vmware®
by **Broadcom**

MENU

VMware Cloud Foundation (VCF) Blog

Home Page | Private AI | Technical | VMware Private AI

LLM Inference Sizing and Performance Guidance

Yuankun Fu
September 25, 2024

When planning to deploy a chatbot or simple Retrieval-Augmentation-Generation (RAG) pipeline on **VMware Private AI Foundation with NVIDIA** [1], you may have questions about sizing (capacity) and performance based on your existing GPU resources or potential future GPU acquisitions. For instance:

- What is the maximum number of concurrent requests that can be supported for a specific Large Language Model (LLM) on a specific GPU?

- What is the maximum sequence length (or prompt size) that a user can send to the chat app without experiencing a noticeably slow response time?

- What is the estimated response time (latency) for generating output tokens, and how does it vary with different input sizes and LLM sizes?

Conversely, if you have specific capacity or latency requirements for utilizing LLMs with X billion parameters, you may wonder:

- What type and quantity of GPUs should you acquire to meet these requirements?

This blog post aims to help answer these questions and guide your inference deployment planning. Please note that the calculations presented below are simplified estimates of inference time, as they do not account for additional factors that can impact performance, such as GPU communication costs, network delays, and software stack overhead. In practice, these factors can increase inference time by up to 50% or more. Therefore, the numbers calculated below should be considered best-case estimates of the actual inference time.

For the remainder of this blog, we will use the following terminology:

- "Prompt" refers to the input sequence sent to the LLM.

- "Response" denotes the output sequence generated by the LLM.

Understand Your GPU's Specifications

When selecting a GPU for your deployment, we distill the following key parameters to consider: FP16 (we decide to not use the sparsity value to estimate), GPU memory size, and GPU memory bandwidth. Below, we provide a list of recent GPUs with their relevant specifications in Table 1. Since most LLMs utilize half-precision floating-point arithmetic (FP16), we focus on the FP16 Tensor Core capabilities of each GPU.

GPU	FP16 (TFLOPS)	GPU Memory (GB)	Memory Bandwidth (GB/s)
A10	125	24	600

GPU	FP16 (TFLOPS)	GPU Memory (GB)	Memory Bandwidth (GB/s)
A30	330	24	933
L40	181	48	864
L40s	362	48	864
A100 40 GB	312	40	1555
A100 40 GB SXM	312	40	1555
A100 80 GB PCIe	312	80	1935
A100 80 GB SXM	312	80	2039
H100 PCIe	756.5	80	2000
H100 SXM	989.5	80	3350
H100 NVL	835.5	188	3900

Table 1. GPU_Specification

Important Consideration for Models Trained on Non-FP16 Data Formats

Some models are trained using alternative data formats like **BF16** (Bfloat16), **FP8**, or **INT8**. While most modern GPUs, such as the L40, A100, and H100 series, offer the same TFLOPS performance for both FP16 and BF16, there are exceptions. For instance, the A30 GPU has different TFLOPS values for FP16 and BF16.

If you plan to use a model trained on one of these specific data formats, it's crucial to replace the FP16 column in Table 1 with the corresponding TFLOPS value of your GPU, e.g., BF16, FP8, or INT8. Additionally, when working with quantized models using FP8 or INT8, be sure to use 1 Byte for memory-related calculations in the formulas below to ensure accurate results.

Understand Your LLM's Specifications

Table 2 presents specifications for recent large language models (LLMs), with the first six columns provided by the model producers. The last column, KV cache size (GiB/token) for each model, is calculated based on the values in the preceding columns. Table 2 includes six models, with the Mixtral-8x7B-v0.1 model [5] being the only Sparse Mixture of Experts (SMoE) architecture. This model has a total of 46.7 billion parameters, but during inference, its internal router utilizes only 2 experts, resulting in an active parameter count of 12.7 billion.

