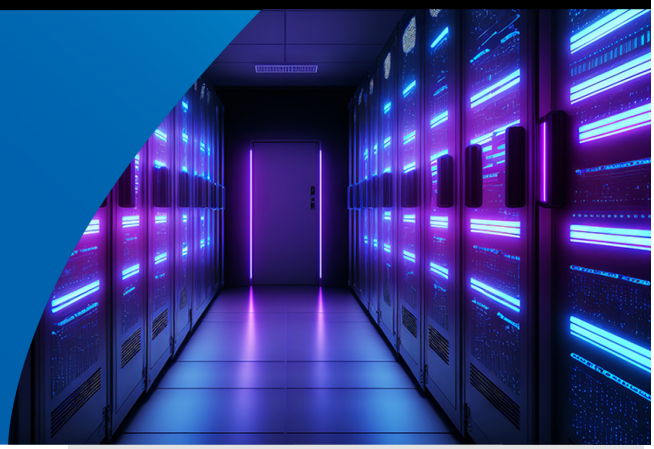
A memory perspective: The effects of fine-tuning LLMs with high-bandwidth memory



Authors: Felippe Vieira Zacarias, Kiran Palli, Sudharshan Vazhkudai, Evelyn Grevelink

In recent years, we have seen considerable growth in artificial intelligence (AI) models for natural language processing (NLP), largely due to the remarkable performance of large language models (LLMs) across a wide range of tasks.

Specifically, the progress in LLM training has been fueled by a rapid growth in model complexity, together with advancements in training methodologies and hardware technology. LLMs present distinct challenges as the number of model parameters soars from millions to trillions. Such growth places new demands on compute power and memory subsystems, with memory requirements potentially escalating from a few gigabytes to several terabytes. The development of specialized hardware — such as Micron's high-bandwidth memory (HBM) — has been essential in meeting the intensive demands of LLM training. In particular, the latest generation of HBM, Micron HBM3E, is a compelling solution for training LLMs.

This report provides an analysis of LLM fine-tuning performance on NVIDIA's HGX H100 platform using HBM, aiming to benchmark the capabilities of HBM in handling the computational demands of fine-tunning LLMs. The testing details — which include system configurations, optimization techniques and performance metrics — are documented to educate system and silicon architects and other key stakeholders about the use of memory solutions for AI workloads. The results serve as a valuable resource for system architects and stakeholders, guiding them in making informed decisions for the design, deployment and procurement of memory solutions optimized for AI workloads.

Key takeaways

50%

Micron 24GB-36GB HBM3E increases per-placement capacity by 50%, addressing the high-capacity needs of LLMs.

This finding can increase per-GPU memory capacity up to 288GB with 8 HBM placements.

30%

Micron's HBM3E consumes 30% lower power than the competition.[a]

Micron HBM3E lowers the overall power consumption and temperature of a server system, reducing total cost of ownership for data center deployments.

96GB & 128GB

Users can avoid out-of-memory errors with high-capacity DIMMs.

Micron high-capacity DDR5 RDIMMs, available in 96GB and 128GB capacities, enable larger LLMs (with 65 billion parameters and above) when training is offloaded to the CPU.

a. Based on internal measurements and publicly available data

Introduction to NLP and LLM training

Natural language processing (NLP) enables computers to manipulate, interpret and generate human language. To learn the rules of language, NLP uses statistical modeling by applying probability distributions to a sequence of words from a large corpus of text. Previously, architectures such as convolutional neural networks (CNN) were adopted to build NLP models, but they suffered from slow computation due to the increasing number of operations when the number of queries was scaled up.

The rapid progress of NLP models in recent years has been made possible mostly through adoption of the transformer architecture [2][b] developed by researchers at Google and the University of Toronto. The transformer architecture was initially used to translate language, but because of its superior computational performance (by processing all inputs in parallel) over previously used architectures, it has been explored in several scenarios. Moreover, because of its success in distinct downstream applications (text summary, autocomplete, chat dialogue generation, etc.), the number of parameters in NLP models has rapidly increased over the years, as shown in Figure 1. The figure shows the evolution of model sizes since 2017, beginning with Transformer model at 65 million parameters announced by Google in June 2017. Model sizes greater than 1 trillion are depicted using a dotted line. The largest models we include can achieve parameters sizes above one trillion because they use sparsely activated structures, where only a portion of the LLM's neurons are activated during inference, rather than all of them. However, their widespread adoption is hindered by factors such as complexity, communication costs, and training instabilities [15]. Despite these obstacles, their architectural design should be considered as a strong candidate for future model scaling. Additionally, models like GPT-4 and Gemini are noted for their multimodal capabilities, meaning they can handle not only text but also visual and auditory inputs such as image, video, and audio. Figure 1 is based on information in reference [1].

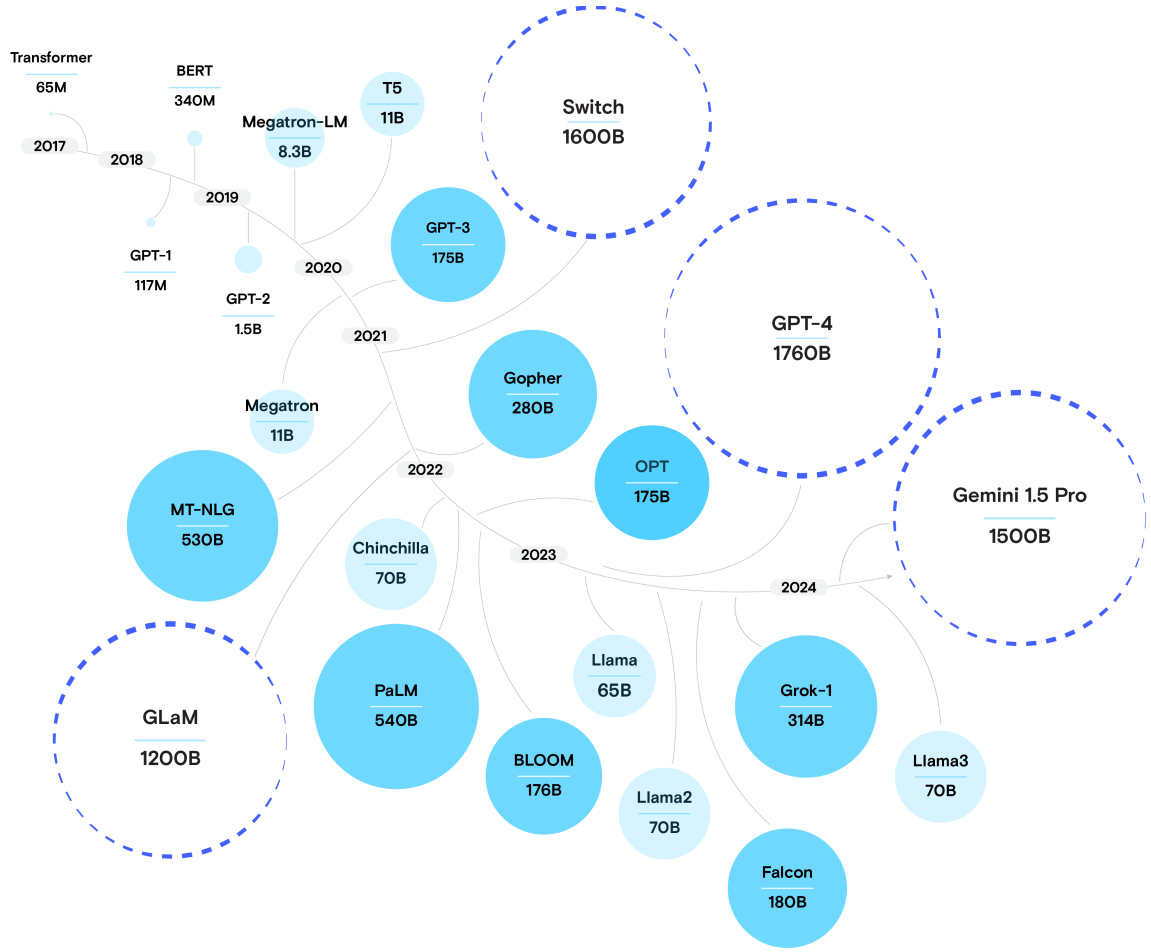


Figure 1. Evolution in the size of state-of-the-art LLMs from 2017 to 2024

b. Numbers in square brackets refer to source documents, which are found in the References section at the end of this report.

Typically, LLMs are trained using a two-staged approach: pretraining and fine-tuning.

Pretraining

In the initial stage (or pretraining), models are pretrained on a large set of data from general sources such as Wikipedia, blogs and academic journals. This stage allows the model to learn the rules of a language but is not intended to perform any specific task, which eliminates the need for labeled data. The absence of labeling allows the model to leverage extensive volumes of data to enhance its performance, as compared to models trained solely on a limited set of labeled data.

Fine-tuning

For the next stage (or fine-tuning), the model can be updated using curated data to provide a more specific output prediction for the desired NLP task.

This two-staged approach is beneficial, and while the pretraining step is significantly more computationally expensive than fine-tuning, it only needs to be done once. The same pretrained model can then be fine-tuned for a variety of applications.

However, training such models has some challenges [1]:

1. Fitting the parameters of these large models in the main memory of the largest GPU available today has become infeasible.
2. Training times are longer due to the higher number of compute operations created by the growing number of model parameters.

Consequently, efficient training of large language models (LLMs) requires robust hardware specifications and advanced software techniques. At the forefront of this field, the latest generation of NVIDIA GPUs consistently deliver higher performance for AI workloads than previous generations of hardware by a wide margin. The Hopper GPU architecture — with its fourth generation of tensor cores that support several precisions, including FP64, TF32, FP32, FP16, INT8 and FP8 — is four times faster in training performance for LLMs [11] over the prior generations. Combined with the NVIDIA NVLink interconnect, which offers 900 GB/s of GPU-to-GPU communication bandwidth and HBM bandwidth, Hopper GPUs efficiently scale and provide the high-quality performance needed to run LLMs.

HBM3E surpasses the previous generation of HBM technology by providing a memory bandwidth that exceeds 1.2 TB/s. Additionally, it offers a 24GB capacity — a 50% increase per 8-high stack — to facilitate training with higher precision and accuracy. For better power efficiency, HBM3E's energy-efficient data path reduces thermal impedance, leading to a 2.5 times improvement in performance per watt.

Transformer architecture for generative AI

This section describes transformer architecture, a deep learning model designed for tasks such as text and image generation. Transformer architecture revolutionized generative AI by introducing attention mechanisms to enable models to focus on relevant parts of input sequences. As AI models become more prevalent, machine learning (ML) libraries are used to simplify model implementation, allowing developers to reuse optimized code and streamline generative AI deployment.

