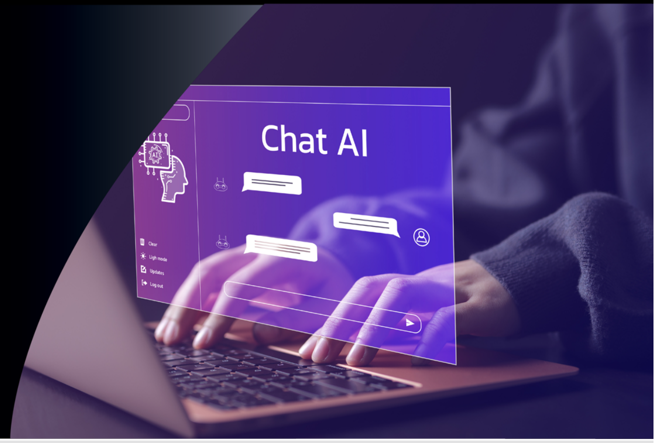
ENGINEERING REPORT – LARGE LANGUAGE MODEL INFERENCE

Analyzing LLM performance: The impact of high-bandwidth memory on model inference



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

The rapid evolution of large language models (LLMs) has been marked by an exponential increase in their size, where models with tens of billions of parameters enhance their capabilities but also place ever greater demands on memory. These models rely on enormous data sets to learn and make predictions, and this need requires significant memory capacity and bandwidth to effectively process the sheer volume of information. These demands are particularly pronounced during the inference phase, presenting a complex challenge for deployment on platforms with resource constraints. Consequently, as LLMs continue to grow, the need for advanced memory technologies has become key to unlocking the enormous potential of AI inference. Micron's innovative portfolio of high-bandwidth memory (HBM) products provides an effective solution to LLM inference challenges.

This report includes an analysis of LLM inference performance on the NVIDIA HGX H100 platform using HBM, aiming to benchmark HBM's effectiveness in handling the intense computational demands of LLMs. The testing details — which include system configurations, optimization techniques and performance metrics — are documented to educate readers and offer insight into the efficiency and capabilities of high-bandwidth memory in real-world AI applications. 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

>20%

21% increase in total throughput with different HBM clock speeds for larger batch sizes

Micron HBM3E increases the total bandwidth by >40% in H200 compared to H100, further enhancing the inference throughput. [a]

~80%

Higher precision (FP16) and accuracy with 80% more HBM capacity

The Llama 2 70B model quantized to FP16 requires more HBM capacity than is available from H100 GPUs (80GB per GPU). Using Micron's HBM3E with Hopper H200 (141GB) results in an ~80% increase in HBM capacity. Compared to Hopper H100 GPU, this meets the model requirements.[b]

>256

Micron HBM3E 24GB capacity to support larger batch sizes that are greater than 256

Micron 24GB-36GB HBM3E increases per-placement capacity by 50% and addresses high-capacity needs of LLMs for larger batch sizes and context length.[c]

~3x

More throughput for an INT4- quantized model over the execution of an FP16 model offloaded to CPU

Larger batch sizes require more HBM capacity and proportionally increase the throughput. Furthermore, using a quantized model that requires less memory capacity means that more clients can be served simultaneously.

[a] Comparing total throughout measured in tokens per second at two HBM clock speeds, 1593 and 2619 MHz. Measurements taken on the NVIDIA HGX H100 platform.

[b] Llama 2 70B in FP16 precision requires 140GB of HBM3 capacity to hold the model weights. H200 has 6 HBM sites while H100 has 5 HBM sites.

[c] Context length is the user input plus the total number of generated tokens.

LLM inference explained

When a user prompts an LLM, by either asking a question or typing a phrase, it processes this information and then outputs a response. This entire process, from receiving the input to generating an output, is called **inference**. Basically, an LLM can be considered a highly knowledgeable source that delivers a comprehensive response to user queries. Additionally, the model can request more details to improve the response, further tailoring the output to a user's specific needs.

One way to measure inference performance is by a model's total **throughput**, which indicates how much work the LLM can do in a certain amount of time. Specifically, it's about how many tokens (pieces of text, which can be words or parts of words) the model can process per second. **Latency,** which is how long it takes the model to respond to input, can also be used to measure performance. However, for this report, we focus only on the throughput of the LLM.

Higher throughput often means faster inference times, but not always. A higher throughput allows the model to process more requests, and handling numerous requests quickly is critical for applications that require real-time responses or manage a large volume of requests. Sometimes higher throughput can be achieved but at the expense of higher latency (or time to first token). In that case, performance is sacrificed in one area to elevate it in another. Additionally, to speed up the inference, tensors can be stored in memory as cache but at the expense of the HBM capacity requirement. This cache increases as a function of the number of requests and their size.

For LLMs to achieve high performance, especially on platforms with finite resources, they must have memory solutions that can accommodate the enormous parameter sets of LLMs (often in the tens of billions) and the cache created, as performance relies heavily on the hardware used and the optimizations applied (such as the type of model quantization). In this context, advancements in memory technology are key to enabling LLMs to operate efficiently and effectively, ensuring that the strides made in LLM architectures and graphics processing unit (GPU) capabilities translate into outstanding performance outcomes for real-world inference applications across many industries, including health sciences, education and gaming.

How LLMs generate tokens

The transformer architecture [1][d] has revolutionized LLMs by enabling highly parallel training, thanks to its sequence parallelism feature. However, inference presents a greater challenge due to the autoregressive nature of the process. The term "autoregressive" simply means that the computation of each token depends on the previous tokens generated. This sequential dependency complicates the parallelization of this process. LLMs generate responses using a two-step process:

- **Prefill phase**: Tokens from the input prompt are processed in parallel. Usually, this phase is dependent on the GPU (where processing power is measured in floating-point operations per second, or FLOPs).
- **Decoding phase**: Output tokens are predicted sequentially in an autoregressive manner. Each newly predicted token is appended to the input sequence and reintroduced to the model to generate the next token. This phase is memory-bound, meaning that it is highly dependent on both memory capacity and bandwidth.