Model	Params (Billion)	dimension of the model (d_model)	number of attention heads (n_heads)	number of times the attention block shows up (n_layers)	Max Context window (N)	kv_cache_size (GiB/token)
Llama-3 8B	8	4096	32	32	8192	0.00049
Llama-3 70B	70	8192	64	80	8192	0.00244
Llama-3.1 8B	8	4096	32	32	131072	0.00049
Llama-3.1-70B	70	8192	64	80	131072	0.00244
Mistral-7B v0.3	7	4096	32	32	32768	0.00049
Mixtral-8x7B-v0.1	47 (total) 13 (active)	4096	32	32	32768	0.00098

Table 2. LLM_Specification

Calculating KV Cache Size per token for each model

At the core of every LLM lies the transformer engine, which consists of two distinct phases: prefill and autoregressive sampling.

- In the **prefill** phase, the model processes the input prompt tokens in parallel, populating the key-value (KV) cache. The KV cache serves as the model's state, embedded within the attention operation. During this phase, no tokens are generated.

- In the **autoregressive sampling** phase, the model leverages the current state stored in the KV cache to sample and decode the next token. By reusing the KV cache, we avoid the computational overhead of recalculating the cache for every new token. This approach enables faster sampling, as we don't need to pass all previously seen tokens through the model.

```
1  kv_cache_size_per_token
2      = (2 × 2 × n_layers × d_model) bytes/token
3      = (4 × 32 × 4096) bytes/token
4      = 524288 bytes/token
5      ~ 0.00049 GiB/token for Llama-3-8b
```

For a detailed explanation of the formula and its derivation, please refer to [2] and [3].

Note: As for the Mixtral-8x7b-v0.1 model, due to its sparse nature, its internal router uses 2 out of 8 expert models for inference, thus we need to additionally multiply by *num_experts* resulting in `0.00049 GiB/token × 2 experts = 0.00098 bytes/token`.

Calculate the Memory Footprint for a Specific Model

Now using the kv_cache_size_per_token value, you can also estimate the memory footprint required to support n_concurrent_request and average_context_window sizes. The formula to estimate the memory footprint is:

```
1  GPU_memory_foot_print
2      = model_weights_size + kv_cache_size
3      = num_model_param × size_fp16 + kv_cache_size_per_token × avg_context_window × n_concurrent_request
4      = 8B prams × 2 Byte + 0.00049 GiB/token × 1024 tokens × 10 requests
5      = 21 GB for llama-3-8B with average 1024 tokens in context window and 10 concurrent requests
```

Note: For the Mixtral-8x7b-v0.1 model, we need to use the model total weights (47B) in the above formula.

With the estimated memory footprint, you can use this information to determine the type and number of GPUs required to meet your needs for **VMware Private AI with NVIDIA**.

Calculate Estimated Capacity

All state-of-the-art LLM models are memory-bound [2, 4], meaning that their performance is limited by the memory access bandwidth rather than the computational capabilities of the GPU, so batching multiple prompts as the KV cache is a common way to improve throughput.

Based on the above two steps, we can estimate the theoretical maximum number of tokens that can be processed by one or more GPUs. This is calculated using the following formula:

```
1  kv_cache_tokens
2      = Remained_GPU_Mem ÷ kv_cache_size
3      = (Total_GPU_Mem - Model's weights) ÷ kv_cache_size_per_token
4      = (48 GB -  8 B prams × 2 Bytes) ÷ 0.00049 GB/token
5      = 65536 tokens for Llama-3-8b
```

Model	Params (Billion)	1× 48G L40/L40s	1× 80G A100/H100	2× 48G L40/L40s	2× 80G A100/H100	4× 48G L40/L40s	4× 80G A100/H100	8× 48G L40/L40s	8× 80G A100/H100
Llama-3-8B	8	65536	131072	163840	294912	360448	622592	753664	1277952
Llama-3-70B	70	OOM	OOM	OOM	8192	21299	73728	99942	204800
Llama-3.1-8B	8	65536	131072	163840	294912	360448	622592	753664	1277952
Llama-3.1-70B	70	OOM	OOM	OOM	8192	21299	73728	99942	204800
Mistral-7B-v0.3	7	69632	135168	167936	299008	364544	626688	757760	1282048
Mixtral-8x7B-v0.1	47	OOM	OOM	OOM	67584	100352	231424	296960	559104