Details of the transformer architecture

To understand transformer architecture, we need to know what the intricacies are of this architecture and how each module contributes to the generation of text. We must also recognize popular frameworks used to deploy such AI models.

Transformer architecture [2] has become a standard choice for creating LLMs. Unlike earlier models that focused on nearby words, the transformer allows models to learn the context of all words in a sentence using a self-attention mechanism. This feature enhances the model's ability to process longer sequences and handle text input.

Transformer architecture [2] typically has two components: an **encoder** that processes the input text (left-hand-side) and a **decoder** that predicts the output (right-hand-side), as presented in Figure 2. The encoder is composed of *N* identical layers (each with two sublayers).

The first sub-layer is the attention mechanism (that captures the relationship between the words in a sequence), and the second is the position-wise fully connected feed-forward network. The decoder is also composed of N identical layers, adding the multi-head attention sub-layer to consume the output of the encoder stack.

Different combinations of these stacks can be used to create distinct models for several scenarios:

- ***Encoder-only*** models are good for learning embeddings used in classification and sentiment analysis problems (e.g., BERT).
- ***Encoder-decoder*** models are used for generative tasks that depend on input like translation and summarization (e.g., T5).
- ***Decoder-only*** models (also called autoregressive models) are good for generative tasks in an autoregressive style such as chatbots (e.g., GPT).

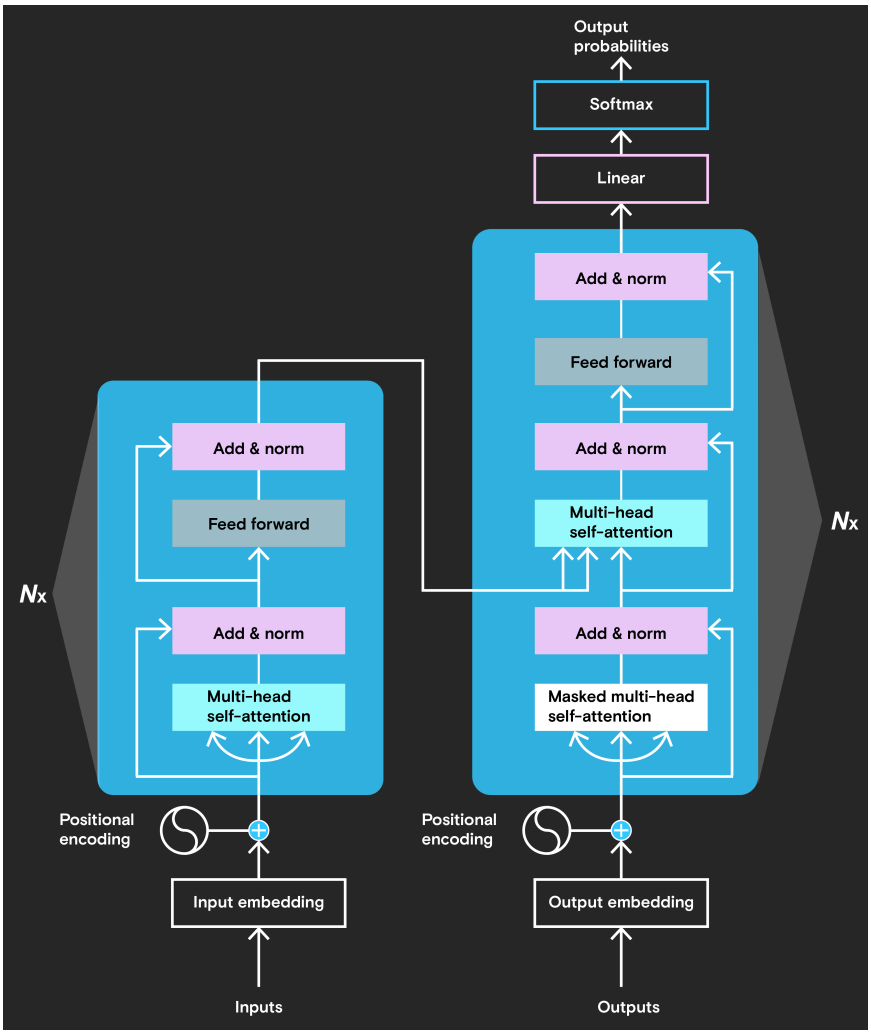


Figure 2. Transformer model architecture (concept from [2])

Over the years, researchers have built notable decoder-only models that are highly efficient at several NLP tasks. One of the most important decoder-only models is the generative pretrained transformer (GPT), which was later enhanced to provide humanlike responses to questions in a chat format. In this architecture (Figure 3), the input sequence (also called query or prompt) is transformed into tokens. Each token is encoded to the model vocabulary and then mapped to an embedding (projected information of the word's meaning to a smaller space) with its relative position. It is then used to calculate the relationship to all other words in the sequence, and the result of this calculation is fed as an input to the feed forward layer. These multi-head self-attention and feed forward modules are repeated throughout all layers in the model. The output of the last layer is used to generate the probability distribution of the next token in the response. The model can then randomize this function to generate different answers to the same question.

The steps executed in the transformer architecture are described below (concept from [3]). This process is repeated for every token requested to be generated.

1. **Text and position embedding**
   a. **Tokenization**: The sequence is first separated into tokens. The tokens are then mapped to the vocabulary list, resulting in an encoded matrix.
   b. **Embeddings**: The encoded matrix is projected into the embedding space (multiplying by the embedding weights), generating the embedding matrix. The size of this matrix is also the size of the model.
   c. **Positional encoding**: Then the position of each token in the sequence is converted into another matrix. It provides the model information about the relative position of the token, thus drawing better conclusions of the relationship between tokens. This matrix will be added to the embedding matrix.
2. **Masked multi-head self-attention**

   Attention mechanism: Transformer models use self-attention [2] to compute the relationship between tokens.
3. **Residual connections and layer normalization: (+) arrows and "Layer norm" in figure**
   a. Residual connections appear after the computation of each submodule (**Masked multi-head self-attention** and **Feed forward**). They pass information from the input to the output of a submodule, ensuring a stable and effective training by addressing issues that result from the gradient-based training.
   b. Layer norms stabilize and improve training by normalizing the output values and keeping them within a range.
4. **Feed forward network**

   The output of the self-attention mechanism is fed to a fully connected feed forward network (multilayer perceptron or MLP).
5. ***N* box (blue)**

   The computation executed in the attention mechanism, feed forward network and two normalization layers are repeated on all *N* transform layers.
6. **Decoder**

   The decoder, which consists of text prediction, receives the output of the last transform layer, converts the embeddings to the vocabulary and gives the probability for different tokens. A random function can be used to select the tokens within range of the highest probabilities.

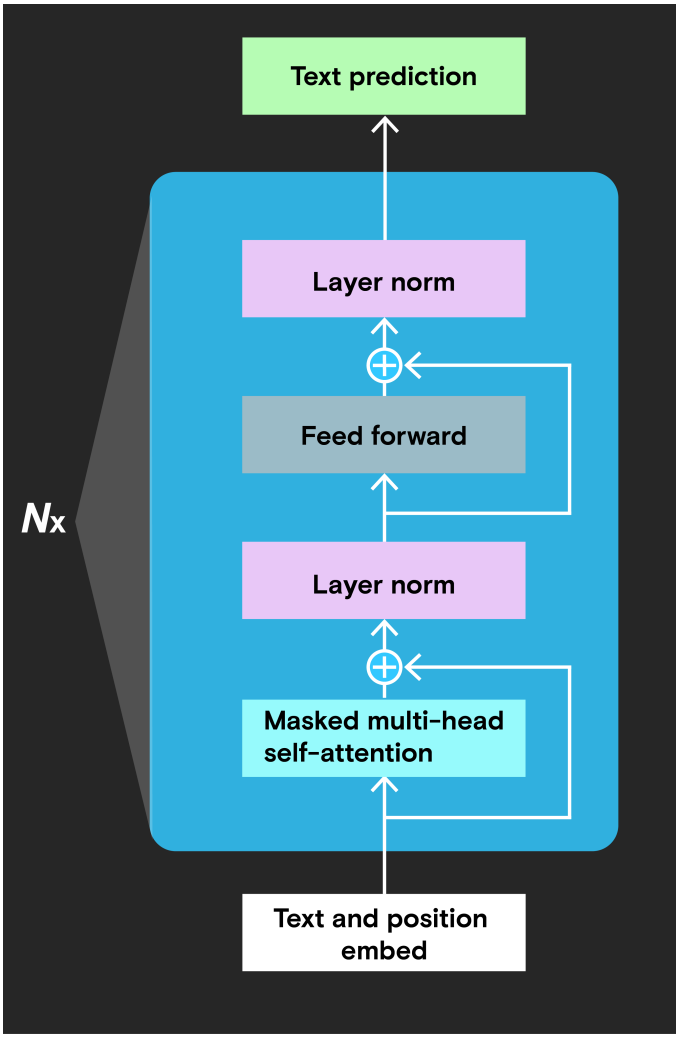


Figure 3. GPT-like transformer decoder architecture

Libraries

As AI models become more widespread, it's necessary to facilitate the development of these models and speed up their deployment. To accomplish those actions, a range of tools, modules, frameworks and libraries are applied to the workflow.

PyTorch

The PyTorch library [13] is an optimized tensor library for deep learning that uses both GPUs and CPUs. The torch package contains data structures for multidimensional tensors and defines mathematical operations over these tensors. PyTorch offers several ways to perform distributed training in a single machine with multiple GPUs or across several GPUs within multiple machines.

DeepSpeed

The DeepSpeed library [14] from Microsoft is an open-source deep learning optimization library solution built on top of PyTorch. It offers parallelism through data, mode, and pipeline, while PyTorch focuses on data parallelism (DP). The library includes the Zero Redundancy Optimizer (ZeRO), which removes memory redundancies across data-parallel processes by partitioning the three model states (optimizer states, gradients and parameters) across the processes instead of replicating them.

The ZeRO approach is divided into three stages [12]:

- **ZeRO-1**: The optimizer states are partitioned across the processes so that each process updates only its partition.
- **ZeRO-2**: The reduced 32-bit gradients for updating the model weights are also partitioned such that each process retains only the gradients corresponding to its portion of the optimizer states.
- **ZeRO-3**: The model parameters are partitioned across the processes. ZeRO-3 automatically collects and partitions them during the forward and backward passes. This stage can also offload all model states to both CPU and NVMe memory for huge memory savings. When using ZeRO-3 on N GPUs, each GPU stores only $1/N$ (or one-Nth) of the model. But ZeRO-3 requires ~50% more network communication between the GPUs to update the parameters. PyTorch released its own ZeRO-3 implementation called Fully Sharded Data Parallel (FSDP).

Other DeepSpeed features include mixed precision training, multi-GPU and multinode training, custom model parallelism, and memory and bandwidth optimizations.