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

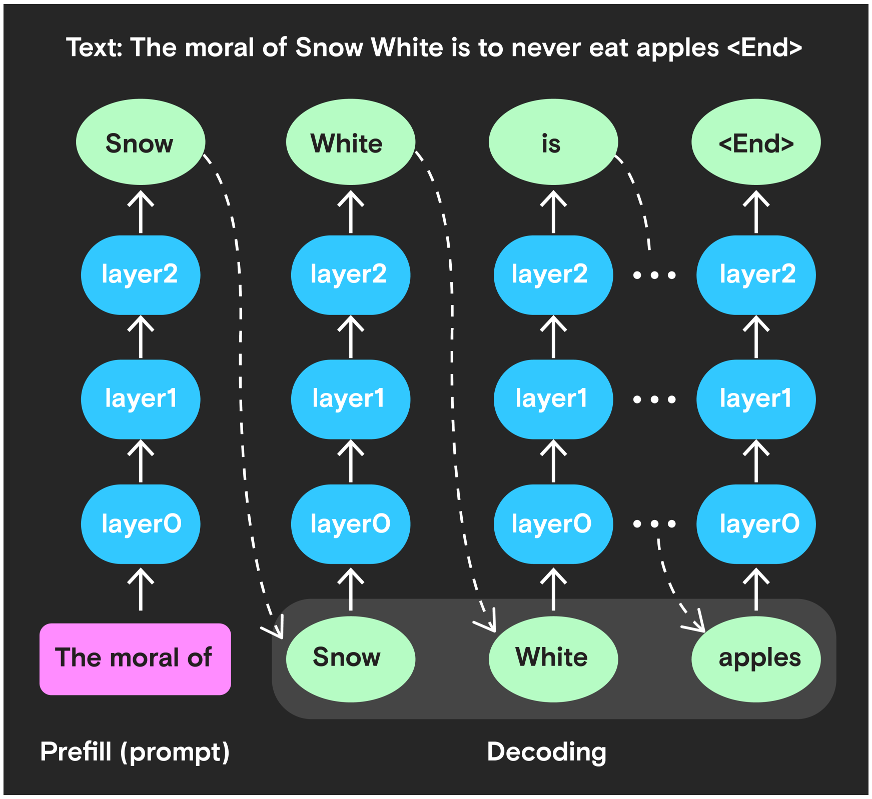


Figure 1: Inference workflow with prefill and decoding phases

This two-step process is illustrated in Figure 1. The process initiates with an input or prompt (in this case, **The moral of**).

During the prefill phase, the model processes the input and outputs the initial token (in our example below, **Snow**).

This token is then appended to the input, and the decoding phase commences, sequentially generating tokens.

This phase continues until the model either predicts a predetermined number of tokens (output length) or encounters an end-of-sequence token, denoted as **<End>** [2].

Challenges for generative inference

The process of generative inference presents several challenges:

- **Model parameter size**: Parameters are large enough to fit on a single GPU. For example, the Llama 2 70B model [4] requires approximately 140GB of HBM capacity per GPU to accommodate the model weights in half precision (FP16). This requirement exceeds the memory capacity of the Hopper H100 GPU.
- **Attention mechanism computation**: The cost to compute the attention mechanism scales quadratically with the length of the sequence. Mathematically, if N represents the sequence length, the computational cost can be expressed as N^2.
- **Key and value tensor storage**: During the decoding phase, the *key* and *value* tensors — also referred to as the KV cache — must be stored in memory. The size of the KV cache increases with larger batch sizes and greater context lengths. For instance, for a multi-head attention model with more than 500 billion parameters, a batch size of 512 and a context length of 2048, the KV cache can reach a total of 3TB [5].

These factors contribute to the complexity of generative inference, necessitating advanced hardware and optimized algorithms to efficiently manage resources and computation.

On top of the capacity constraint, this enormous amount of data (for example, 3TB) generates significant overhead when dealing with the tensors. For smaller batch sizes and shorter sequence lengths, the key concern is the time required to load the model. Conversely, for larger batch sizes and longer sequence lengths, the loading of the KV cache dominates. Operating large models at large batch sizes is essential for optimizing cost and performance. However, this approach necessitates a larger KV cache, which consequently requires a greater number of GPUs to execute the model.

Different approaches to inference

The inference process is fundamentally constrained by the GPU's memory capacity [2]. To address this limitation, several strategies have been devised to mitigate memory constraints, particularly to speed up the decoding phase or target to local deployments and consumer-grade devices. The following methods can accelerate inference or reduce costs:

