Large Language Model Inference Performance

xueguofeng2011@gmail.com

Feb 6, 2024

Goals

- To analyze the nature of the large language model (LLM) inference.
- To assess the inference performance of LLM: various approaches such as batch processing, coroutine, multithreading, and multiprocessing will be employed.
- Three models—Yi-34B-Chat-4bits, Yi-6B-Chat, and GPT2—will be used.
- To examine performance disparities between Windows and WSL2.
- VRAM usage will be observed under different circumstances.
- The primary focus is on Consumer GPUs and low-level implementation.

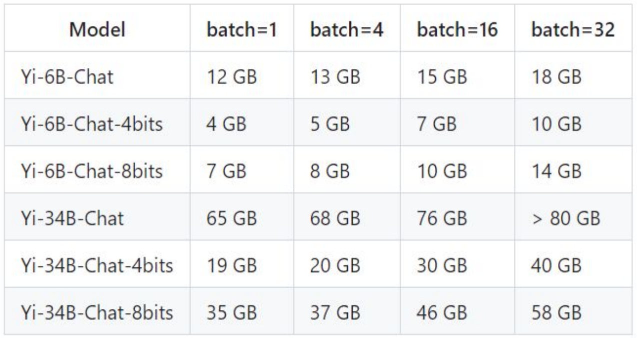
Analysis of LLM Inference

```
def generate_next_token(model, input_ids):
     # CPU invokes the kernel functions in GPU and transfers data with GPU - blocking
    input_ids = input_ids[:, -200:]
    outputs = model(input_ids=input_ids)  # Forward Propagation
    logits = outputs.logits
    next_token_logits = logits[0, -1, :]
    next_token_logits = next_token_logits / g_temperature
    next_token_logits[g_unk_id] = -float('Inf')
    iltered_logits = top_k_top_p_filtering(next_token_logits, top_k=g_top_k, top_p=g_top_p)
    next_token_id = torch.multinomial(F.softmax(filtered_logits, dim=-1), num_samples=1)
```

```
def predict_one_sample(model, tokenizer, device, title, context):
    # CPU operations
    title_ids = tokenizer.encode(title, add_special_tokens=False)
    context_ids = tokenizer.encode(context, add_special_tokens=False)
    input_ids = title_ids + [g_sep_id] + context_ids
    cur_len = len(input_ids)
    last_token_id = input_ids[-1]
    # CPU invokes the kernel functions in GPU and transfers data with GPU - blocking
    input_ids = torch.tensor([input_ids], dtype=torch.long, device=device)

    while True:  # many rounds
        # CPU invokes the kernel functions and transfers data with GPU - blocking
        next_token_id = generate_next_token(input_ids)
        # CPU invokes the kernel functions and transfers data with GPU - blocking
        input_ids = torch.cat((input_ids, next_token_id.unsqueeze(0)), dim=1)
        # CPU operations
        cur_len += 1
        word = tokenizer.convert_ids_to_tokens( next_token_id.item() )
         if cur_len >= g_generate_max_len and last_token_id == 8 and next_token_id == 3:
           break
         if cur_len >= g_generate_max_len and word in [".", "。", "！", "!", "?", "？", ",", "，"]:
           break
         if next_token_id == g_eod_id:
           break

    # CPU operations
    result = tokenizer.decode(input_ids.squeeze(0))
    content = result.split("<sep>")[1]
    return content
```

- For the Transformer-based models, the inference process involves intensive operations on both the CPU and GPU.
- The CPU handles tasks such as input processing and managing the overall execution of the model (e.g., Tokenizer, Top P/K, EOS, and Max Length).
- The GPU is particularly for parallelizable matrix operations in the neural network's forward pass.
- Autoregressive - the previously generated outputs are fed back into the model as input for generating subsequent outputs.
- VRAM usage

  1)The model size
  2)Input & KV Cache, 20% ~ 50% of the model
  3)Others (the CUDA context and libraries)

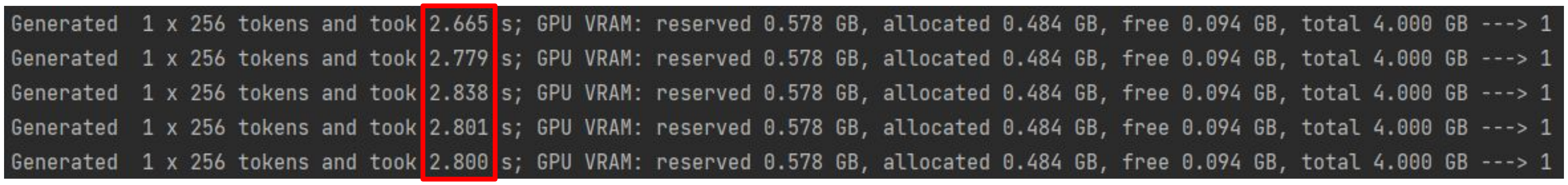


Model	batch=1	batch=4	batch=16	batch=32
Yi-6B-Chat	12 GB	13 GB	15 GB	18 GB
Yi-6B-Chat-4bits	4 GB	5 GB	7 GB	10 GB
Yi-6B-Chat-8bits	7 GB	8 GB	10 GB	14 GB
Yi-34B-Chat	65 GB	68 GB	76 GB	> 80 GB
Yi-34B-Chat-4bits	19 GB	20 GB	30 GB	40 GB
Yi-34B-Chat-8bits	35 GB	37 GB	46 GB	58 GB