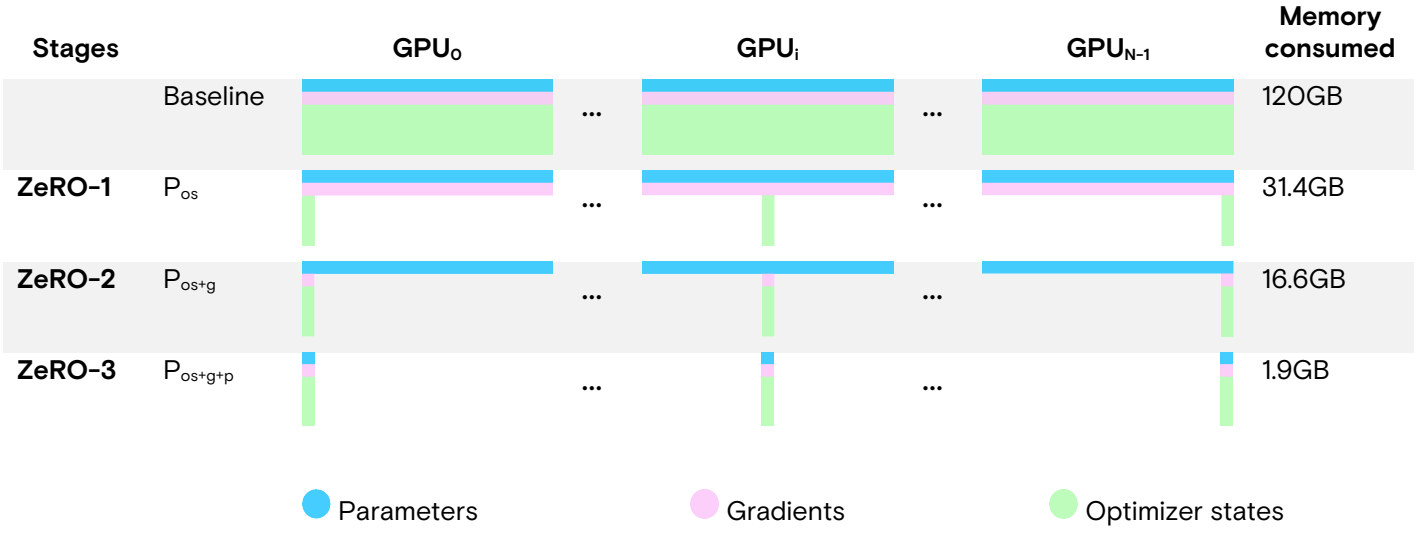


Figure 4. Comparison of the per-device memory consumption of model states with three stages of optimization

In Figure 4 [4], Ψ denotes the number of parameters, K denotes the memory multiplier, and N_d indicates the degree of parallelism.

Quantization

Another common way to handle the capacity requirements of LLMs is by using quantization. Quantization is the process of representing input from a state of holding more information to a state of holding less information. For example, LLMs usually take datatypes with more bits and convert them to fewer bits since the model's parameters are usually FLOAT32 (FP32) or FLOAT16 (FP16). There are two main weight quantization techniques:

- **Post-training quantization (PTQ)**: This technique converts the weights of a pretrained model to lower precision.
- **Quantization-aware training (QAT)**: Quantization is applied during the pretraining at the expense of extra computational resources [10]. QAT is more computationally intensive and requires more time than training using full precision, making it less popular than PTQ.

QLoRA

An example of the PTQ technique is the QLoRA [8] approach. It's an efficient fine-tuning approach that backpropagates gradients through a 4-bit quantized pretrained model into low-rank adapters (LoRA). LoRA [9] reduces memory requirements by updating a small set of trainable parameters (adapters) while the full model parameters remain fixed. Compared to a 16-bit fully fine-tuned baseline, QLoRA reduces the average memory requirements of fine-tuning a 65B parameter model from more than 780GB of high-bandwidth memory to less than 48GB without degrading the runtime or predictive performance [8]. This reduction is mostly achieved by applying two techniques:

- **4-bit NormalFloat (NF4)**: This is a datatype with information that is theoretically optimal for normally distributed weights.
- **Double quantization**: This technique reduces the average memory footprint by quantizing the quantization constants.

System setup

To analyze the performance of select LLM models, testing and validation were carried out on a single NVIDIA HGX H100 box (manufactured by Supermicro). Tables 1 and 2 detail the GPU and host configuration of the system under test (SUT), while Figure 5 depicts the system architecture. Note that "HGX" is used as shorthand in the sections that follow.

SUT GPU configuration	
Model	NVIDIA H100
Form factor	8x NVIDIA H100 SXM
HPC and AI compute (FP64/TF32/FP16/FP8/INT8)	535TF/8PF/16PF/32PF/32POPS
GPU max frequency	1980 MHz
Memory max frequency	2619 MHz
Memory capacity	640GB, where each H100 GPU has 80GB of HBM3 memory
NVSwitch GPU-to-GPU bandwidth	900 GB/s
Total aggregate HBM bandwidth	~27 TB/s
GPU max operating temp	87 C
GPU shutdown temp	92 C
Memory max operating temp	95 C
Power limit	700W

Table 1. GPU configuration of the system under test

SUT host configuration	
Model	Intel® Xeon® Platinum 8468
Core(s) per socket	48
Socket(s)	2
CPU max speed	2101 MHz
L3 cache	120 MiB
Memory type	32x Micron 64GB DDR5 RDIMMs
Total memory capacity	2TB
Memory speed	4400 MT/s
DIMMs/channel	2 DIMMs per channel / 16 channels
Operating system	Ubuntu 20.04 (Kernel 5.4.0)

Table 2. Host configuration of the system under test

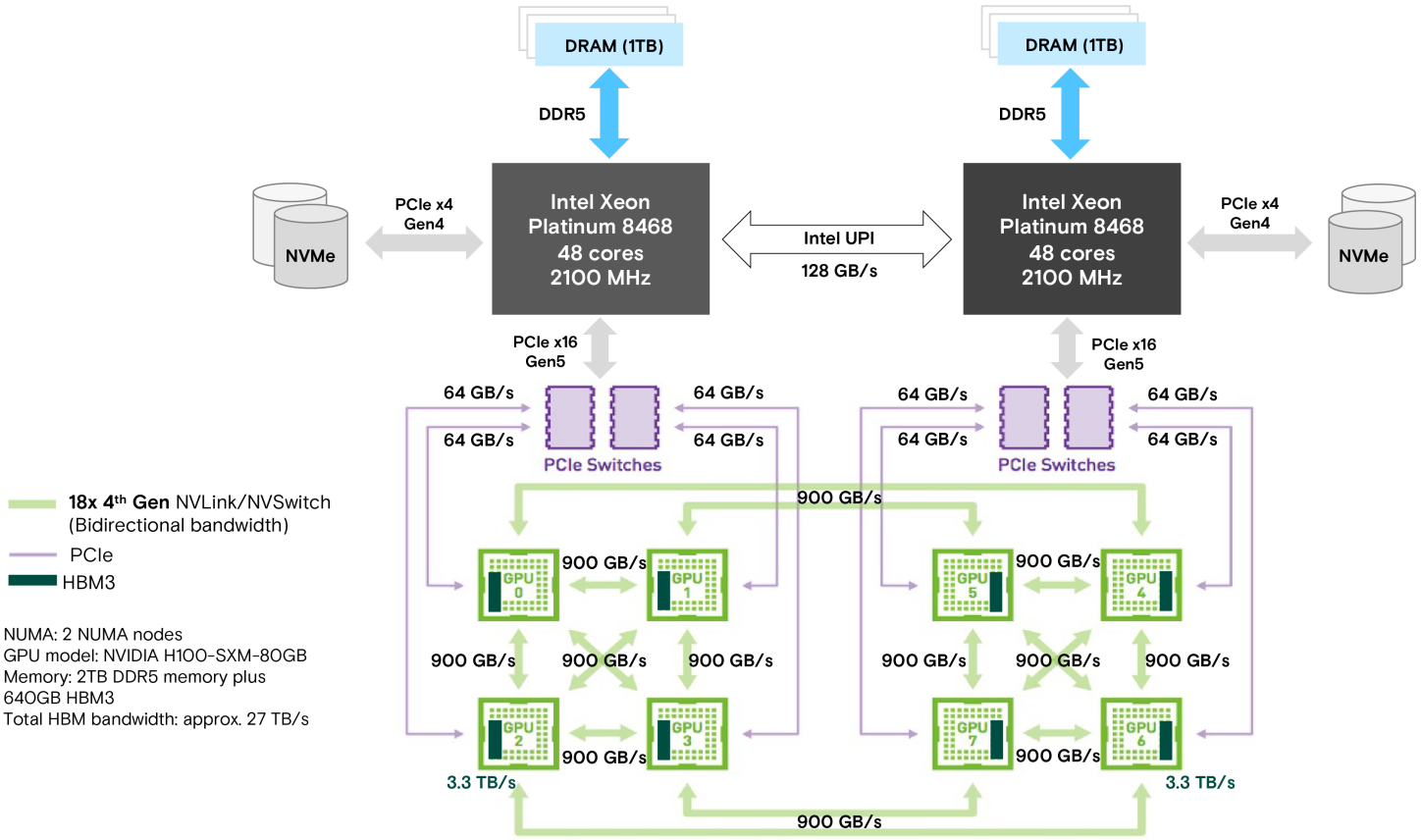


Figure 5. System architecture of NVIDIA HGX H100 used in the experiments

Methodology

The test results presented in this document are based on the system setup shown in the figure above and the following methodology of fine-tuning. Micron Data Center Workload Engineering (DCWE) measured the results on a single H100 HGX node. For the analysis, see the next section.

Unlike pretraining, fine-tuning is the process of specializing the model to perform a specific task using a curated dataset. Fine-tuning leverages the potential of the model for distinct scenarios and allows end users to create several derivative models to meet the requirements of various application scenarios (code generation, grammar correction, classification, etc.). While fine-tuning requires fewer resources than pretraining, end users are still restricted to the memory available on their server nodes, which limits the size of the model that can be fine-tuned.

In this analysis, we used a collection of decoder-only language models (Llama)[6]. The model series meet these criteria:

- They vary from 7B to 65B parameters.
- They are pretrained on trillions of tokens and competitive with the best and large existing LLMs.
- They implement improvements proposed by a few models such as PaLM, GPT3 and GPTNeo.

The fine-tuning methodology was the training recipe and data released by Stanford University's Center for Research on Foundation Models [5] and used to fine-tune Meta's Llama models [6] into a high-quality instruction-following model called Alpaca, which is trained on 52K instruction-following demonstrations. To fit the model into our architecture, we used DeepSpeed, which includes ZeRO to remove memory redundancies. We also used QLoRA quantization to decrease the computation and memory capacity required to fine-tune large models.