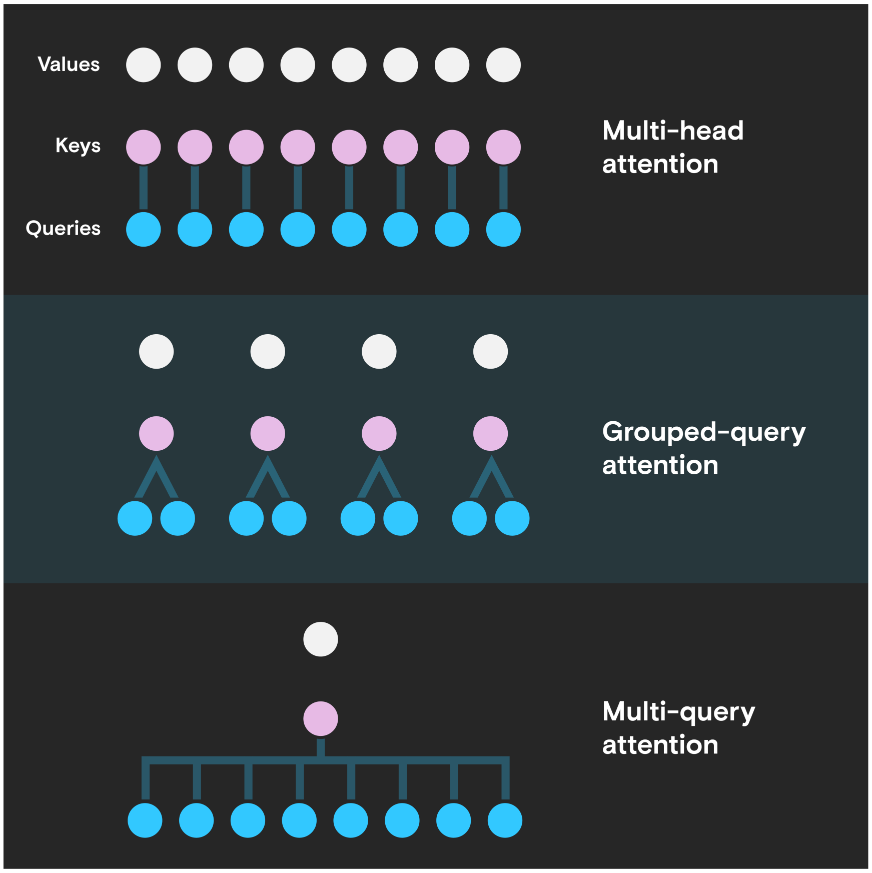
- **Smart parallelism**: This method involves distributing the model across multiple GPUs to scale up the model's capabilities [6][8].
- **Model offloading**: This technique sends inactive data to the CPU and/or NVMe storage and reads it back as needed [6][7][8].
- **Smart batching**: This approach consolidates consecutive requests to fully utilize both high-bandwidth memory and GPU resources [9][10].
- **Improved model architecture**: This strategy includes making changes to the model's architecture, such as optimizing attention layers to speed up the decoding phase [3][10][11].
- **Compression**: Techniques like pruning [12], distillation [13] and quantization [14][15][16] are applied to reduce the model requirements (high-bandwidth memory and processing).

Microsoft's DeepSpeed library [17][18] implements several improvements for the training and inference of LLMs. Its ZeRO-Inference [6] feature adapts and optimizes the previous ZeRO-Infinity techniques for model inference run on GPUs by hosting the model weights in CPU or NVMe memory, effectively hosting no (zero) weights in the GPU.

This feature works by anchoring the entire model weights in CPU or NVMe storage and streaming the weights layer by layer into the GPU for inference computation. Once a layer is computed, the outputs are retained in GPU memory to serve as inputs for the subsequent layer, while the memory used by the layer weights is freed for use by the next layer.

This method ensures efficient computation for throughput-oriented inference tasks — despite the latency associated with fetching model weights from the CPU or NVMe storage over PCIe interconnect — because it can use the majority of GPU memory to support a substantial number of input tokens in the form of long sequences or large batch sizes.

Figure 2 shows three types of attention mechanisms [3] often used to address challenges to generative inference.



Includes *N* heads for query, key and values

Shares single key and value heads for a group of queries

Shares single key and value heads across all queries

Figure 2: Three types of attention mechanisms often implemented by LLM models

Multi-head attention

N heads for query, key and values

Instead of using a single function for keys, values and queries, the transformer model uses **multi-head attention** [1], which projects keys, values and queries ***H*** times with different learned linear projections. On each projected version of the keys, values and queries, the attention function is executed in parallel and then appended to project the result. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. However, this mechanism requires loading decoder weights and attention keys and values at each processing step, leading to significant memory consumption.

Grouped-query attention

Single key and value heads for a group of queries

In general, **grouped-query attention** [3] introduces a structure that divides query heads into ***G*** groups, with each group sharing a single key and value head. This method strikes a balance between optimizing performance and maintaining the quality associated with multi-head attention. Grouped-query attention with a single group (a single key and value head) is equivalent to a multi-query approach. Conversely, when each group equals the number of heads, it mirrors multi-head attention.

Multi-query attention

Single key and value heads across all queries

A variant of multi-head attention is **multi-query attention** [20], where multiple query heads share a single set of keys and values. This approach significantly reduces memory bandwidth and capacity requirements, enhancing decoder inference speed. However, this approach may result in some loss of quality and training stability.

In comparison to multi-head attention, multi-query attention reduces the KV cache to a single key and value head, decreasing the amount of data loaded by a factor of H [3]. On the other hand, as the number of heads generally scales with model size, grouped-query attention maintains a proportional decrease in bandwidth and capacity as the model size grows.