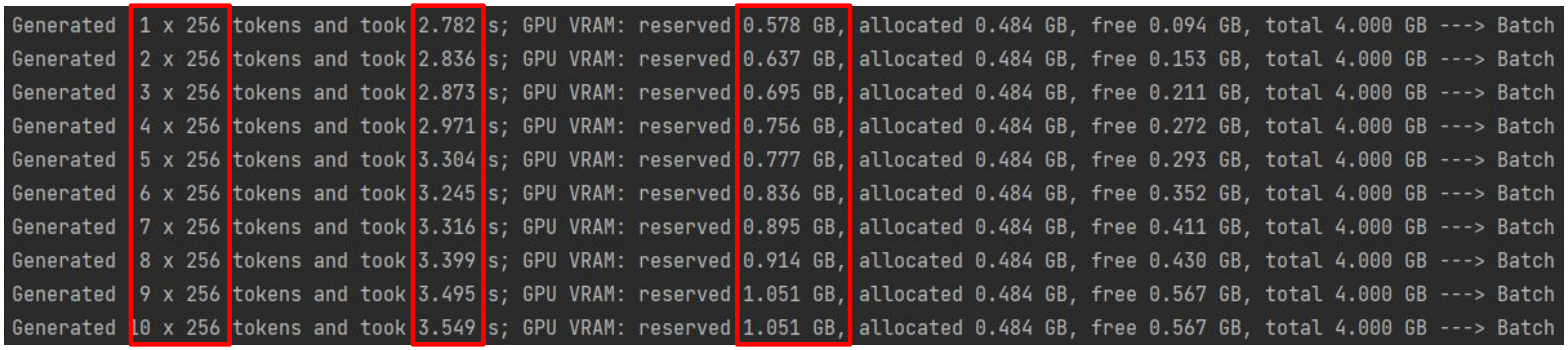
https://github.com/01-ai/Yi/blob/main/docs/deployment.md#hardware-requirements
https://www.jinghong-chen.net/estimate-vram-usage-in-llm-inference/

Batch Inference: Windows, GPT2, RTX A2000

Model	GPT2, around 500 MB
Environment	Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The performance remains consistently stable, taking approximately 2.8 seconds to generate 256 tokens. As the batch size increases from 1 to 10, the inference time only sees a modest increase of around 25% , generating from 92 tokens per second to 730 tokens per second; however, the VRAM usage nearly doubles. Batch inference can significantly enhance performance by effectively leveraging the GPU cache and parallel processing capabilities. The optimal batch size can vary depending on the characteristics of the model, dataset, and hardware, necessitating experimentation and ongoing performance monitoring.



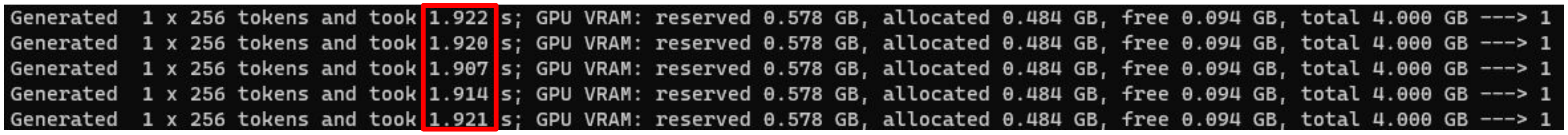
```
Generated  1 x 256 tokens and took 2.665 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.779 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.838 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.801 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.800 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
```



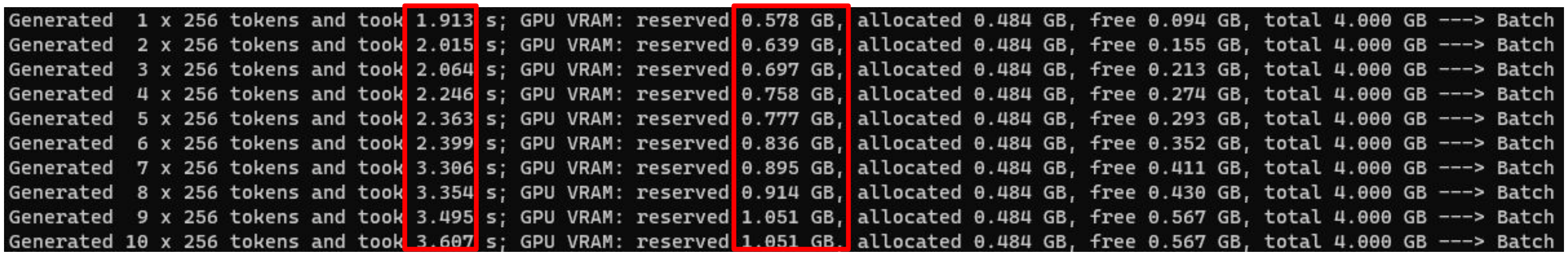
```
Generated  1 x 256 tokens and took 2.782 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Batch
Generated  2 x 256 tokens and took 2.836 s; GPU VRAM: reserved 0.637 GB, allocated 0.484 GB, free 0.153 GB, total 4.000 GB ---> Batch
Generated  3 x 256 tokens and took 2.873 s; GPU VRAM: reserved 0.695 GB, allocated 0.484 GB, free 0.211 GB, total 4.000 GB ---> Batch
Generated  4 x 256 tokens and took 2.971 s; GPU VRAM: reserved 0.756 GB, allocated 0.484 GB, free 0.272 GB, total 4.000 GB ---> Batch
Generated  5 x 256 tokens and took 3.304 s; GPU VRAM: reserved 0.777 GB, allocated 0.484 GB, free 0.293 GB, total 4.000 GB ---> Batch
Generated  6 x 256 tokens and took 3.245 s; GPU VRAM: reserved 0.836 GB, allocated 0.484 GB, free 0.352 GB, total 4.000 GB ---> Batch
Generated  7 x 256 tokens and took 3.316 s; GPU VRAM: reserved 0.895 GB, allocated 0.484 GB, free 0.411 GB, total 4.000 GB ---> Batch
Generated  8 x 256 tokens and took 3.399 s; GPU VRAM: reserved 0.914 GB, allocated 0.484 GB, free 0.430 GB, total 4.000 GB ---> Batch
Generated  9 x 256 tokens and took 3.495 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
Generated 10 x 256 tokens and took 3.549 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
```

