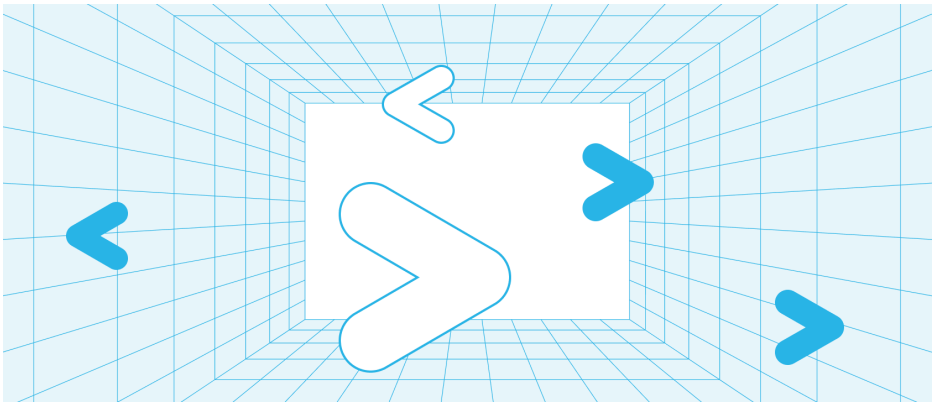
JUL 23, 2024

AUTHOR
Snowflake AI Research

Fine-Tuning Llama 3.1 405B on a Single Node using Snowflake's Memory-Optimized AI Stack

SHARE



Fine-tuning and inferencing large models, such as Meta Llama 3.1 405B, present significant challenges due to their enormous size, requiring innovative approaches to overcome memory constraints. This blog outlines a series of optimizations that enable fine-tuning models with 400+ billion parameters on small multi-node clusters or even on a single 8 x H100-80GB host (see our post on inference optimizations here). By combining parameter-efficient fine-tuning (specifically LoRA), FP8 quantization, ZeRO-3-inspired sharding and targeted parameter off-loading, the Snowflake AI Research team dramatically reduced per-GPU memory requirements, from an initial 6.5 TB, to under 80 GB of GPU memory. These techniques, implemented primarily as drop-in replacements for PyTorch's nn.Linear module and upstreamed to DeepSpeed, enable efficient handling of the model's 405 billion parameters while managing activation memory and other overheads. This approach not only makes fine-tuning Meta's Llama 3.1 405B possible, but also demonstrates a path for working with extremely large language models (LLMs) on limited

hardware resources. Our optimized fine-tuning implementation for Llama 3.1 405B is open sourced, and it will also power Cortex fine-tuning of Llama 3.1 405B, coming soon.

Understanding memory consumption

Llama 3.1 405B truly exemplifies the '*large*' in LLM, making basic fine-tuning extremely difficult or impossible — even on the most advanced hardware. To gain a deeper understanding of the memory overheads required by Llama 3.1 405B, let's first break down the parameter counts and memory overheads required by training below.

AUTHOR
Snowflake AI Research

Module	Parameters	Total
LlamaMLP	2.62 billion x 126 layers	330 billion
LlamaAttention	0.6 billion x 126 layers	76 billion
Embedding + norms	2 billion	2 billion
Grand total		**408 billion**

In our setup, we assume a single 8 x H100-80GB host (e.g., p5.48xlarge) and leverage DeepSpeed's ZeRO to help reduce optimizer state memory overheads. The calculations below are based on Figure 1 from ZeRO: Memory Optimizations Toward Training Trillion Parameter Models.

ZeRO strategy w. bfloat16	Memory required per GPU
Baseline w/o ZeRO	6.5 TB
ZeRO stage 1	2.2 TB
ZeRO stage 2	1.4 TB
ZeRO stage 3	0.8 TB

As demonstrated above, even by going all the way to ZeRO stage 3 we are unable to fit all the required optimizer states and model weights in a single 8 x H100 host. While more-advanced techniques like ZeRO-Infinity could be explored, we chose to go down a slightly different path in this work.

AUTHOR
Snowflake AI Research

System Design

We will now go over the optimizations we applied to fine-tune Llama 3.1 405B and update our per-GPU memory overhead along the way.