System setup

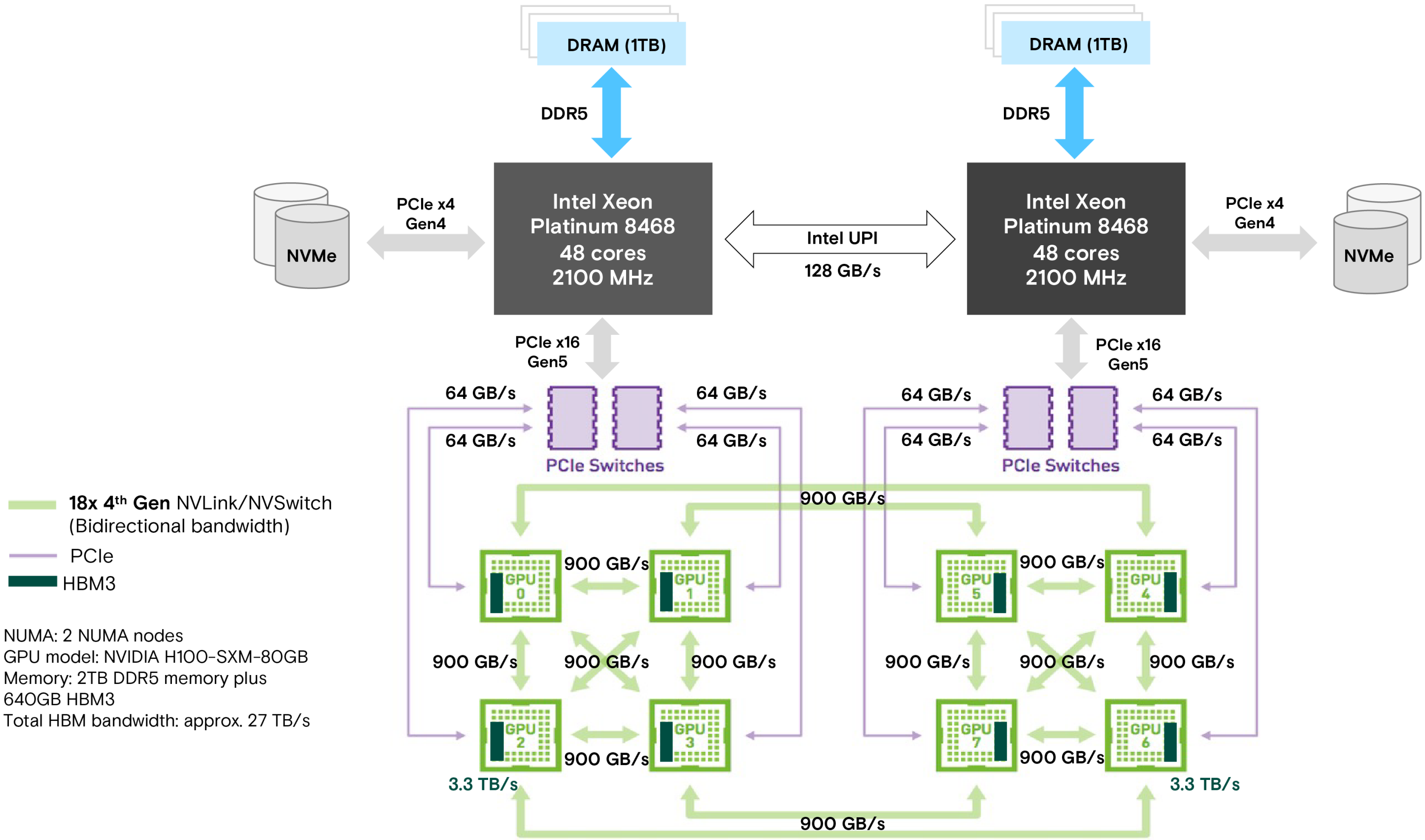
To analyze the inference performance of the select LLM model and its impact on HBM, testing and validation were carried out on a single NVIDIA HGX H100 box (manufactured by Supermicro). Tables 1 and 2 detail the GPU and host configuration of the system under test (SUT), while Figure 3 illustrates the 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 (approximation)
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	2x Intel® Xeon® Platinum 8468
Core(s) per socket	48
Socket(s)	2
CPU max frequency	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



NUMA: 2 NUMA nodes
GPU model: NVIDIA H100-SXM-80GB
Memory: 2TB DDR5 memory plus 640GB HBM3
Total HBM bandwidth: approx. 27 TB/s

Figure 3: System architecture of NVIDIA HGX H100

Methodology

We conducted several inference experiments on the GPU using the Llama 2 70B model developed by Meta [4]. For our experiment, we applied the ZeRO-Inference strategy (from the DeepSpeed library) to analyze the performance when the model weights are quantized (FP16 to INT4). In addition to quantizing the model weights, we optionally offloaded the model weights (if the model failed) to the CPU DRAM memory subsystem to free up space for larger batch sizes and longer context length. Throughout these experiments, the KV cache remained on the GPU HBM and the input prompt was fixed at 512 tokens, meaning that the total context length was the sum of the input prompt and the tokens generated for output.

Moreover, we measured the impact of HBM clock speed on inference throughput and carried out a practical analysis of the number of concurrent clients supported on a single GPU for a different number of generated output tokens.

As mentioned above, we used the Llama 2 70B model for our experiments. Llama 2 [4] is an updated version of the first Llama [19] release from Meta. It's a collection of models that ranges from 7 billion to 70 billion parameters that are pretrained on a new mix of publicly available data. The dataset size is 40% larger than the dataset used on the first version of the model. Among the key architectural changes of this family of models are an increase in the context length and adoption of grouped-query attention [3]. Figure 2 depicts the distinct attention methods, including the grouped-query attention method employed by Llama 2 models to address memory bandwidth challenges during the decoding stage.

Detailed results

CPU offload analysis

This section presents the performance results for inference when running the **Llama 2 70B** model in configurations that varied GPU count, model precision type (INT4 or FP16) and CPU offload (enabled or not enabled).

- Throughput as a function of the number of output tokens
- Throughput as a function of the batch size with KV cache stored on the GPU HBM

Throughput as a function of number of output tokens

Figure 4 shows the total inference throughput for the Llama 2 70B model across different configurations of output tokens, batch size, model precision and number of GPUs. The results are separated on the figure into different panels (or columns), each of which represents a different number of GPUs and batch size. We can correlate the batch size to the number of concurrent clients or requests that the server can support at one given time. Figure 4 also shows the performance of each scenario when the model weights are or are not offloaded to the CPU DRAM memory subsystem, thus reducing the required GPU resources. Recall that offloading is slowed down by the PCIe interconnect, which results in high inference latency [2]. However, offloading enables GPUs with lower HBM capacity to run inference beyond their limits.