Batch Inference: WSL2, GPT2, RTX A2000

Model	GPT2, around 500 MB
Environment	WSL2 of Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The WSL2 performance surpasses that of Windows when the batch size is set to 1, with a time of 1.9 seconds compared to 2.8 seconds. However, the performance becomes similar between WSL2 and Windows when the batch size is increased to 10. The VRAM usage appears almost identical when observed within the processes running on both WSL and Windows, but there is a great disparity when checking through the Window Task Manager or using 'nvidia-smi'.



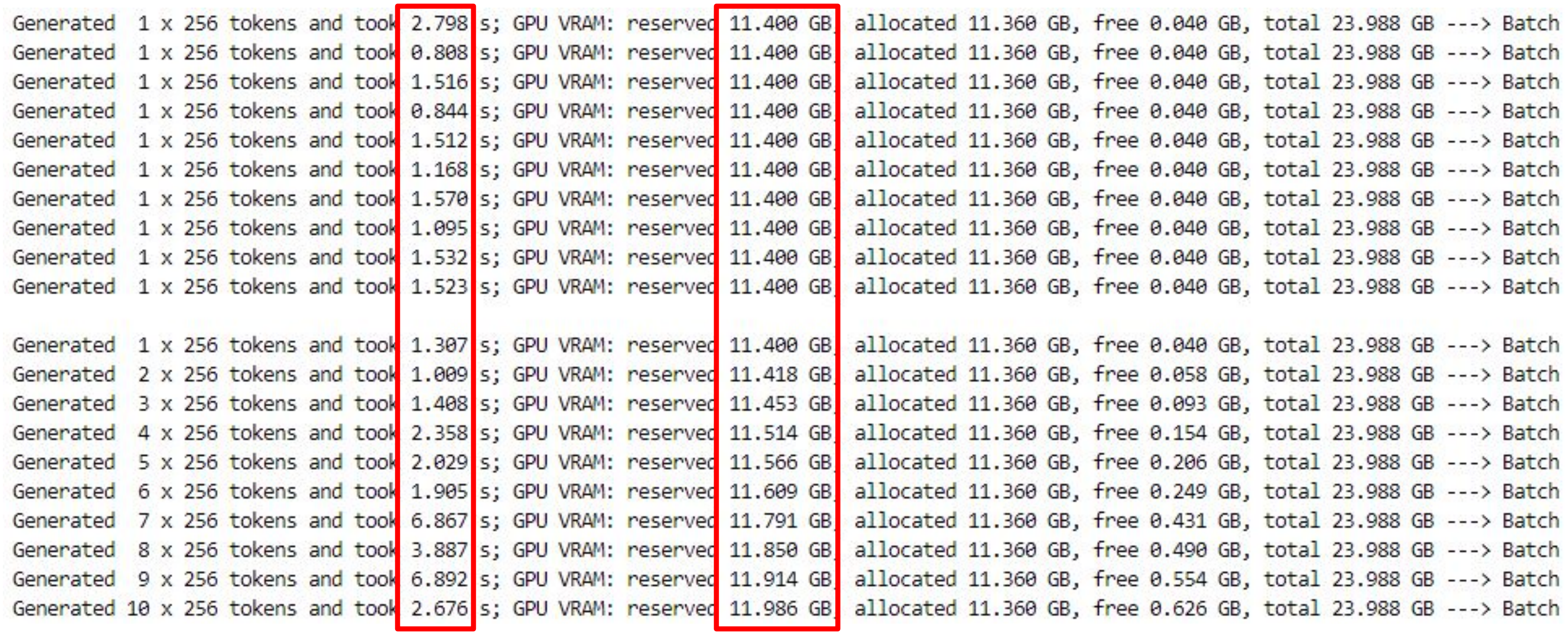
```
Generated  1 x 256 tokens and took 1.922 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 1.920 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 1.907 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 1.914 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 1.921 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
```



```
Generated  1 x 256 tokens and took 1.913 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Batch
Generated  2 x 256 tokens and took 2.015 s; GPU VRAM: reserved 0.639 GB, allocated 0.484 GB, free 0.155 GB, total 4.000 GB ---> Batch
Generated  3 x 256 tokens and took 2.064 s; GPU VRAM: reserved 0.697 GB, allocated 0.484 GB, free 0.213 GB, total 4.000 GB ---> Batch
Generated  4 x 256 tokens and took 2.246 s; GPU VRAM: reserved 0.758 GB, allocated 0.484 GB, free 0.274 GB, total 4.000 GB ---> Batch
Generated  5 x 256 tokens and took 2.363 s; GPU VRAM: reserved 0.777 GB, allocated 0.484 GB, free 0.293 GB, total 4.000 GB ---> Batch
Generated  6 x 256 tokens and took 2.399 s; GPU VRAM: reserved 0.836 GB, allocated 0.484 GB, free 0.352 GB, total 4.000 GB ---> Batch
Generated  7 x 256 tokens and took 3.306 s; GPU VRAM: reserved 0.895 GB, allocated 0.484 GB, free 0.411 GB, total 4.000 GB ---> Batch
Generated  8 x 256 tokens and took 3.354 s; GPU VRAM: reserved 0.914 GB, allocated 0.484 GB, free 0.430 GB, total 4.000 GB ---> Batch
Generated  9 x 256 tokens and took 3.495 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
Generated 10 x 256 tokens and took 3.607 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
```