Detailed Results

This section presents the performance and power results from testing on **Llama 7B**, **Llama 65B** and **Llama 65B** using QLoRA. It also compares these LLM models to some commonly used AI models, focusing on their performance.

Llama 7B

This subsection presents our findings from testing on **Llama 7B** (the Llama model with 7 billion parameters) during fine-tuning, comparing two clock speeds. These values result from increasing the HBM clock speed from 1593 MHz to 2619 MHz:

- **15%** training performance improvement when increasing HBM clock speed from 1593 MHz to 2619 MHz
- **>80%** GPU utilization on average for both clock frequencies
- **60%** peak HBM bandwidth utilization for 2619 MHz and more than **90%** capacity utilization for both clocks
- **10%** higher GPU power draw when increasing HBM clock speed from 1593 MHz to 2619 MHz
- **5%** higher GPU and HBM temperature when increasing HBM clock speed from 1593 MHz to 2619 MHz

Fine-tuning a 7B model requires more than 112GB of HBM capacity per GPU without any optimization. We applied several techniques to fine-tune these models when using a single NVIDIA H100 HGX system. One technique was model parallelism, which removed memory redundancies across the training implemented by DeepSpeed. We applied DeepSpeed ZeRO-1 model parallelism where the optimizer states are partitioned across the GPUs, while the gradients and parameters are replicated across all the GPUs. The figures below show characteristics of the model running on the NVIDIA HGX H100 system when we vary the HBM clock speed.

During fine-tuning, the performance (throughput) increased by 15% for Llama 7B with an approximately 65% increase in HBM clock speed, from 1593 MHz to 2619 MHz. See Figure 6. The empirical result of using HBM highlights that the model is bandwidth-sensitive and would therefore benefit from Micron's HBM3E superior performance and efficiency at 4000 MHz.

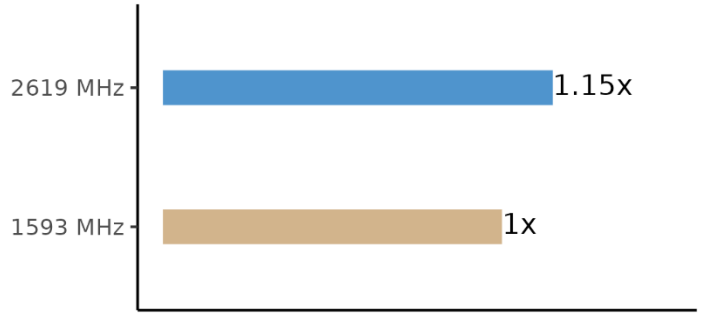


Figure 6. Performance improvement for fine-tuning the Llama 7B model by increasing HBM clock speed 65%

Figures 7 and 8 are complementary and show system utilization while fine-tuning the model. Figure 7 compares the GPU and HBM utilization, and Figure 8 compares the HBM capacity and bandwidth utilization. While GPU utilization is high (more than 80% on average for both memory clock speeds), HBM bandwidth is close to half the maximum capacity of the system on average. The peak bandwidth achieved was around 2 TB/s out of 3.3 TB/s for 2619 MHz. Conversely, the capacity is fully utilized, which shows that the workload is constrained in capacity. With twice the number of through-silicon vias (TSVs) than current HBM3 shipping solutions and an industry-leading data rate greater than 9.2 GB/s, Micron's HBM3E offers 50% more capacity with an 8-high 24GB cube allowing not only training at higher precision and accuracy but also higher system-level performance with increased bandwidth. As shown in the figure below, HBM at 1593 MHz shows a higher bandwidth percentage but it's relative to the maximum bandwidth achieved with that clock speed, which is inherently lower than what could be achieved using a higher clock speed. The higher HBM clock speed of 2619 MHz is more efficient as it enables faster fine-tuning, and we expect it to be even more efficient with Micron's HBM3E.

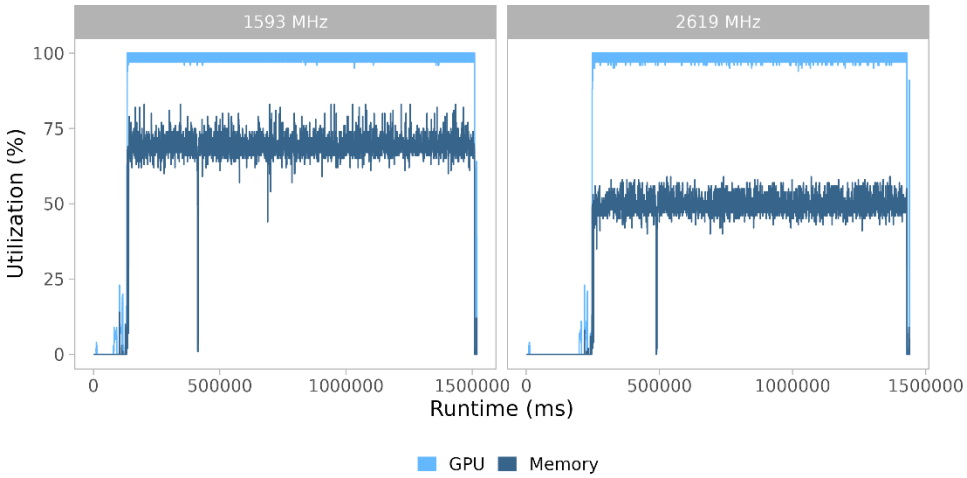


Figure 7. GPU and HBM utilization in 20ms intervals during Llama 7B fine-tuning using different HBM clock speeds

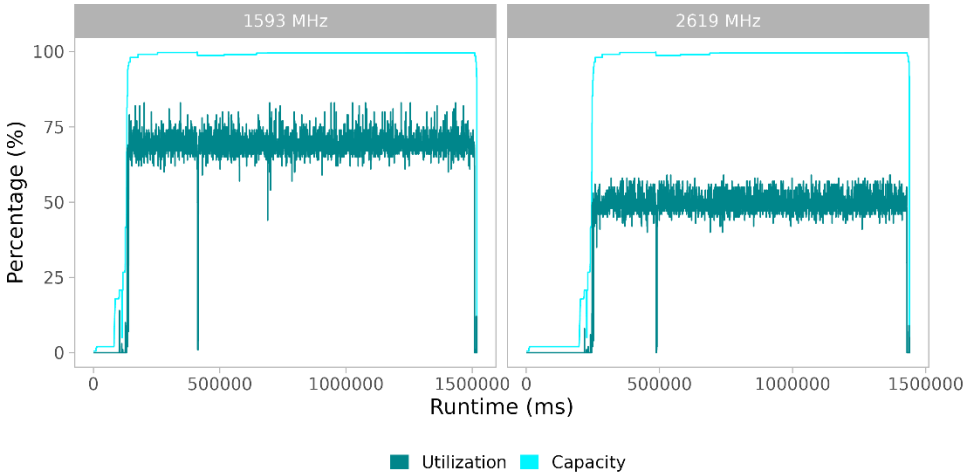


Figure 8. HBM capacity and bandwidth utilization in 20ms intervals during Llama 7B fine-tuning using different HBM clock speeds

Figures 9 and 10 show the power draw (or consumption) and temperature of the HGX while running the model. Even though power consumption at 2619 MHz is close to the power threshold of 700W, the temperature is far below the temperature threshold. Essentially, fine-tuning the model at 2619 MHz does not impose any thermal violations to the system. Higher memory clock speeds increase the consumption of both metrics, specifically a 10% increase in power draw and a 5% higher HBM temperature for 2619 MHz. Despite the increase in power consumption, the performance scaling achieved by increasing the memory clock yields 7% more FLOPS (floating point operations per second) per watt. As a result, fine-tuning the model at a higher clock speed is more power efficient than running it at a lower memory clock speed.

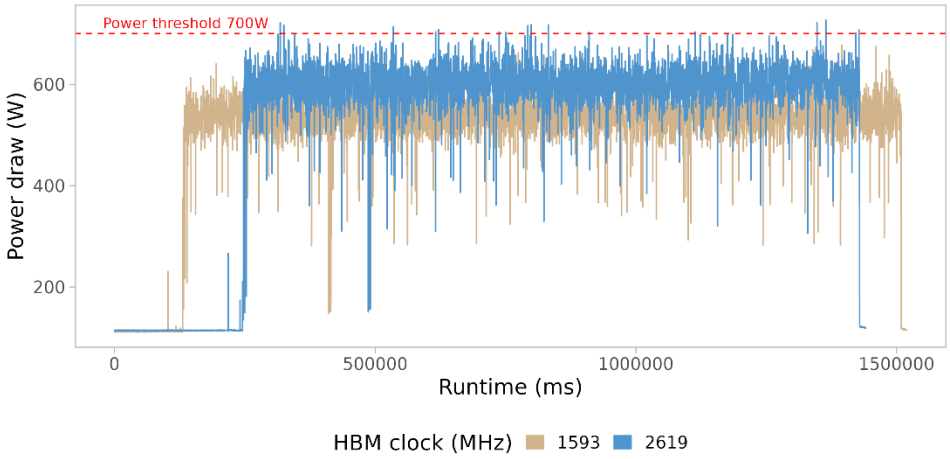


Figure 9. GPU power draw in 20ms intervals during Llama 7B fine-tuning using different HBM clock speeds

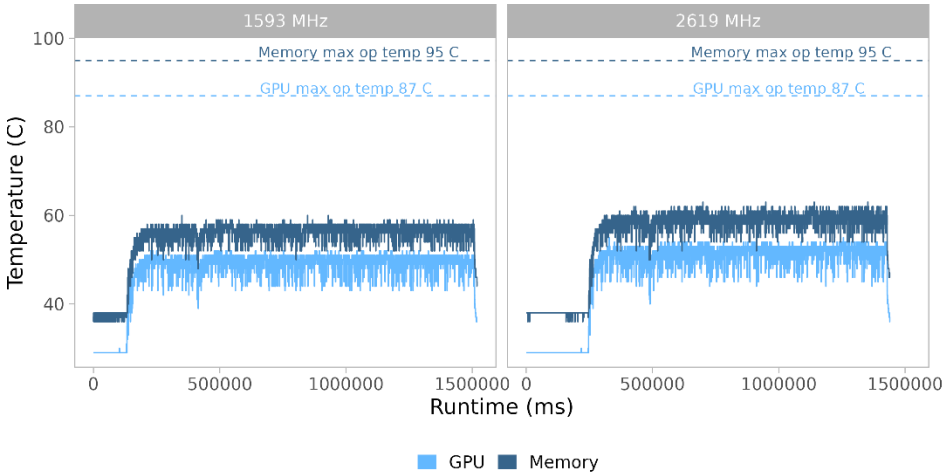


Figure 10. GPU and HBM temperature in 20ms intervals during Llama 7B fine-tuning using different HBM clock speeds

Micron's HBM3E energy-efficient data path, combined with process innovations, reduces thermal impedance. Micron HBM3E consumes 30% lower power than the competition on a typical workload. By applying an advanced data eye mask design, Micron HBM3E delivers system signal and power integrity improvements as well. All these factors improve LLM fine tuning performance.