Table 3. Max number of tokens allowed in the KV cache for a model with different numbers and types of GPUs

In **Table 3**, "OOM" stands for Out of Memory, i.e., a particular GPU does not have sufficient resources to handle a specific model. To overcome this limitation, we assume that a single model is distributed across multiple GPUs using parallelism, e.g., tensor parallelism. This allows us to handle larger models by splitting the model's weights across multiple GPUs, reducing the memory requirements for each GPU. For example, when holding host llama-3-70b models, at least `70 B prams × 2 Bytes = 140 GB` for model weights is required, thus we need at minimum 7× A10/A30-24GB GPUs, or 4× L40/L40s-48GB GPUs, or 4× A100-40GB GPUs, or 2× A100/H100-80GB GPUs to hold it. Thus, each GPU contains `70 ÷ 7 = 10B`, `70 ÷ 4 = 17.5B`, or `70 ÷ 2 = 35B` parameters.

After you get the maximum number of tokens allowed in the KV cache, assuming the longest acceptable prompt length of a model, we can estimate the maximum number of concurrent requests that can be handled by each LLM. This represents the worst-case scenario, as it's unlikely that all concurrent users will utilize the maximum context length. However, this calculation provides a useful upper bound, helping you understand the limitations of your underlying system.

```
1    concurrent_requests_with_the_longest_prompt_size
2        = Max_tokens_in_KV_cache ÷ maximum_context_window
3        = 65536 tokens ÷ 8192 tokens per request
4        = 8 concurrent prompts for Llama-3-8b
```

Model	Max Context Window	1× 48G L40/L40s	1× 80G A100/H100	2× 48G L40/L40s	2× 80G A100/H100	4× 48G L40/L40s	4× 80G A100/H100	8× 48G L40/L40s	8× 80G A100/H100
Llama-3-8B	8192	8	16	20	36	44	76	92	156
Llama-3-70B	8192	OOM	OOM	OOM	1	3	9	12	25
Llama-3.1-8B	131072	1	1	1	2	3	5	6	10
Llama-3.1-70B	131072	OOM	OOM	OOM	0	0	1	1	2
Mistral-7B-v0.3	32768	2	4	5	9	11	19	23	39
Mixtral-8x7B-v0.1	32768	OOM	OOM	OOM	2	3	7	9	17

Table 4. Max concurrent requests when using the largest available context window for a prompt

In real-world use cases, average prompt sizes are often shorter, resulting in higher concurrent throughput. For instance, if the average context window is 4096 tokens, you can expect a significant increase in concurrent requests compared to using the maximum acceptable prompt length.

Model	Context Window	1× 48G L40/L40s	1× 80G A100/H100	2× 48G L40/L40s	2× 80G A100/H100	4× 48G L40/L40s	4× 80G A100/H100	8× 48G L40/L40s	8× 80G A100/H100
Llama-3-8B	4096	16	32	40	72	88	152	184	312
Llama-3-70B	4096	OOM	OOM	OOM	2	5	18	24	50
Llama-3.1-8B	4096	16	32	40	72	88	152	184	312
Llama-3.1-70B	4096	OOM	OOM	OOM	2	5	18	24	50
Mistral-7B-v0.3	4096	17	33	41	73	89	153	185	313
Mixtral-8x7B-v0.1	4096	OOM	OOM	OOM	17	25	57	73	137

Table 5. Max concurrent requests when using an average context window of 4096 tokens

Calculate Load for Multiple Model Replicas

When calculating the load for multiple replicas of a model, you can use the first available number that indicates the model can be held in a GPU setup.

For instance, consider the case where you want to deploy two replicas of the Llama-3-8B model on 2x L40 GPUs, with load balancing between them. In this case, each L40 GPU contains a whole-weight copy of the Llama-3-8B model, rather than distributing the weights among GPUs.