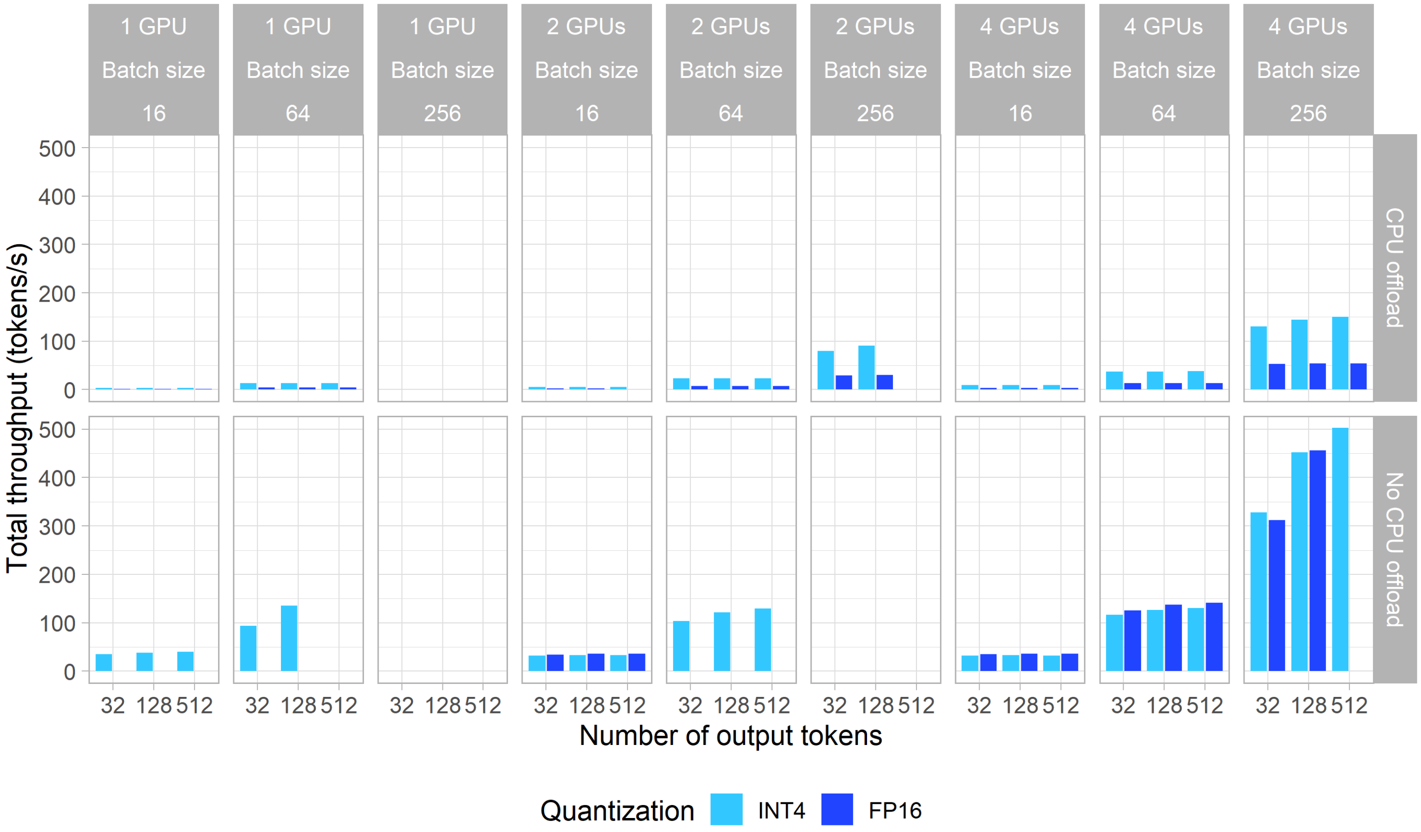


Figure 4: Inference throughput of Llama 2 70B as a function of the number of output tokens

Here are brief explanations of our findings:

- **Batch size 16**: When using a batch size of 16, a single GPU can run the model entirely on GPU HBM without the need for offloading. However, quantizing the model to INT4 is necessary because it fails when running at FP16 precision. When the model is deployed across two or four GPUs, it supports both quantization methods (INT4 and FP16), and the throughput remains comparable to that of a single GPU. Moreover, increasing the number of output tokens does not yield higher throughput. When using CPU offload, a single GPU can run the model in FP16. The drawback is that the throughput is low across all executions.
- **Batch size 64**: For a batch size of 64, we observe a nearly linear increase in throughput when the model executes entirely on the GPU HBM. With two GPUs, we can no longer run the model in FP16 as the model size and KV cache exceed the GPU HBM capacity. However, using INT4, we can output more tokens than we can with a single GPU. With CPU offload, we have a small increase in throughput and an improvement of INT4 over FP16 in all scenarios.
- **Batch size 256**: Due to HBM capacity limitations, only a reduced number of GPUs (fewer than four) can execute the model when it's quantized to INT4 as the batch size increases. If we attempt to run a single GPU inference with an increased number of output tokens and batch size, even with a model quantized to INT4, the process fails. Without offloading, using up to two GPUs for a batch size of 256 results in failure for both INT4 and FP16.

To summarize, our results show that INT4 quantization consistently demonstrates higher performance compared to FP16 when the model weights are offloaded to the CPU. However, this performance advantage is not evident when the model runs entirely on the GPU HBM. This outcome is primarily because the H100 GPUs are designed to optimize the performance of FP16 data processing.

Secondly, HBM speed is fast enough to keep the GPU busy while transferring the data, even in higher precision. Even though CPU offload allows fewer GPUs to execute the model in FP16 precision using higher batch sizes, it fails when the KV cache requirement is big enough to exceed the HBM capacity of a single GPU in the system. Increasing the batch size also yields proportional performance improvements on throughput (almost linearly) more than increasing the number of output tokens (autoregressive nature).

Throughput as a function of the batch size, holding output tokens constant

Figure 5 presents the results of the total inference throughput for the Llama 2 70B model. In this experiment, we held the number of output tokens constant at 512 while varying the batch size, model precision, number of GPUs and use of offloading. Also, for this analysis the KV cache is stored on the GPU HBM.