Llama 65B

This subsection presents our findings from testing on Llama 65B (the Llama model with 65 billion parameters) during fine-tuning, comparing two clock speeds and using CPU offload (for model parallelism). These values result from increasing the HBM clock speed from 1593 MHz to 2619 MHz:

- **3.3%** performance improvement when increasing HBM clock speed from 1593 MHz to 2619 MHz
- **63%** GPU and 20% HBM utilization on average for 2619 MHz
- Fully utilized HBM capacity in either case
- **3%** higher GPU power draw on average when increasing HBM clock speed from 1593 MHz to 2619 MHz
- Similar HBM temperature
- **90%** host DDR memory utilization and 50% CPU utilization on average for either clock frequencies

Because the Llama 65B is nearly 10 times the size of the Llama 7B model, to fine-tune it, we applied advanced techniques to reduce its resource use. For example, due to the HBM capacity in the system, it was necessary to shard the parameters, gradients and optimizer states on all GPUs by applying DeepSpeed ZeRO-3. Moreover, we offloaded data to the host CPU, which meant some data would be stored in the host DDR memory and sent to the GPU when needed for the computation. In this scenario, the CPU also performed a few computations to increase the efficiency of fine-tuning and achieve model parallelism. This approach incurs more overhead but allows LLMs to run in a resource-constrained environment. The context length was also reduced from 512 tokens to 256 tokens to reduce constraints on memory. As shown in Figure 11, we measured only a 3.3% increase in performance (or higher throughput) for Llama 65B with an approximately 65% increase in HBM clock speed, from 1593 MHz to 2619 MHz. This small percentage increase in performance is mostly due to the CPU offload, where the interconnect bandwidth between CPU and GPU is less than the interconnect bandwidth between multiple GPUs, slowing down the overall computation.

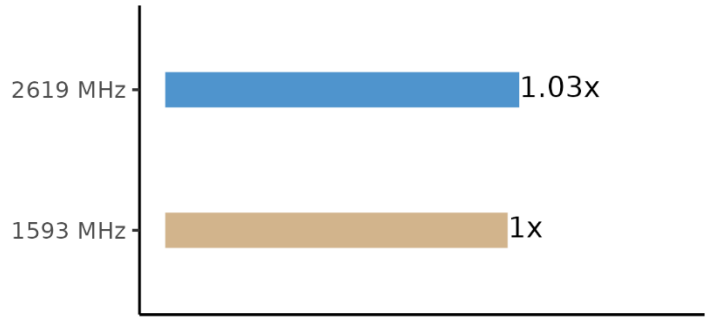


Figure 11. Performance improvement for fine-tunning Llama 65B model by increasing HBM clock speed 65%

Figures 12 and 13 show the GPU and HBM utilization. Because HBM clock speed does not fundamentally affect the metrics when CPU offload is enabled (as observed for the performance), we only show the metrics when running with the highest HBM clock speed (2619 MHz). However, we show both clock speed performance for power and temperature. We observed high peak utilization of GPU processing power; however, the average is only 63%. The average utilization is even lower for HBM bandwidth at only 20%. On the other hand, capacity is still a limitation as we observed it was fully utilized during the fine-tuning. Even though offloading drops utilization of the GPU, it's a powerful technique that allows the fine-tuning of such large models using fewer resources.

Micron's HBM3E higher memory capacity enables more data to be processed in the GPU, therefore leveraging its memory bandwidth to accelerate the computation. Figures 14 and 15 show the power and temperature, respectively, of the NVIDIA HGX H100 while running the model. Both HBM clock speeds present similar power consumption and temperature per GPU. Higher HBM speed only draws 3% more power on average. It clearly indicates the effects of dropping the average GPU use because of CPU offload.

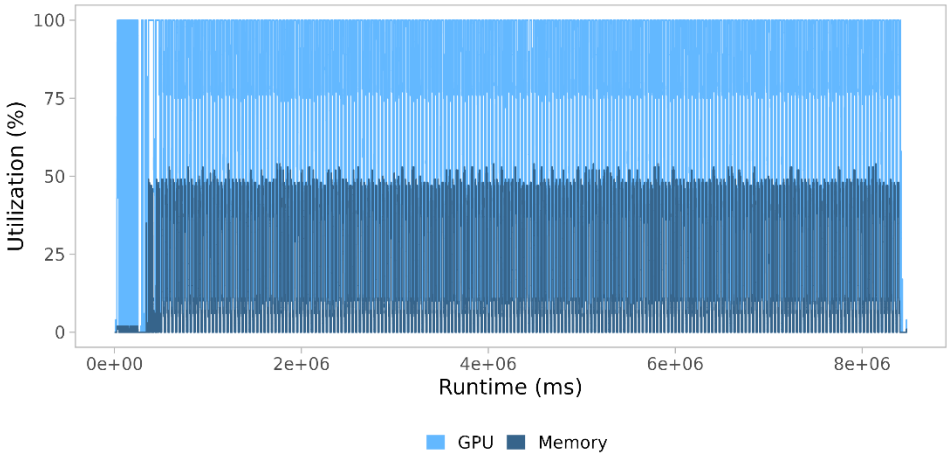


Figure 12. GPU and HBM utilization in 20ms intervals during Llama 65B fine-tuning and CPU offload

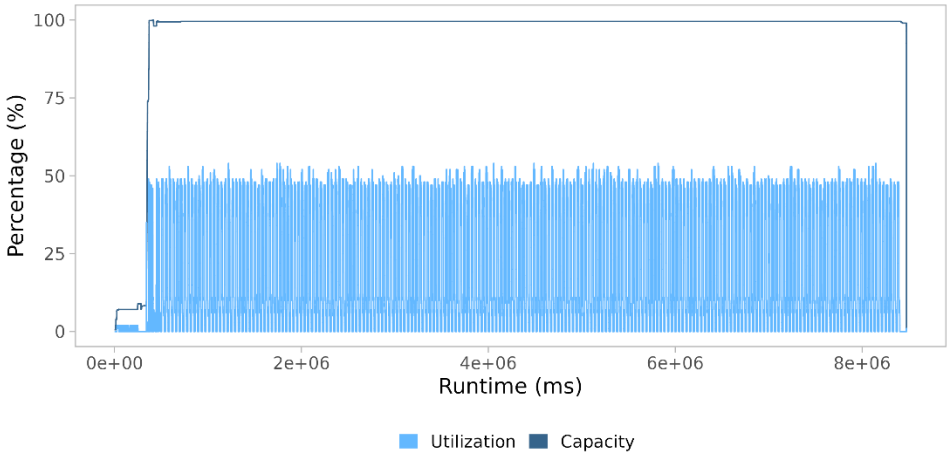


Figure 13. HBM capacity and bandwidth utilization in 20ms intervals during Llama 65B fine-tuning and CPU offload

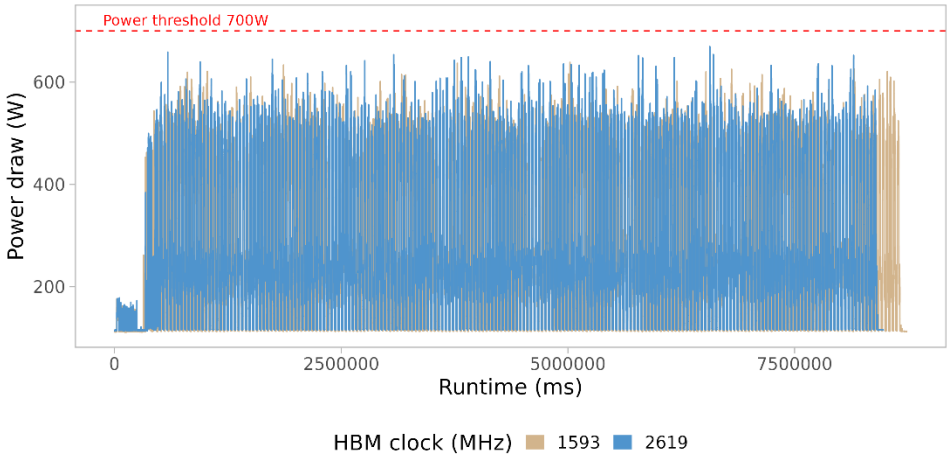


Figure 14. GPU power draw in 20ms intervals during Llama 65B fine-tuning using different HBM clock speeds and CPU offload

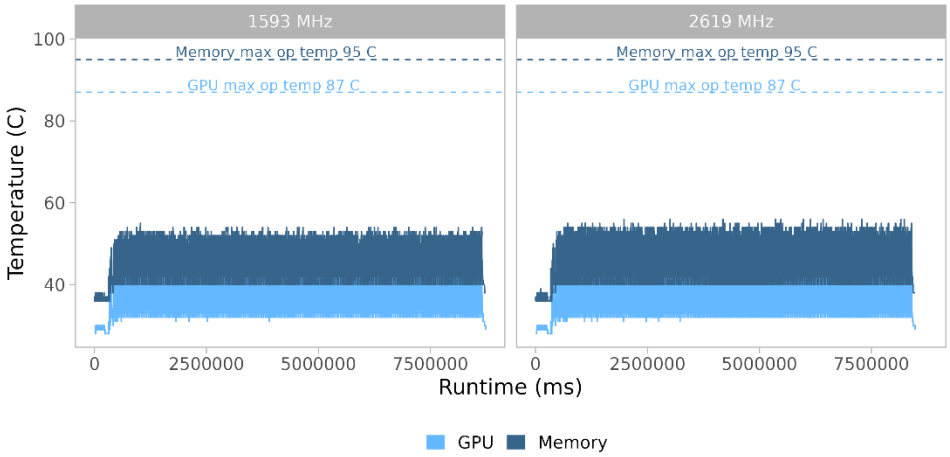


Figure 15. GPU and HBM temperature in 20ms intervals during Llama 65B fine-tuning using different HBM clock speeds and CPU offload

As part of the data and computation that are offloaded to the CPU, it's important to check the host utilization. Figure 16 shows the percentage utilization of the CPU and DDR memory in terms of capacity. Figure 16 shows high utilization of the host resources, with almost 50% CPU utilization since it's alternated with GPU computation and a peak memory utilization of 90% (1.8TB of DDR5) allocated memory capacity. This result suggests that the use of high-capacity DIMMS is necessary to fine-tune LLMs when data is offloaded to the host DDR. Micron's diverse portfolio of high-capacity DIMMs, such as Micron's 96GB and 128GB RDIMMs, offer exceptional cost benefits to power AI computation.