To calculate the total number of concurrent requests, you can multiply the number of replicas by the number of concurrent prompts supported by each replica. For example, assuming a prompt size of 4k, the calculation would be:

```
2 replicas × 16 concurrent prompts per replica = 32 concurrent requests
```

Calculate Estimated Latency

The total time to solution, which is the time it takes to receive a response, consists of two main components:

- **Prefill Time**: The time it takes to process the input prompt, which includes the time to populate the key-value (KV) cache.

- **Token Generation Time**: The time it takes to generate each new token in the response.

Note: The calculations presented below provide simplified estimates of inference time, which should be considered as lower bounds (or best estimates), as they do not account for various additional factors that can impact performance, such as GPU communication costs, network delays, and software stack overhead. In practice, these factors can increase inference time by up to 50% or more, depending on the specific system configuration and workload.

Prefill_time_per_token on each GPU

The prefill section assumes that we batch all of the prompt tokens into a single forward pass. For simplicity, we assume the limiting bottleneck is compute, and not memory. Thus, we can use the following formula to estimate the prefill time (in milliseconds) required to process a single token in a prompt.

```
1 Prefill_time_per_token
2     = model weights per GPU ÷ accelerator compute capabilty
3     = (8B prams × 2) FLOP ÷ 181 TFLOP/s
4     = 0.088 ms for llama-3-8b on L40
```

When using parallelism to deploy a model (e.g., tensor parallelism), weights are evenly distributed among multi-GPUs, thus we should use the portion of parameters in each GPU to estimate. In Table 6, for 70B and 8x7B models, we show the minimum number of GPUs required to hold them. Specifically, for 8x7B models, we use the "active" 13 billion parameters used by two experts for estimation.

Model Size	7B	8B	70B and Minimum GPUs Required		8x7B and Minimum GPUs Required	
A10	0.112	0.128	0.160	7	0.208	5
A30	0.042	0.048	0.061	7	0.079	5
L40	0.077	0.088	0.193	4	0.144	3
L40s	0.039	0.044	0.097	4	0.072	3
A100 40 GB	0.045	0.051	0.112	4	0.083	3
A100 40 GB SXM	0.045	0.051	0.112	4	0.083	3
A100 80 GB PCIe	0.045	0.051	0.224	2	0.083	2
A100 80 GB SXM	0.045	0.051	0.224	2	0.083	2
H100 PCIe	0.019	0.021	0.093	2	0.034	2
H100 SXM	0.014	0.016	0.071	2	0.026	2
H100 NVL	0.017	0.019	0.084	2	0.031	2

Table 6. Prefill time (ms) to compute a single token of different model sizes in the prompt for GPUs

Generation_time_per_token on each GPU

The autoregressive part of generation is memory-bound [3]. We can estimate the time (in milliseconds) required to generate a single token in the response by the following formula. This metric is also known as Time Per Output Token (TPOT).

```
1 Generation_time_per_token
2     = number of bytes moved (the model weights) per GPU ÷ accelerator memory bandwidth
3     = (8B prams × 2) Bytes ÷ 864 GB/s
4     = 18.5 ms for llama-3-8b on L40
```

Similarly in Table 7, we use the portion of parameters allocated to each GPU to estimate sizing and performance for 70B and 8x7B models.

Model sizes	7B	8B	70B and Minimum GPUs Required		8x7B and Minimum GPUs Required	
A10	23.3	26.7	33.3	7	43.3	5
A30	15.0	17.1	21.4	7	27.9	5
L40	16.2	18.5	40.5	4	30.1	3
L40s	16.2	18.5	40.5	4	30.1	3
A100 40 GB	9.0	10.3	22.5	4	16.7	3
A100 40 GB SXM	9.0	10.3	22.5	4	16.7	3
A100 80 GB PCIe	7.2	8.3	36.2	2	13.4	2
A100 80 GB SXM	6.9	7.8	34.3	2	12.8	2
H100 PCIe	7.0	8.0	35.0	2	13.0	2
00 SXM	4.2	4.8	20.9	2	7.8	2

| H100 NVL | 3.6 | 4.1 | 17.9 | 2 | 6.7 | 2 |

Table 7. Generation_time_per_token (ms) of different model sizes in the response for GPUs

The Generation_time_per_token is useful to estimate whether it meets your requirements of time-to-first-token (TTFT) and Token-per-second (TPS) or inter-token-latency (ITL).