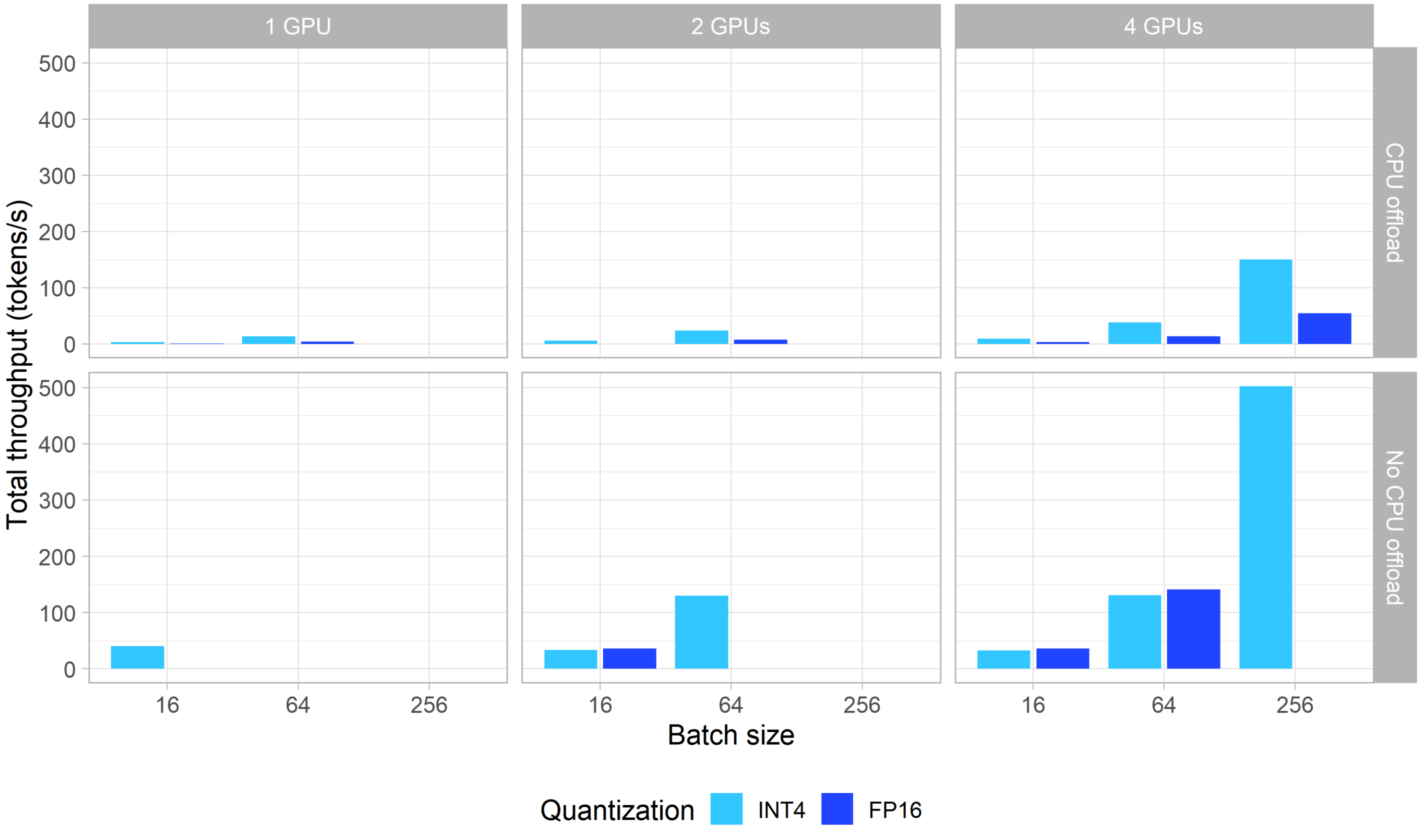


Figure 5: Inference throughput of Llama 2 70B as a function of the batch size

Consistent with Figure 4, Figure 5 shows that the batch size (shown on the *x*-axis) has a significant impact on throughput. When running with a lower number of GPUs (one GPU and two GPUs), even with model quantization, inference fails at larger batch sizes (in this case, 256) because the KV cache grows beyond the available HBM capacity.

A notable observation is that reducing resources by 50% (from four GPUs down to two GPUs), along with quantizing the model to INT4 precision, results in comparable performance at a batch size of 16, even without offloading the model weights to the CPU DRAM subsystem. Conversely, fewer GPUs fail to execute larger batch sizes. Also, when offloading is enabled, INT4 has three times greater performance than that of FP16, as shown in the panel using four GPUs.

Effects of varying HBM clock speed

For this experiment, we varied the HBM clock speed to evaluate its effect on inference throughput. As depicted in Figure 6, the throughput across four GPUs is presented as a function of both batch size and HBM clock speed. Each panel shows the result of using a different number of output tokens (denoted on the right) and type of model precision (listed at the top).

During this experiment, our intent was to solely observe the impact of varying HBM speed on inference performance. Thus, no offloading was used, and all model weights were loaded to the GPU HBM.