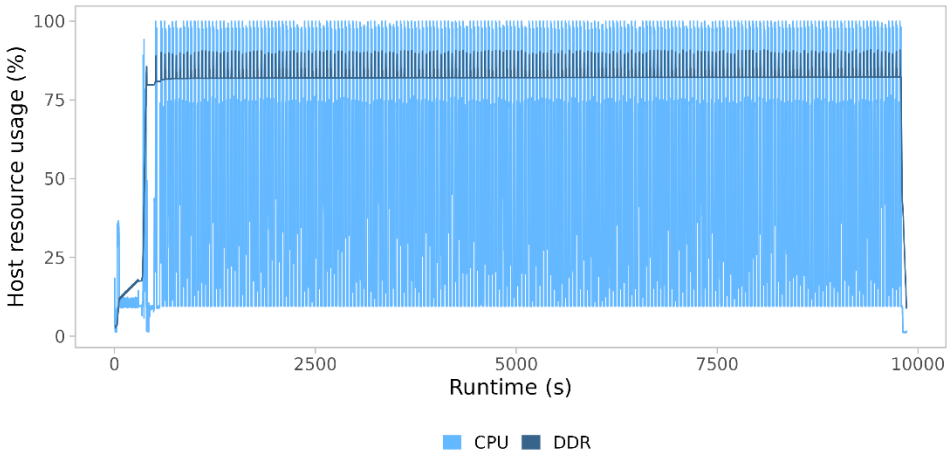


Figure 16. Host CPU and DDR memory utilization in 1s interval during Llama 65B fine-tuning and CPU offload

Comparative sensitivity analysis for ML models

A more comprehensive approach and broader way to compare several available machine learning (ML) models is to analyze their sensitivity to HBM speed. This analysis allows us to deepen our understanding of the models and project their performance for different scenarios. In this section, we compare the following classes of ML models:

Computer vision (CV)

Computer vision (CV) is a field within AI that enables systems to extract meaningful information from visual data, including videos and digital images. Key tasks for CV models include object segmentation, classification, detection and tracking, each contributing to the system's understanding of visual content. Notable models like single-shot multibox detector (SSD) [17] excel in object detection while ResNet [18] performs image classification, and Mask R-CNN [19] is known for its precision in image segmentation. These models are built on the architecture of convolutional neural networks (CNNs), which consist of convolutional layers paired with subsampling layers that learn image features through the application of filters, also known as kernels.

Recommendation systems (RecSys)

Recommendation systems (RecSys) are networks that use deep learning concepts to personalize user experiences. These systems are trained to identify the unique characteristics of users and products by analyzing historical data, including past decisions and preferences derived from user interactions. There are various types of recommendation systems, each with its own approach. For instance, collaborative filtering algorithms make suggestions based on collective user behavior patterns. Content-based filtering uses item attributes to recommend similar items that align with a user's preferences. Additionally, context-based filtering incorporates situational information from the user to enhance the relevance of recommendations. Advanced models such as neural collaborative filtering (NCF) [20] and the deep learning recommendation model (DLRM) [21] are a couple examples of this model class, offering more nuanced and predictive capabilities.

Natural language processing (NLP)

In addition to LLMs — such as Llama [6][7]and BLOOM [22] that use GPT-like transformer decoder-only architectures for text generation — there are also models like Bidirectional Encoder Representations from Transformers (BERT) [23] and Google Neural Machine Translation (GNMT) [24]. BERT is an encoder-only transformer architecture that can read an entire sequence of words at once, allowing it to learn the context of a word based on the words close to it. BERT combines masked language modeling — which masks words in a sentence and then attempts to predict the original value based on the context provided by the nonmasked words — with next-sentence prediction to anticipate subsequent text. This capability makes BERT versatile for tasks including classification, entity recognition and sentiment analysis. On the other hand, GNMT leverages a deep long short-term memory (LSTM) network with encoder-decoder layers using residual connections as well as attention connections from the decoder part. GNMT translates the entire sentence in one go by encoding the semantics of the sentence, which provides a more fluent translation.

Performance of ML models on HBM

Figure 17 depicts the performance of several distinct models on HBM. The size of each bubble represents the average HBM bandwidth utilization of an ML model, while the axes display the read and write percentages of HBM accesses.

Our analysis reveals that most ML models usually have a higher proportion of read operations (60%) than writes (40%), since most of the calculations are matrix multiplications. Interestingly, the Llama 33B and Llama 65B models show lower HBM bandwidth utilization than other models. This finding is attributed to the use of CPU offloading, which reduces the average GPU resource use to 63%, as observed in our experiments with Llama 65B.

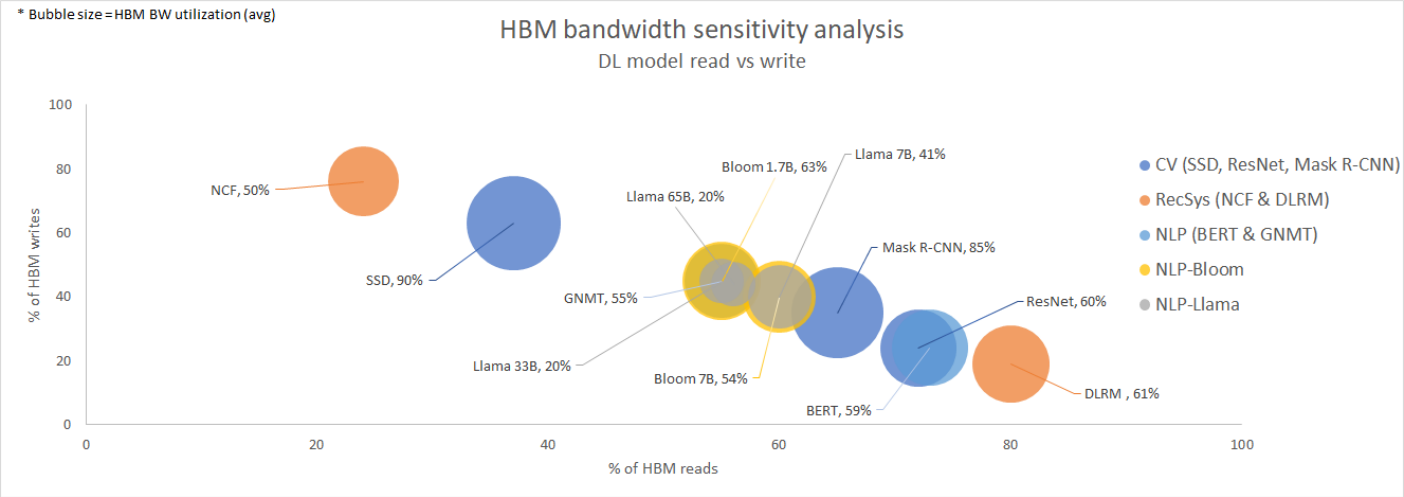


Figure 17. Comparative bubble chart for performance between several ML models

GPU power analysis of ML models during fine-tuning

Figure 18 presents a GPU power analysis as a function of the HBM state during fine-tuning. The size of each bubble represents the average GPU power consumption for each model, while the *x*- and *y*-axes correspond to the average HBM temperature and utilization, respectively. Notably, the BERT model exhibits lower power use than other models with comparable utilization, a finding that can be attributed to its reduced temperature and resource demands.

Even though Llama 65B and Llama 33B models show lower HBM utilization on average due to offloading, the average power consumption for these models is moderate because the HBM capacity is fully utilized. By leveraging the cutting-edge features of Micron's HBM3E technology, these models can achieve enhanced power and thermal efficiency, resulting in a decrease in overall system power requirements.

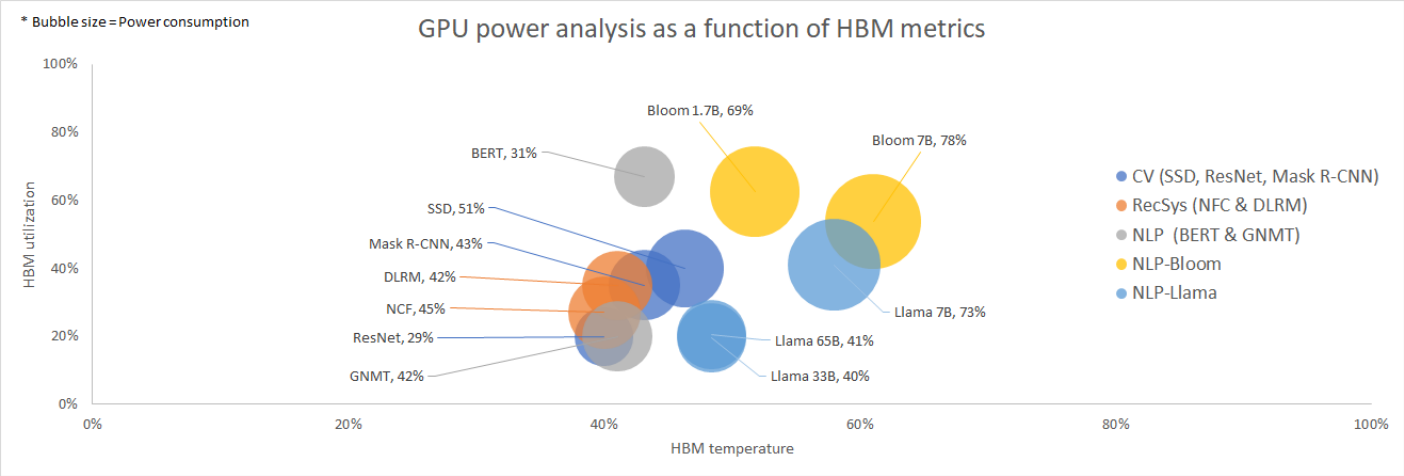


Figure 18. Comparative bubble chart for power analysis as a function of HBM utilization and temperature between several ML models

Llama 65B using QLoRA

This subsection presents our findings from testing on Llama 65B using QLoRA. These values result from increasing the HBM clock speed from 1593 MHz to 2619 MHz:

- **18%** performance improvement when increasing HBM clock speed from 1593 MHz to 2619 MHz
- **80%** GPU utilization on average for 2619 MHz
- **56%** HBM utilization on average for 2619 MHz
- **10%** higher GPU power draw on average when increasing HBM clock speed from 1593 MHz to 2619 MHz
- Consistently higher resource utilization than CPU offload

To fine-tune Llama 65B in our setup, we applied DeepSpeed ZeRO-3 and CPU offload — where parameters, gradients and optimizer states are partitioned on all GPUs instead of replicated and some computations are done on the host side — to avoid out-of-memory problems due to HBM capacity constraints. However, we previously observed that HBM clock variation showed no meaningful improvement in model performance (less than 3%) and CPU utilization was about 50% on average and achieved peak use of 100% during the optimizer computation parameters.

