowflake

Communications

diana.shaw@sno

United States

By submitting this form, I understand Snowflake will process my personal information in accordance with their **Privacy Notice**. Additionally, I consent to my information being shared with Event Partners in accordance with Snowflake's **Event Privacy Notice**. I understand I may withdraw my consent or update my preferences **here** at any time.

SUBSCRIBE NOW



AUTHOR

Snowflake AI Research

SHARE