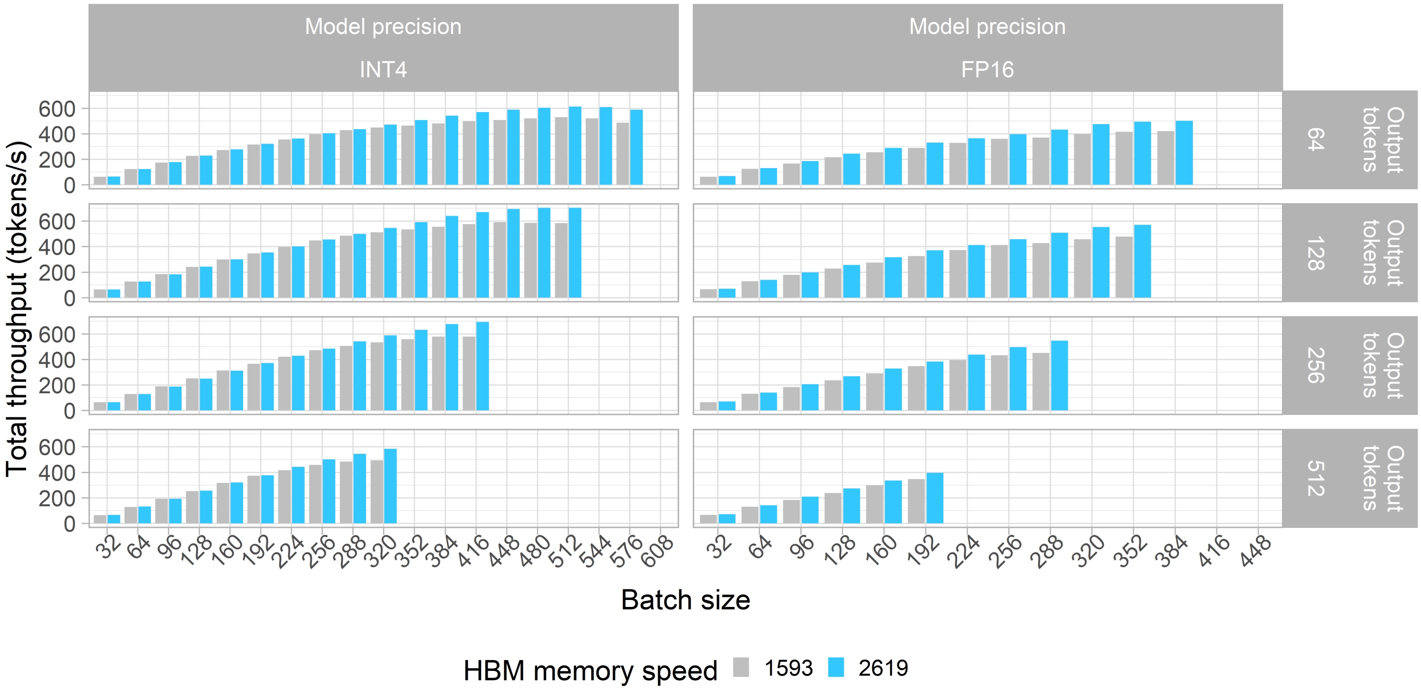


Figure 6: Inference throughput for Llama 2 70B on four GPUs as a function of the batch size

We can observe that, for both model precisions, INT4 and FP16, lower batch sizes do not highlight the difference between HBM clock speeds, as both clock speeds have similar performance. On the other hand, an increase in the batch size increases the observable performance difference between both clock speeds (1593 and 2619 MHz) up to **21%**. This occurs because the GPU performs more computation and more data movement at higher batch sizes, which necessitates more HBM bandwidth to keep the GPU busy. Thus, the difference between both clock speeds is more notable.

Larger batch sizes also emphasize the difference between INT4 and FP16. INT4 precision allows the system to execute larger batch sizes compared to FP16, therefore achieving higher performance. However, the HBM capacity constraints limit the combinations of batch size and output tokens that the system can run, further restricting our analysis.

Analysis of concurrent clients

We also analyzed the highest combination of batch size and context length that we could run on a single NVIDIA HGX H100 system using the Llama 2 70B model, quantized to INT4. To enable the model to run entirely on the HBM capacity of a single GPU and at larger batch sizes, we quantized the model down.

We correlated the batch size to the number of concurrent clients a server could handle, assuming each client issued a single request of *N* context length tokens, and the requests were batched. The context length itself is a sum of the user's input (prompt) and the number of output tokens. For this analysis, we varied the output tokens but kept the prompt constant at 512 tokens.