By implementing quantization methods like QLoRA, the Llama 65B model can run without any DeepSpeed optimizations, where all data is replicated across all GPUs. This approach allows for an increase in batch size during fine-tuning, enhancing overall efficiency. A larger batch size contributes to reduced training time, as it will pass through the dataset faster and perform fewer updates to the models. Provided that the long-learning-rate hyperparameter is adjusted accordingly, the model's performance shouldn't be compromised.

Empirical results demonstrate that using QLoRA on Llama 65B facilitates an 18% performance boost, as depicted in Figure 19. This boost corresponds with an approximately 65% increase in HBM clock speed, from 1593 MHz to 2619 MHz.

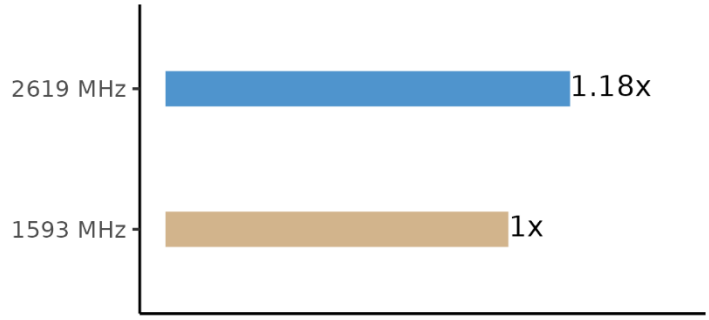


Figure 19. Performance improvement for fine-tunning Llama 65B model by applying QLoRA optimization and increasing HBM clock speed 65%

Figures 20 and 21 show the power consumption comparison when doing CPU offload and QLoRA (where there is no offloading). Performing offload, both HBM clock speeds present similar power consumption per GPU. We found only 3% more power consumption on average for higher memory speed due to the drop in GPU utilization because of the communication between CPU and GPU. On the other hand, QLoRA allows use of a GPU without offload and an increase in batch size, so both clock speeds draw more power on average than the offload execution, with higher memory speed consuming 10% more power on average than the lower clock speed. GPU-only execution increases the power draw by almost 50% on average compared to the CPU offload execution presented earlier (see the Llama 65B section for more details).

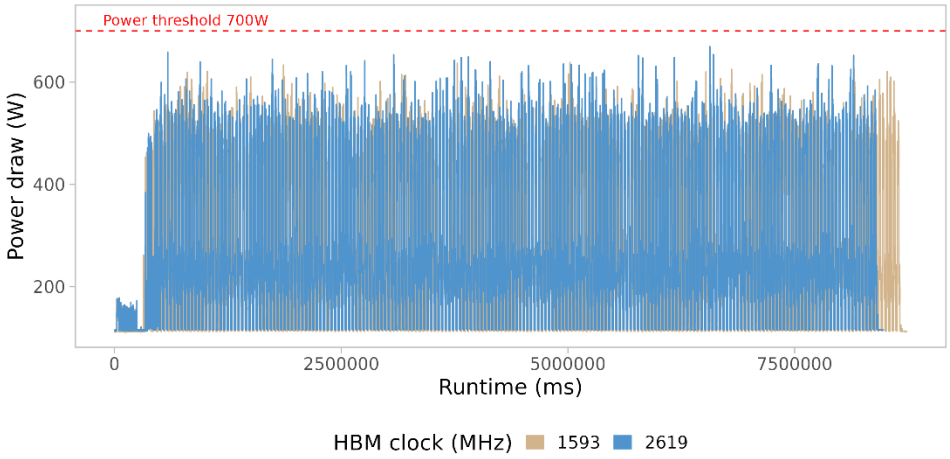


Figure 20. Power draw for fine-tuning Llama 65B using CPU offload in 20ms intervals

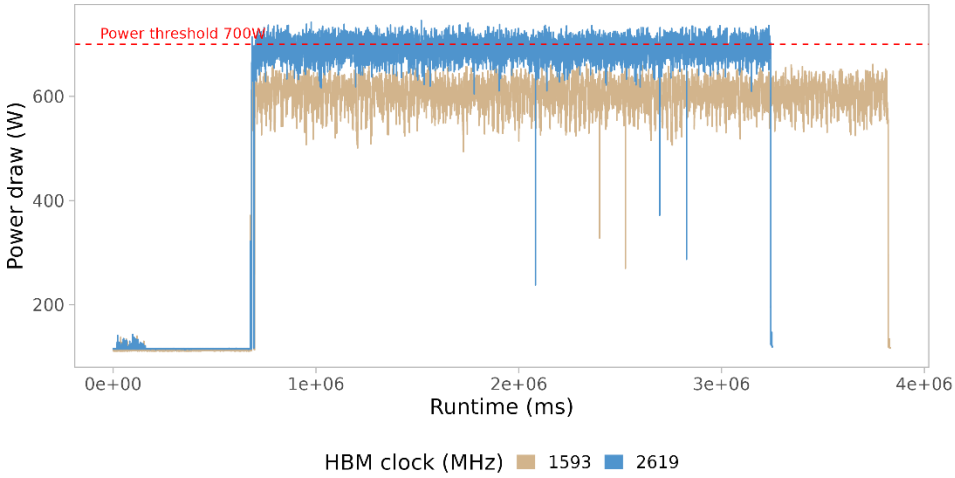


Figure 21. Power draw for fine-tuning Llama 65B using QLoRA in 20ms intervals

Figures 22 and 23 compare GPU utilization with and without offload. In both scenarios, we observe a high peak utilization of GPU processing power; however, using CPU offload, the average is 63%, whereas QLoRA increases it to almost 80% on average. These findings translate to 26% higher GPU utilization running a model without offload. While peak HBM bandwidth utilization is around 54% and 20% on average for the CPU offload, QLoRA increases peak utilization to 80% and the average to 56%. Recall that, in both scenarios, the capacity is fully utilized. In summary, performance using QLoRA is consistent with the observed performance of smaller models that fully ran on the GPU HBM without CPU offload.

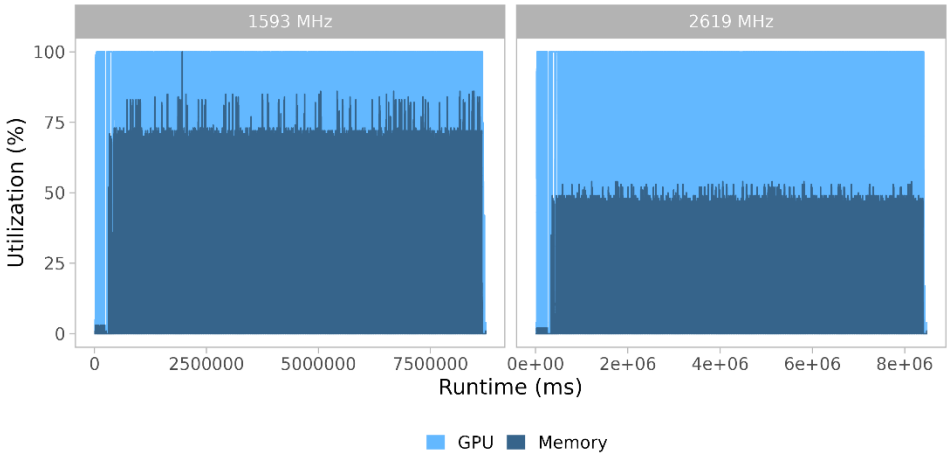


Figure 22. GPU and HBM utilization in 20ms intervals for fine-tuning Llama 65B using CPU offload

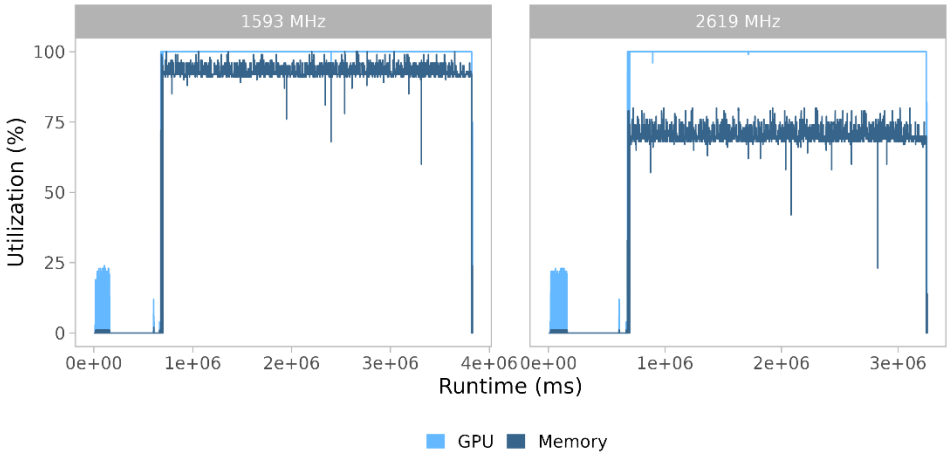


Figure 23. GPU and HBM utilization in 20ms intervals for fine-tuning Llama 65B using QLoRA

QLoRA Performance Analysis

Figure 24 contrasts performance of the models that use QLoRA with other models analyzed previously. Compared to the previous execution of Llama 65B using CPU offload with QLoRA, the HBM utilization for Llama 65B increases from 20% to 56%. It demonstrates that fine-tuning benefits from Micron's HBM3E bandwidth.

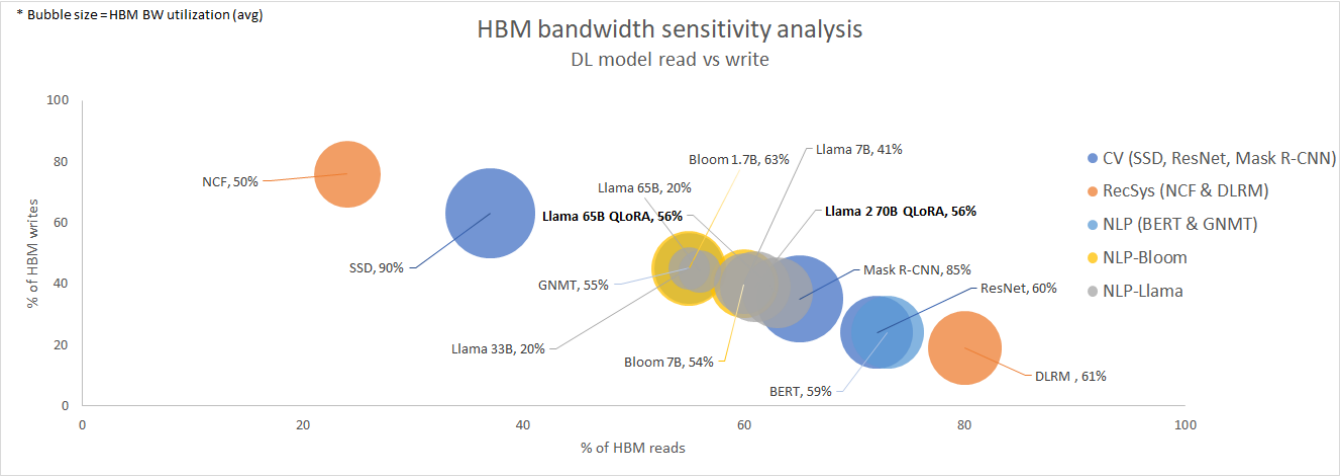


Figure 24. Comparative bubble chart for memory sensitivity between several ML models and QLoRA quantization

High-capacity DIMM analysis

We ran several experiments on high-capacity DIMMs for the fine-tuning execution to see how changing the host DDR memory subsystem configuration would affect CPU offloading performance. These tests involved varying the DIMM configurations within the system for two distinct models: Llama 33B and Llama 65B with CPU offload. We used Micron's 64GB and 96GB modules for comparison against a commercially available 128GB 3DS TSV-based module. The configurations were deployed with one and two DIMM modules per channel (DPC), resulting in a total system memory range of 2TB to 4TB.