Batch Inference: WSL2, Yi-6B-Chat, RTX 4090

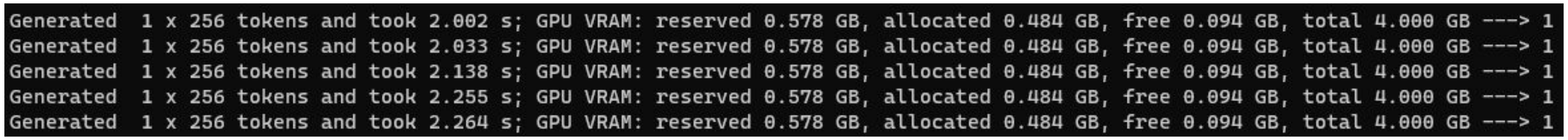
Model	Yi-6B-Chat, around **10 GB**
Environment (Cloud, shared)	WSL2 of Windows GPU: RTX 4090, 24GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The performance exhibits inconsistency, ranging from 0.8 seconds to 2.7 seconds for the generations of 256 tokens. With an increase in batch size, the inference time shows a gradual but unstable growth. Notably, the VRAM usage rises by approximately 0.5 GB when the batch size is set to 10. It's worth noting that the GPU was idle during the test, suggesting that the inconsistent performance may stem from a busy CPU.



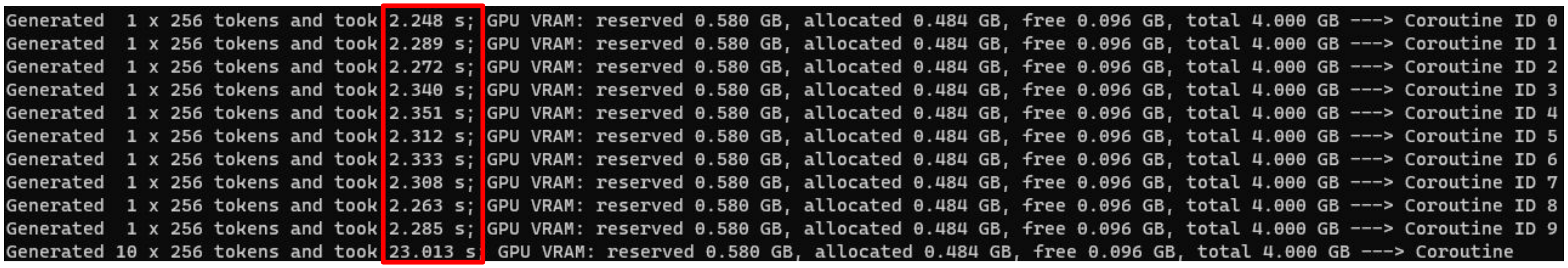
```
Generated  1 x 256 tokens and took 2.798 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 0.808 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.516 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 0.844 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.512 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.168 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.570 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.095 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.532 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  1 x 256 tokens and took 1.523 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch

Generated  1 x 256 tokens and took 1.307 s; GPU VRAM: reserved 11.400 GB, allocated 11.360 GB, free 0.040 GB, total 23.988 GB ---> Batch
Generated  2 x 256 tokens and took 1.009 s; GPU VRAM: reserved 11.418 GB, allocated 11.360 GB, free 0.058 GB, total 23.988 GB ---> Batch
Generated  3 x 256 tokens and took 1.408 s; GPU VRAM: reserved 11.453 GB, allocated 11.360 GB, free 0.093 GB, total 23.988 GB ---> Batch
Generated  4 x 256 tokens and took 2.358 s; GPU VRAM: reserved 11.514 GB, allocated 11.360 GB, free 0.154 GB, total 23.988 GB ---> Batch
Generated  5 x 256 tokens and took 2.029 s; GPU VRAM: reserved 11.566 GB, allocated 11.360 GB, free 0.206 GB, total 23.988 GB ---> Batch
Generated  6 x 256 tokens and took 1.905 s; GPU VRAM: reserved 11.609 GB, allocated 11.360 GB, free 0.249 GB, total 23.988 GB ---> Batch
Generated  7 x 256 tokens and took 6.867 s; GPU VRAM: reserved 11.791 GB, allocated 11.360 GB, free 0.431 GB, total 23.988 GB ---> Batch
Generated  8 x 256 tokens and took 3.887 s; GPU VRAM: reserved 11.850 GB, allocated 11.360 GB, free 0.490 GB, total 23.988 GB ---> Batch
Generated  9 x 256 tokens and took 6.892 s; GPU VRAM: reserved 11.914 GB, allocated 11.360 GB, free 0.554 GB, total 23.988 GB ---> Batch
Generated 10 x 256 tokens and took 2.676 s; GPU VRAM: reserved 11.986 GB, allocated 11.360 GB, free 0.626 GB, total 23.988 GB ---> Batch
```