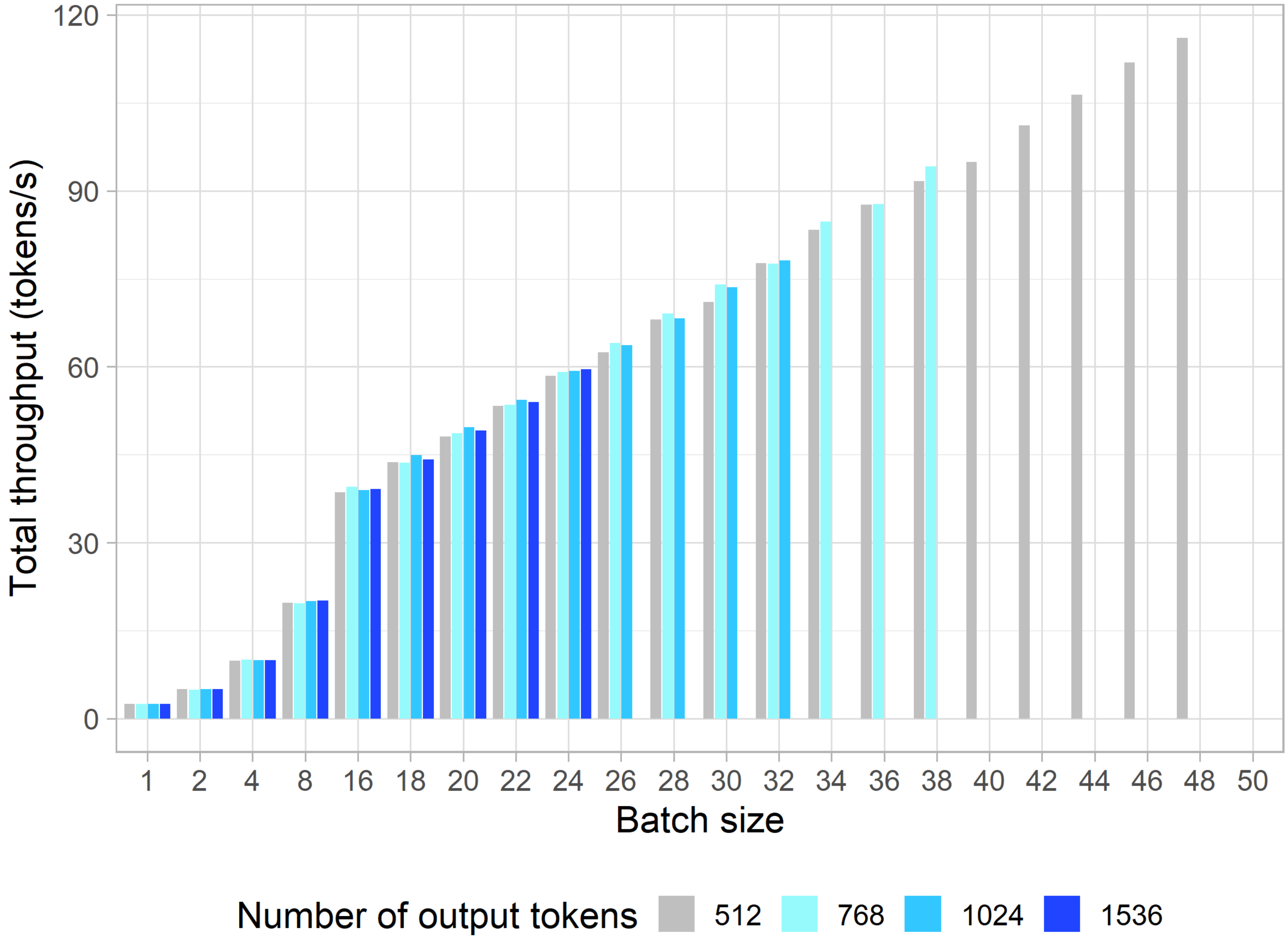


Figure 7: Total throughput for one H100 80GB GPU as a function of the batch size for Llama 2 70B quantized to INT4.

Figure 7 presents the total throughput achieved on a single GPU as a function of the batch size. The findings indicate that a single GPU can run 48 concurrent clients with a context length of 1024 tokens (comprising a prompt of 512 tokens and an equal number of output tokens). When the context length is doubled to 2048 tokens (a prompt of 512 tokens plus 1536 output tokens), the throughput increases only slightly. However, as a side effect, the number of concurrent clients (or batch size) is reduced by half (here, 24). Doubling the client count results in a nearly linear improvement (approximately two times) in throughput, while increasing the context length reduces the throughput proportionate to the reduction of clients.

Conclusion

LLMs have shown exceptional proficiency on a range of downstream tasks in natural language processing such as text classification and sentiment analysis, where progress has largely been driven by the adoption of the transformer architecture and innovations in GPU technology.

Yet, the computational demands of inference for these ever-expanding models are expensive, primarily due to the model's substantial capacity requirements and the memory bandwidth overhead incurred during parameter loading and KV cache operations in the decode phase. Addressing these challenges requires continued advancements in memory technologies to fully realize the enormous potential of LLMs. Micron's innovative memory portfolio of high-bandwidth and high-capacity products, especially HBM3E, emerges as a compelling solution to meet the demands of LLMs and AI workloads.

In addition to enhancements of the stacked architecture, the higher bandwidth, higher capacity, and improved power efficiency of HBM3E, as compared to previous generations, is key to meeting the demands of modern GPUs and the AI workloads they power. Along with HBM3E, Micron's memory products play an important role in elevating the performance of both GPUs and LLMs, helping to more fully harness the power of AI in our everyday lives.

References

[1] 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).

[2] Song, Yixin, Zeyu Mi, Haotong Xie, and Haibo Chen. "PowerInfer: Fast large language model serving with a consumer-grade GPU." arXiv preprint arXiv:2312.12456 (2023).

[3] Ainslie, Joshua, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. "GQA: Training generalized multi-query transformer models from multi-head checkpoints." arXiv preprint arXiv:2305.13245 (2023).

[4] 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).

[5] Pope, Reiner, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. "Efficiently scaling transformer inference." Proceedings of machine learning and systems 5 (2023).

[6] Aminabadi, Reza Yazdani, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase et al. "DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale." In SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-15. IEEE, 2022.

[7] Georgi Gerganov. ggerganov/llama.cpp: Port of Facebook's Llama model in C/C++. https://github.com/ggerganov/llama.cpp, 2023.

[8] Sheng, Ying, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. "FlexGen: High-throughput generative inference of large language models with a single GPU." In International Conference on Machine Learning, pp. 31094-31116. PMLR, 2023.

[9] Cheng, Zhoujun, Jungo Kasai, and Tao Yu. "Batch prompting: Efficient inference with large language model APIs." arXiv preprint arXiv:2301.08721 (2023).

[10] Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. "Efficient memory management for large language model serving with PagedAttention." In Proceedings of the 29th Symposium on Operating Systems Principles, pp. 611-626. 2023.

[11] Dao, Tri, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. "FlashAttention: Fast and memory-efficient exact attention with IO-awareness." Advances in neural information processing systems 35 (2022): 16344-16359.

[12] Wang, Ziheng, Jeremy Wohlwend, and Tao Lei. "Structured pruning of large language models." arXiv preprint arXiv:1910.04732 (2019).

[13] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

[14] Frantar, Elias, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. "GPTQ: Accurate post-training compression for generative pretrained transformers." arXiv preprint arXiv:2210.17323 (2022).

[15] Dettmers, Tim, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. "LLM.int8(): 8-bit matrix multiplication for transformers at scale." arXiv preprint arXiv:2208.07339 (2022).

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

[17] 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.

[18] Rasley, Jeff, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. "DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters." In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 3505-3506. 2020.

[19] Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière et al. "Llama: Open and efficient foundation language models." arXiv preprint arXiv:2302.13971 (2023).

[20] Shazeer, Noam. "Fast transformer decoding: One write-head is all you need." arXiv preprint arXiv:1911.02150 (2019).

micron.com/hbm