Figures 25a and 25b show the runtime improvements for fine-tuning both Llama 33B and Llama 65B models. These improvements are shown as percentages and based on the setups indicated on the *x*-axis relative to the configurations shown on each panel. For instance, the first panel of Figure 25a — representing the 64GB-2DPC configuration for Llama 33B — serves as a reference point for all other *x*-axis configurations. When the *x*-axis matches the panel's configuration, it acts as the baseline, so no improvement is recorded (0%). Conversely, subsequent configurations like the 96GB-1DPC show a 2% improvement (or 2% lower runtime) compared to the 64GB-2DPC baseline.

We can quickly perceive from Figures 25a and 25b that the performance difference between each configuration is minor, suggesting that a more complex and costly memory module like the TSV-based one may be unnecessary. During fine-tuning, the Llama 65B model encountered out-of-memory (OOM) errors in configurations with less than 2TB of total system memory, underscoring the need for high-capacity DIMMs to support larger models (see Table 1). Notably, configurations using 1DPC consistently outperformed 2DPC, a finding that is attributed to the latter's reduction in memory speed and consequent decrease in available memory bandwidth for the workload.

To meet the need for high-capacity DIMMs that support AI workloads, Micron plans to offer a cost-effective solution for reducing the total cost of ownership for data centers. Micron's solution uses a 32Gb die-based 128GB RDIMM for better bandwidth and power efficiency. It provides an innovative die architecture choice for leading array efficiency and the densest monolithic DRAM die. By incorporating voltage domain and refresh management features, the power delivery network is optimized, leading to significant improvements in energy efficiency. Notably, this solution offers up to 24% better energy efficiency than its TSV-based counterparts.

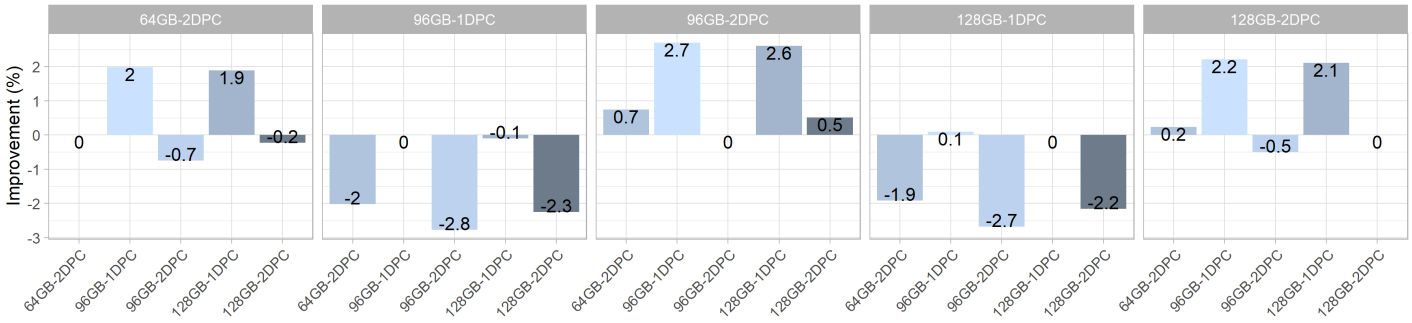


Figure 25a. Comparison of different configurations of DDR5 for Llama 33B

Note: Positive columns in both Figure 25a and 25b denote improvement of the *x*-axis configuration over the panel configuration.

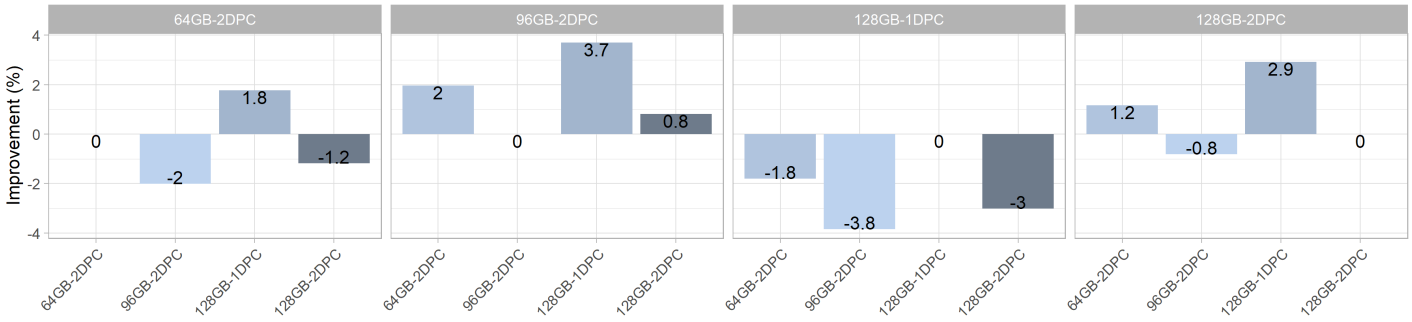


Figure 25b. Comparison of different configurations of DDR5 for Llama 65B

Table 3 summarizes results for execution of Llama 65B with offload on the HGX H100 platform across various DDR5 memory configurations. As previously mentioned, the average time difference between the memory setups is small, and 1DPC configurations consistently outperform 2DPC configurations. A notable observation is the inverse relationship between total system capacity and peak DDR utilization: As we increase the total system DDR capacity, the percentage of DDR usage relative to the total system capacity drops if the model size remains unchanged. Also, failure of the model execution with a 1.5TB setup indicates a minimum requirement of 256GB of DDR per GPU for the eight GPUs within the HGX H100 to successfully fine-tune a 65B model using offload. This finding demonstrates that the ratio between DDR and HBM should increase as we expand the model size.

Memory setup	System capacity	Capacity per GPU	DDR:HBM ratio	Peak DDR use	DDR GB/core provisioned	Avg. training time (s)
64GB 2DPC	2TB	256GB	3.2	90%	21	9242
96GB 1DPC	1.5TB	192GB	2.4	OOM	16	-
96GB 2DPC	3TB	384GB	4.8	60%	32	9427
TSV 1DPC	2TB	256GB	3.2	90%	21	9078
TSV 2DPC	4TB	512GB	6.4	45%	42	9352

Table 3. Summary for Llama 65B, with offloading model execution across various DDR5 memory configurations

Conclusion

LLMs have demonstrated remarkable proficiency on downstream tasks related to NLP tasks, a success largely attributed to adoption of transformer architecture coupled with advancements in GPU processing capabilities. As it's designed for parallel processing, transformer architecture relies heavily on efficient memory access to sustain optimal performance during fine-tuning. Moreover, the demands from extensive matrix multiplications and data transfers involved in the self-attention mechanism of the architecture make memory bandwidth and capacity a critical bottleneck.

As LLMs grow in scale and complexity, advancements in memory technology are essential to unlock their potential. Micron's HBM3E offers a compelling solution for meeting the memory performance requirements of these workloads. Compared to previous generations of HBM, Micron's HBM3E shows key improvements in a stacked architecture — alongside a significant increase in bandwidth, capacity and energy efficiency (performance per watt) — that position it to meet the rigorous demands of modern GPUs and complex AI workloads. Consequently, our HBM3E is expected to play a critical role in enhancing the overall performance of GPUs and LLMs.

References

[1] Narayanan, Deepak, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand et al. "Efficient large-scale language model training on GPU clusters using Megatron-LM." In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-15. 2021.

[2] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[3] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving language understanding by generative pre-training." (2018).

[4] Rajbhandari, Samyam, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. "Zero: Memory optimizations toward training trillion parameter models." In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-16. IEEE, 2020.

[5] https://github.com/tatsu-lab/stanford_alpaca

[6] Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière et al. "Llama: Open and efficient foundation language models. CoRR, abs/2302.13971, 2023. doi: 10.48550." arXiv preprint arXiv.2302.13971.

[7] Touvron, Hugo, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov et al. "Llama 2: Open foundation and fine-tuned chat models." arXiv preprint arXiv:2307.09288 (2023).

[8] Dettmers, Tim, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. "QLoRA: Efficient finetuning of quantized LLMs." arXiv preprint arXiv:2305.14314 (2023).

[9] Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "LoRA: Low-rank adaptation of large language models." arXiv preprint arXiv:2106.09685 (2021).

[10] Weng, Lilian. Large transformer model inference optimization. Lil'Log. https://lilianweng.github.io/posts/2023-01-10-inference-optimization/. (2023).

[11] https://www.nvidia.com/en-us/data-center/h100/

[12] https://www.deepspeed.ai/tutorials/zero/

[13] https://pytorch.org/

[14] https://www.deepspeed.ai/

[15] Fedus, William, Barret Zoph, and Noam Shazeer. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity." Journal of Machine Learning Research 23, no. 120 (2022): 1-39.

[16] Dr Alan D. Thompson, LifeArchitect.ai (Jan/2024).

[17] Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "Ssd: Single shot multibox detector." In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, pp. 21-37. Springer International Publishing, 2016.

[18] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[19] He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In Proceedings of the IEEE international conference on computer vision, pp. 2961-2969. 2017.

[20] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. "Neural collaborative filtering." In Proceedings of the 26th international conference on world wide web, pp. 173-182. 2017.

[21] Naumov, Maxim, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang et al. "Deep learning recommendation model for personalization and recommendation systems." arXiv preprint arXiv:1906.00091 (2019).

[22] Le Scao, Teven, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné et al. "Bloom: A 176b-parameter open-access multilingual language model." (2023).

[23] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[24] Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

micron.com/hbm