Coroutine Inference: WSL2, GPT2, RTX A2000

Model	GPT2, around 500 MB
Environment	WSL2 of Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	Regular IO operations are inherently blocking. To leverage concurrency effectively, Python asyncio-compatible libraries are required. The inference functions by Hugging Face and PyTorch are synchronous and not asyncio-compatible, indicating that calling threads are blocked and wait for the GPU result after invoking ‘model.generate’ or ‘model()’. When defining 10 coroutine tasks using ‘async def’, it's important to note that these tasks will run serially, one after the other. Consequently, the inference time increases linearly as the number of coroutine task numbers grows.



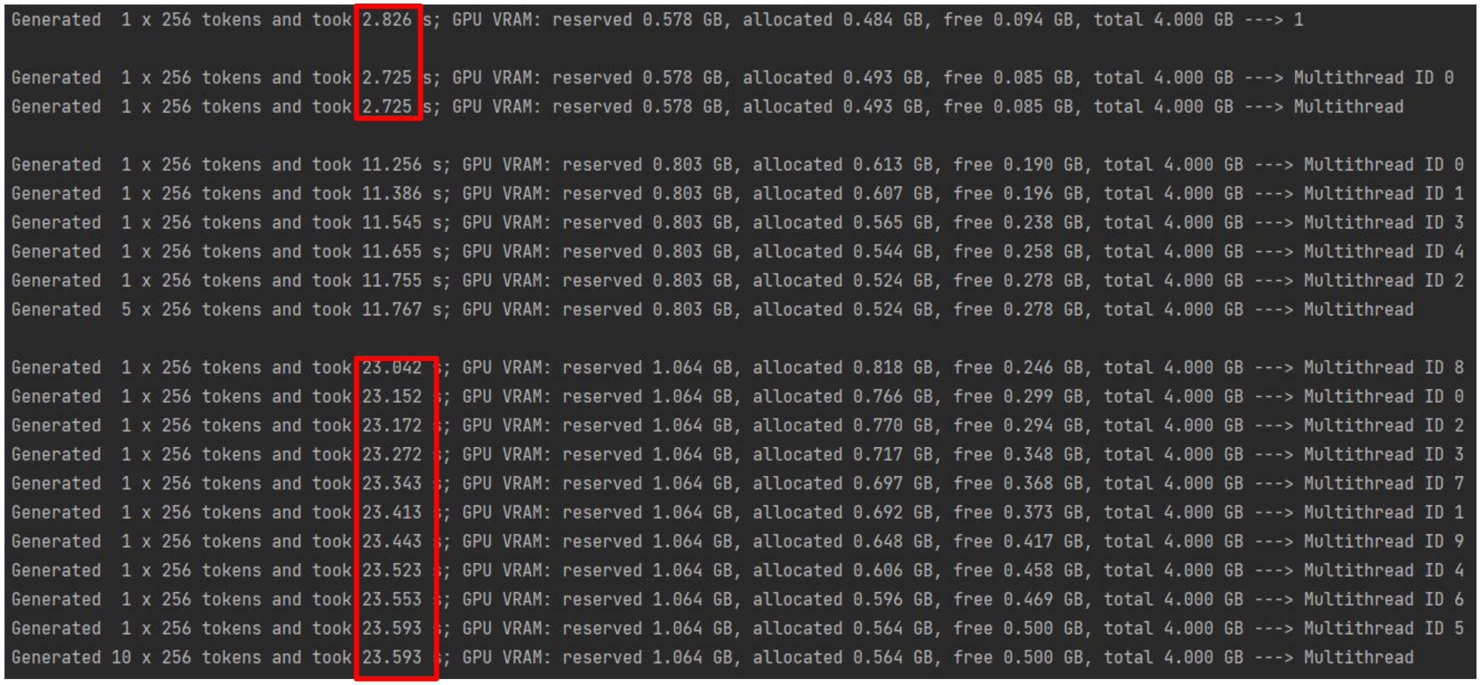
```
Generated  1 x 256 tokens and took 2.002 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.033 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.138 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.255 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.264 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
```



```
Generated  1 x 256 tokens and took 2.248 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 0
Generated  1 x 256 tokens and took 2.289 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 1
Generated  1 x 256 tokens and took 2.272 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 2
Generated  1 x 256 tokens and took 2.340 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 3
Generated  1 x 256 tokens and took 2.351 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 4
Generated  1 x 256 tokens and took 2.312 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 5
Generated  1 x 256 tokens and took 2.333 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 6
Generated  1 x 256 tokens and took 2.308 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 7
Generated  1 x 256 tokens and took 2.263 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 8
Generated  1 x 256 tokens and took 2.285 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine ID 9
Generated 10 x 256 tokens and took 23.013 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Coroutine
```