Parameter-efficient fine-tuning + ZeRO stage 2

In order to reduce the significant overheads of full-precision fine-tuning, we employed a common practice of parameter-efficient fine-tuning. Specifically, we implemented and upstreamed to DeepSpeed a version of low-rank adaptation (LoRA), using a lora_r value of 64. This optimization alone reduced our per-GPU memory requirements with ZeRO-2 from 1.4 TB down to 825 GB.

FP8 quantization

Following the approach used for **Arctic** and **Llama inference**, we have developed hardware-agnostic FP8 quantization kernels. These kernels support both sparse MoE models like Arctic and dense models like Llama 3.1, both have been upstreamed to DeepSpeed. For details on our quantization methods, please see our **accompanying post** on these optimizations in the context of inferencing Llama 3.1 405B.

More specifically, we applied FP8 quantization to the "base weights" that are frozen during training, which makes applying this type of quantization relatively straightforward since it does not require generating gradients. This optimization further reduces our memory requirements above from 825 GB down to 417 GB.

ZeRO-3 inspired sharding

The total frozen and quantized parameters now sum up to 408 billion parameters. We apply a very similar sharding strategy as ZeRO-3 but targeted specifically at these frozen "base weights." Unlike traditional ZeRO-3, our implementation requires no backward pass, simplifying the implementation of sharding. We shard the weights by 8 and add an allgather during the forward for each frozen linear weight. The allgather is done over NVLink, minimizing performance penalty.

All optimizations described here are developed as a drop-in replacement for torch's nn.Linear module. This allows us to have very tight control over all of our optimizations and simplifies the process of isolating our frozen "base weights" and sharding them across all 8 GPUs.

By sharding our base weights, we now reduce our memory requirements above from 417 GB down to 60 GB.

AUTHOR
Snowflake AI Research

Targeted-parameter off-loading

You might think we can now claim victory with the previous optimization since we have now brought down the per-GPU memory requirements under 80 GB. However, these estimates do not account for several other factors that contribute to memory overheads during training. These other factors include (but are not limited to) activation memory, memory fragmentation, NCCL and other library overheads.

The most significant of these memory overheads, especially for this model, comes from activation memory. A significant portion of this can be reduced by applying activation checkpointing. However, this still requires a non-trivial amount of memory to:

1. Store the activation checkpoints,
2. Store the full set activations for a single transformer layer between two activation checkpoints, and
3. Store the projection-layer forward and backward memory

In addition to these, the LoRA model states also occupy a non-trivial amount of memory for a large model like the Llama-3.1 405B. In combination, these four require roughly 14 GB of memory.

> For a batch size of 1K, the memory required for activation checkpoints can be estimated as: 16k (hidden size) x 126 (# of layers) x 1k (# of tokens) x 2 /1k / 1k / 1k ~= 4GB.
>
> Similarly, the memory needed to store the full set of activations for a single transformer can be estimated as 16k (hidden size) x

1k (# of tokens) x 64 (estimated scale factor, see eq.1 of Megatron-LM for more details and replace GeLU with SiLU) x 3 (base weight path, LoRA path and some other overhead) / 1k / 1k /1k ~= 3GB.

The projection-layer forward and backward memory can be estimated as [128k (vocab size) x 1k (# of tokens) x 6 (est. of activation sizes) x 2 + 128k (vocab size) x 16k (embed size)] x 2 (gradient of weights)] / 1k / 1k / 1k ~= 5.5GB.

Finally, the LoRA parameters contribute to 126 (# of layers) x [6 .6k. 2 (Q,O) + (64x16k+64x2k)x2 (k,V) + (64x52k+64x16k)x3 (MLP)] x (2+12) /10^9 = 4.1GB.

These additions bring our per GPU memory requirement from 60 GB to about 74 GB, which is extremely tight.

To address these remaining requirements, we apply targeted off-loading of a percentage of nn.Linear base weights to CPU when we are not using them. All of these optimizations combined finally allow us to fine-tune Llama 3.1 405B on a single node! See Figure 1 (below) for a visualization of the final states and their rough proportions. In an environment with two or more nodes we no longer need CPU off-load due to higher sharding opportunities.