To deliver a seamless user experience in chat-type applications, **it's essential to achieve a time-to-first-token (TTFT) below the average human visual reaction time of 200 milliseconds**. This metric is also impacted by other factors, such as network speed, prompt length, and model size. Consider these factors that will impact your real values on your infrastructure.

A fast TTFT is only half the equation; a high tokens-per-second (TPS) is equally important for real-time applications like chat. Typically, **a TPS of 30 or higher is recommended.** This corresponds to an ITL of around 33.3 milliseconds when using streaming output. To put this into perspective, consider the average reading speed of humans. The average reading speed is estimated to be between 200-300 words per minute, with exceptional readers reaching up to 1,000 words per minute. In comparison, a model generating 30 tokens per second (approximately 90 words per second) can produce around 1,350 words per minute. This is significantly faster than even the fastest human readers, making it more than capable of keeping pace with their reading speed.

Total latency of a request

We can calculate the estimated total time for a single prompt and its response by the following formula.

```
1  Estimated response time
2      = prefill_time + generation time
3      = prompt_size × prefill_time_per_token + response_size × generation_time_per_token
4      = 4000 × 0.088 ms + 256 × 18.5 ms
5      = 5.1 s for llama-3-8b on L40
```

Model sizes	7B	8B	70B and Minimum GPUs Required		8x7B and Minimum GPUs Required	
A10	6.4	7.3	9.2	7	11.9	5
A30	4.0	4.6	5.7	7	7.4	5
L40	4.5	5.1	11.1	4	8.3	3
L40s	4.3	4.9	10.8	4	8.0	3
A100 40 GB	2.5	2.8	6.2	4	4.6	3
A100 40 GB SXM	2.5	2.8	6.2	4	4.6	3
A100 80 GB PCIe	2.0	2.3	10.2	2	3.8	2
A100 80 GB SXM	1.9	2.2	9.7	2	3.6	2
H100 PCIe	1.9	2.1	9.3	2	3.5	2
H100 SXM	1.1	1.3	5.6	2	2.1	2
H100 NVL	1.0	1.1	4.9	2	1.8	2

Table 8. Estimated Latency (s) of prompt_size = 4000 and response_size = 256

Use the Estimation Calculator Script

Now that you have a solid understanding of the key factors that impact LLM inference performance, you can leverage the provided [Python script](#) to estimate the memory footprint, capacity, and latency on [VMware Private Foundation AI with NVIDIA](#). To get started, this script allows users to customize its behavior by providing input values for the following parameters to reflect your desired configuration:

- **num_gpu (-g):** Specify the number of GPUs you plan to use for your deployment.

- **prompt_sz (-p):** Define the average size of the input prompts you expect to process.

- **response_sz (-r):** Set the average size of the responses you expect to generate.

- **n_concurrent_req (-c):** Indicate the number of concurrent requests you anticipate handling.