Multithreading Inference: Windows, GPT2, RTX A2000

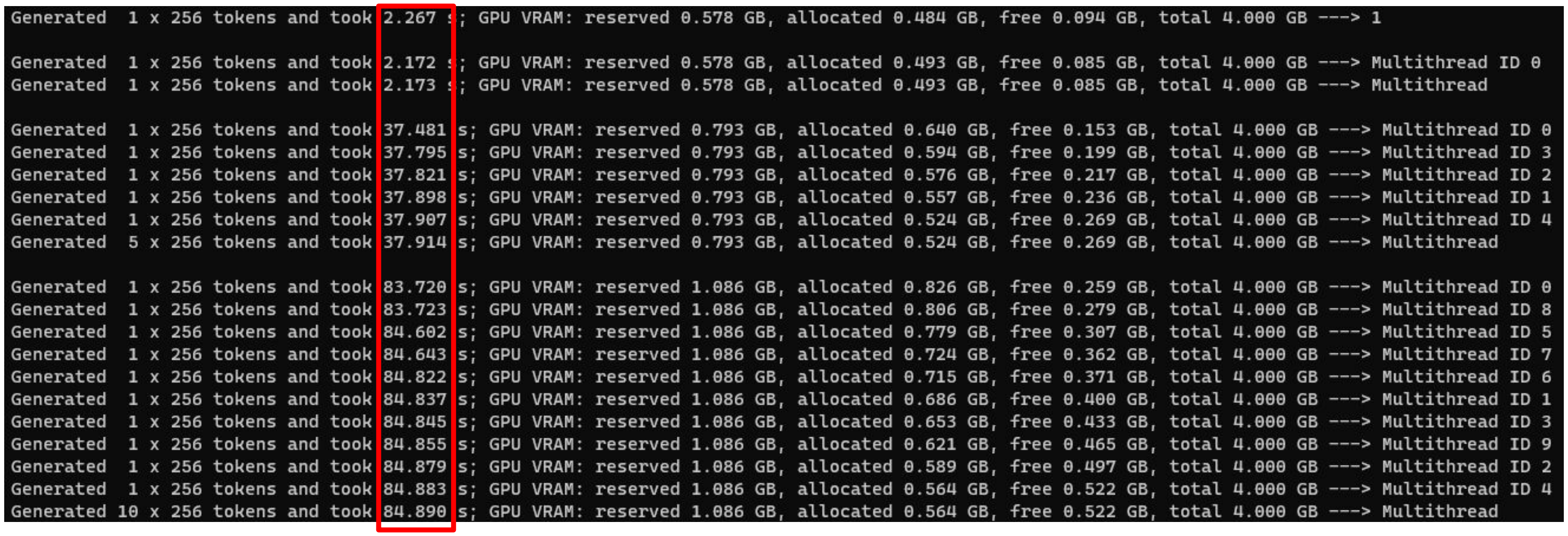
Model	GPT2, around 500 MB
Environment	Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The Global Interpreter Lock (GIL) in Python allows only one thread to execute Python byte code at a time within a process. While the GIL can impact the performance of multithreaded applications, certain operations are not affected by the GIL, including I/O operations and certain C extensions such as Numpy. The inference process involves operations on both the CPU and GPU. When thread A is interrupted or waits for the GPU result, such as cudaMemcpy and cudaMalloc/cudaFree (all synchronous operations), thread B can be running. However, threads A and B run inferences at different layers/stages on the GPU, which limits the optimal utilization of the GPU cache. From the test results, it's evident that the throughput of 10 Python threads is still better than 10 individual inferences. Two cases were tested with no significant difference: 10 threads sharing the default CUDA stream, and each thread having its own CUDA stream. Thus, the primary performance bottlenecks are mainly attributed to the Python GIL and GPU (no batch).