AUTHOR
Snowflake AI Research

AUTHOR

Snowflake AI Research

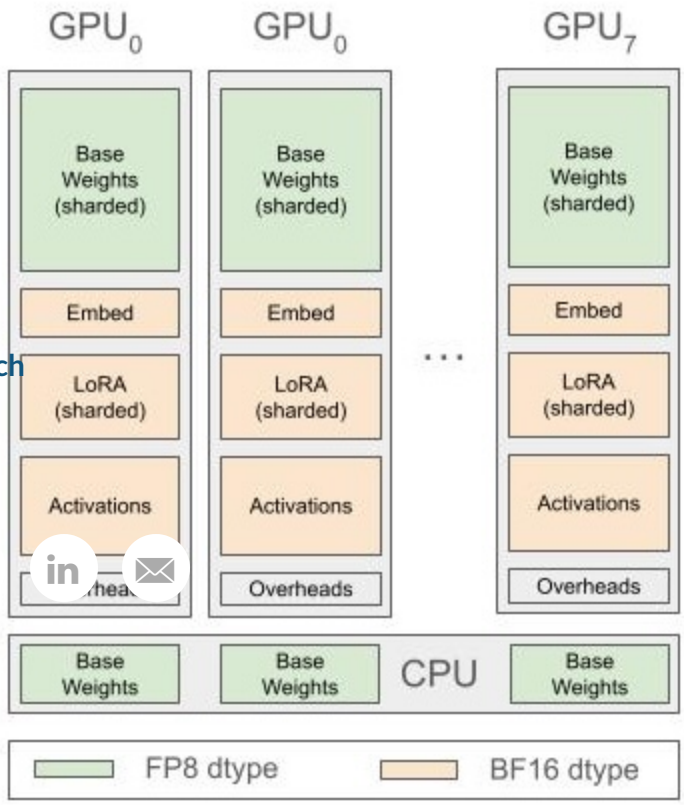


Figure 1. Memory overheads with all optimizations in a one-node scenario.

End-to-end training times

With all of the above optimizations, single node fine-tuning over 20M tokens on 8xH100-80GB takes roughly 24 hours and 4 hours on two nodes at the time of publication. For latest results see our repo.

Quality evaluation

From an end-to-end perspective, all of the optimizations described here have been validated by comparing the training loss, with and without each optimization enabled, using our fine-tuning pipeline on the alpaca data set. We provide this script in our GitHub repo for your reference.

Getting Started

Open source fine-tuning system: For the most up-to-date resources on how to run Meta's Llama 3.1 405B with Snowflake'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.

Llama 3.1 405B in Snowflake Cortex: Llama 3.1 405B will be available in public preview for our Cortex Fine-Tuning customers soon.

AUTHOR

Snowflake AI Research

Contributors

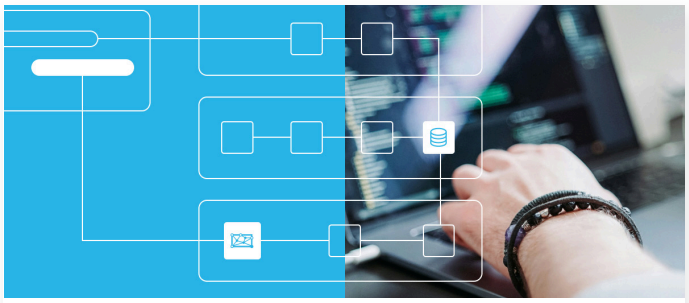
Snowflake AI Research: Jeff Rasley, Rajhans Samdani, Zhewei Yao, Yuxiong He, Samyam Rajbhandari (Tech Lead)

Acknowledgements We would like to thank Meta for their [illegible] leadership, the OSS AI community for their collaboration, and our leadership at Snowflake for their continued support.

SHARE

SHARE

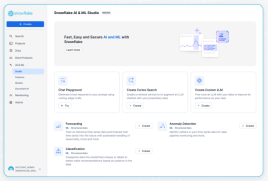
RELATED CONTENT



Product and Technology AI & ML

MAR 05, 2024

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



Product and Technology AI & ML

JUN 04, 2024

Snowflake Announces State-of-the-Art AI to Talk

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…

More

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

AUTHOR

Snowflake AI Research