By modifying these input values or inserting your target models in the source code, you can easily estimate the performance characteristics of your LLM deployment and make informed decisions about your infrastructure requirements.

```
 1   > python LLM_size_per_calculator.py -g 4 -p 4096 -l 256 -c 10
 2   num_gpu = 4, prompt_size = 4096 tokens, response_size = 256 tokens
 3   n_concurrent_request = 10
 4
 5   ******************** Estimate LLM Memory Footprint ********************
 6   | Model          | KV Cache Size per Token  | Memory Footprint  |
 7   |---------------+--------------------------+-------------------|
 8   | Llama-3.1-70B | 0.002441 GiB/token       | 246.25 GB         |
 9
10   ******************** Estimate LLM Capacity and Latency ********************
11   | Model         | GPU       |   Max # KV Cache Tokens | Prefill Time | TPOT (ms) | TTFT    | E2E Latency | Output Tokens Thro
12   |---------------+-----------+-------------------------+--------------+-----------+---------+-------------+-------------------
13   | Llama-3.1-70B | H100 PCIe |                   73728 | 0.046 ms     | 17.500 ms | 0.064 s | 4.7 s       | 54.82 tokens/sec
14   | Llama-3.1-70B | H100 SXM  |                   73728 | 0.035 ms     | 10.448 ms | 0.046 s | 2.8 s       | 90.80 tokens/sec
15   | Llama-3.1-70B | H100 NVL  |                   96665 | 0.042 ms     | 8.974 ms  | 0.051 s | 2.5 s       | 103.68 tokens/sec
16   | Llama-3.1-70B | H200 SXM  |                  173670 | 0.035 ms     | 7.292 ms  | 0.043 s | 2.0 s       | 127.27 tokens/sec
17   | Llama-3.1-70B | H200 NVL  |                  173670 | 0.042 ms     | 7.292 ms  | 0.049 s | 2.0 s       | 125.60 tokens/sec
18
```

Conclusion

In this blog post, we have provided a comprehensive guide to help you plan and deploy Large Language Models (LLMs) on VMware Private AI Foundation with NVIDIA. We have discussed the key factors that impact LLM inference performance, including GPU specifications and model specifications. We have also provided a set of formulas, tables, and a Python script to help you estimate the memory footprint, capacity, and latency of your LLM deployment based on your requirements. We hope that this blog post helps to guide you deliver a seamless user experience in chat-type applications on VMware Private AI Foundation with NVIDIA.

Reference

[1] VMware Private AI Foundation with NVIDIA

[2] A guide to LLM inference and performance

[3] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

[4] Efficient Memory Management for Large Language Model Serving with PagedAttention

[5] Mixtral 8x7B

Acknowledgments

Yuankun Fu

Yuankun Fu is a seasoned performance engineer at VMware by Broadcom, focusing on optimizing AI and HPC performance, with a Ph.D. from Purdue University and 14+ years of HPC experience.

Related Articles

Home Page

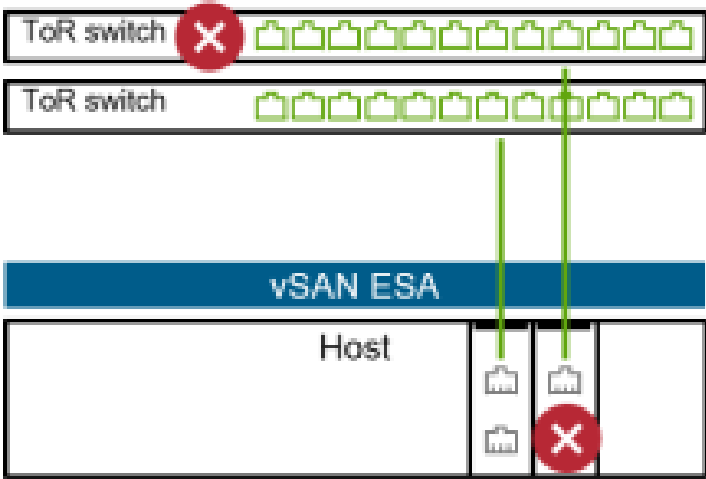## Rotating Credentials in VCF using PowerCLI SDK

Angel Petrov
June 9, 2025

Home Page

Introducing the Hidden Power of the PowerCLI SDK

Angel Petrov
June 3, 2025

vSAN Networking –
Teaming for Redundancy

VCF Storage (vSAN)

vSAN Networking - Teaming for Redundancy

Pete Koehler
June 3, 2025

vmware®
by **Broadcom**

Resources

Blogs

Careers

Communities

Customer Stories

News and Stories

Topics

Trust Center

Support

Broadcom Support

Documentation

Hands-On Labs

Licensing

Twitter

YouTube

Facebook

LinkedIn

Contact Sales

Accessibility

Privacy

Supplier Responsibility

Terms Of Use