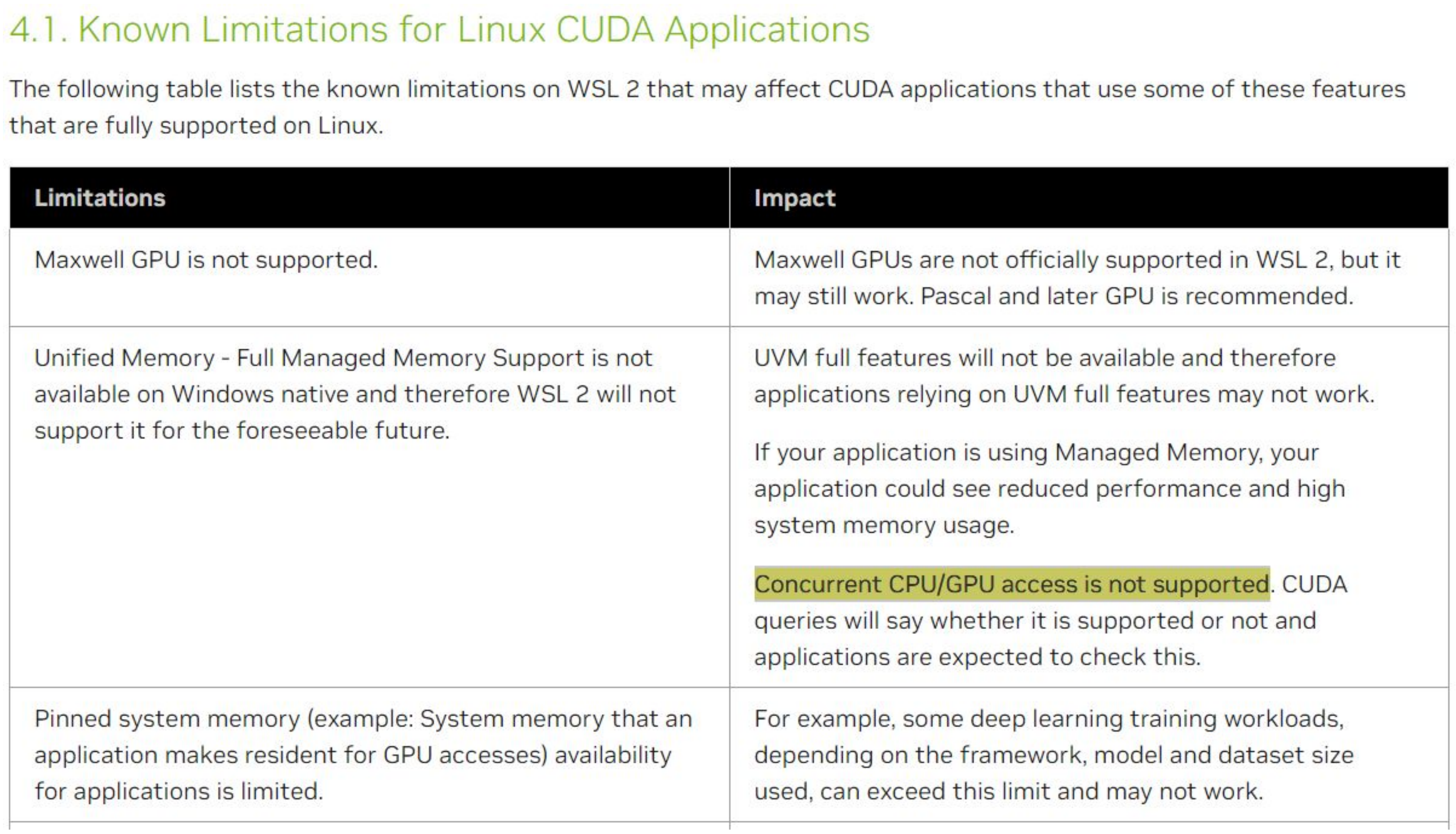
```
Generated  1 x 256 tokens and took 2.826 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1

Generated  1 x 256 tokens and took 2.725 s; GPU VRAM: reserved 0.578 GB, allocated 0.493 GB, free 0.085 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 2.725 s; GPU VRAM: reserved 0.578 GB, allocated 0.493 GB, free 0.085 GB, total 4.000 GB ---> Multithread

Generated  1 x 256 tokens and took 11.256 s; GPU VRAM: reserved 0.803 GB, allocated 0.613 GB, free 0.190 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 11.386 s; GPU VRAM: reserved 0.803 GB, allocated 0.607 GB, free 0.196 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 11.545 s; GPU VRAM: reserved 0.803 GB, allocated 0.565 GB, free 0.238 GB, total 4.000 GB ---> Multithread ID 3
Generated  1 x 256 tokens and took 11.655 s; GPU VRAM: reserved 0.803 GB, allocated 0.544 GB, free 0.258 GB, total 4.000 GB ---> Multithread ID 4
Generated  1 x 256 tokens and took 11.755 s; GPU VRAM: reserved 0.803 GB, allocated 0.524 GB, free 0.278 GB, total 4.000 GB ---> Multithread ID 2
Generated  5 x 256 tokens and took 11.767 s; GPU VRAM: reserved 0.803 GB, allocated 0.524 GB, free 0.278 GB, total 4.000 GB ---> Multithread

Generated  1 x 256 tokens and took 23.042 s; GPU VRAM: reserved 1.064 GB, allocated 0.818 GB, free 0.246 GB, total 4.000 GB ---> Multithread ID 8
Generated  1 x 256 tokens and took 23.152 s; GPU VRAM: reserved 1.064 GB, allocated 0.766 GB, free 0.299 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 23.172 s; GPU VRAM: reserved 1.064 GB, allocated 0.770 GB, free 0.294 GB, total 4.000 GB ---> Multithread ID 2
Generated  1 x 256 tokens and took 23.272 s; GPU VRAM: reserved 1.064 GB, allocated 0.717 GB, free 0.348 GB, total 4.000 GB ---> Multithread ID 3
Generated  1 x 256 tokens and took 23.343 s; GPU VRAM: reserved 1.064 GB, allocated 0.697 GB, free 0.368 GB, total 4.000 GB ---> Multithread ID 7
Generated  1 x 256 tokens and took 23.413 s; GPU VRAM: reserved 1.064 GB, allocated 0.692 GB, free 0.373 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 23.443 s; GPU VRAM: reserved 1.064 GB, allocated 0.648 GB, free 0.417 GB, total 4.000 GB ---> Multithread ID 9
Generated  1 x 256 tokens and took 23.523 s; GPU VRAM: reserved 1.064 GB, allocated 0.606 GB, free 0.458 GB, total 4.000 GB ---> Multithread ID 4
Generated  1 x 256 tokens and took 23.553 s; GPU VRAM: reserved 1.064 GB, allocated 0.596 GB, free 0.469 GB, total 4.000 GB ---> Multithread ID 6
Generated  1 x 256 tokens and took 23.593 s; GPU VRAM: reserved 1.064 GB, allocated 0.564 GB, free 0.500 GB, total 4.000 GB ---> Multithread ID 5
Generated 10 x 256 tokens and took 23.593 s; GPU VRAM: reserved 1.064 GB, allocated 0.564 GB, free 0.500 GB, total 4.000 GB ---> Multithread
```

Multithreading Inference: WSL2, GPT2, RTX A2000

Model	GPT2, around 500 MB
Environment	WSL2 of Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The performance of multithreading inference on Windows degrades to ¼ speed on WSL2. The inference time (84.8 s) for 10 Python threads on WSL2 exceeds that of running 40 individual inferences (2.1 s). While on Windows, the inference time for 10 Python threads (23 s) is approximately 8 times that of one inference (2.7 s). Didn't find a comprehensive explanation about this phenomenon. There are some known limitations for CUDA application on WSL2 from NVIDIA and discussions about multithreading issues in WSL2 can be found on the internet.



```
Generated  1 x 256 tokens and took 2.267 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1

Generated  1 x 256 tokens and took 2.172 s; GPU VRAM: reserved 0.578 GB, allocated 0.493 GB, free 0.085 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 2.173 s; GPU VRAM: reserved 0.578 GB, allocated 0.493 GB, free 0.085 GB, total 4.000 GB ---> Multithread

Generated  1 x 256 tokens and took 37.481 s; GPU VRAM: reserved 0.793 GB, allocated 0.640 GB, free 0.153 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 37.795 s; GPU VRAM: reserved 0.793 GB, allocated 0.594 GB, free 0.199 GB, total 4.000 GB ---> Multithread ID 3
Generated  1 x 256 tokens and took 37.821 s; GPU VRAM: reserved 0.793 GB, allocated 0.576 GB, free 0.217 GB, total 4.000 GB ---> Multithread ID 2
Generated  1 x 256 tokens and took 37.898 s; GPU VRAM: reserved 0.793 GB, allocated 0.557 GB, free 0.236 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 37.907 s; GPU VRAM: reserved 0.793 GB, allocated 0.524 GB, free 0.269 GB, total 4.000 GB ---> Multithread ID 4
Generated  5 x 256 tokens and took 37.914 s; GPU VRAM: reserved 0.793 GB, allocated 0.524 GB, free 0.269 GB, total 4.000 GB ---> Multithread

Generated  1 x 256 tokens and took 83.720 s; GPU VRAM: reserved 1.086 GB, allocated 0.826 GB, free 0.259 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 83.723 s; GPU VRAM: reserved 1.086 GB, allocated 0.806 GB, free 0.279 GB, total 4.000 GB ---> Multithread ID 8
Generated  1 x 256 tokens and took 84.602 s; GPU VRAM: reserved 1.086 GB, allocated 0.779 GB, free 0.307 GB, total 4.000 GB ---> Multithread ID 5
Generated  1 x 256 tokens and took 84.643 s; GPU VRAM: reserved 1.086 GB, allocated 0.724 GB, free 0.362 GB, total 4.000 GB ---> Multithread ID 7
Generated  1 x 256 tokens and took 84.822 s; GPU VRAM: reserved 1.086 GB, allocated 0.715 GB, free 0.371 GB, total 4.000 GB ---> Multithread ID 6
Generated  1 x 256 tokens and took 84.837 s; GPU VRAM: reserved 1.086 GB, allocated 0.686 GB, free 0.400 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 84.845 s; GPU VRAM: reserved 1.086 GB, allocated 0.653 GB, free 0.433 GB, total 4.000 GB ---> Multithread ID 3
Generated  1 x 256 tokens and took 84.855 s; GPU VRAM: reserved 1.086 GB, allocated 0.621 GB, free 0.465 GB, total 4.000 GB ---> Multithread ID 9
Generated  1 x 256 tokens and took 84.879 s; GPU VRAM: reserved 1.086 GB, allocated 0.589 GB, free 0.497 GB, total 4.000 GB ---> Multithread ID 2
Generated  1 x 256 tokens and took 84.883 s; GPU VRAM: reserved 1.086 GB, allocated 0.564 GB, free 0.522 GB, total 4.000 GB ---> Multithread ID 4
Generated 10 x 256 tokens and took 84.890 s; GPU VRAM: reserved 1.086 GB, allocated 0.564 GB, free 0.522 GB, total 4.000 GB ---> Multithread
```



4.1. Known Limitations for Linux CUDA Applications

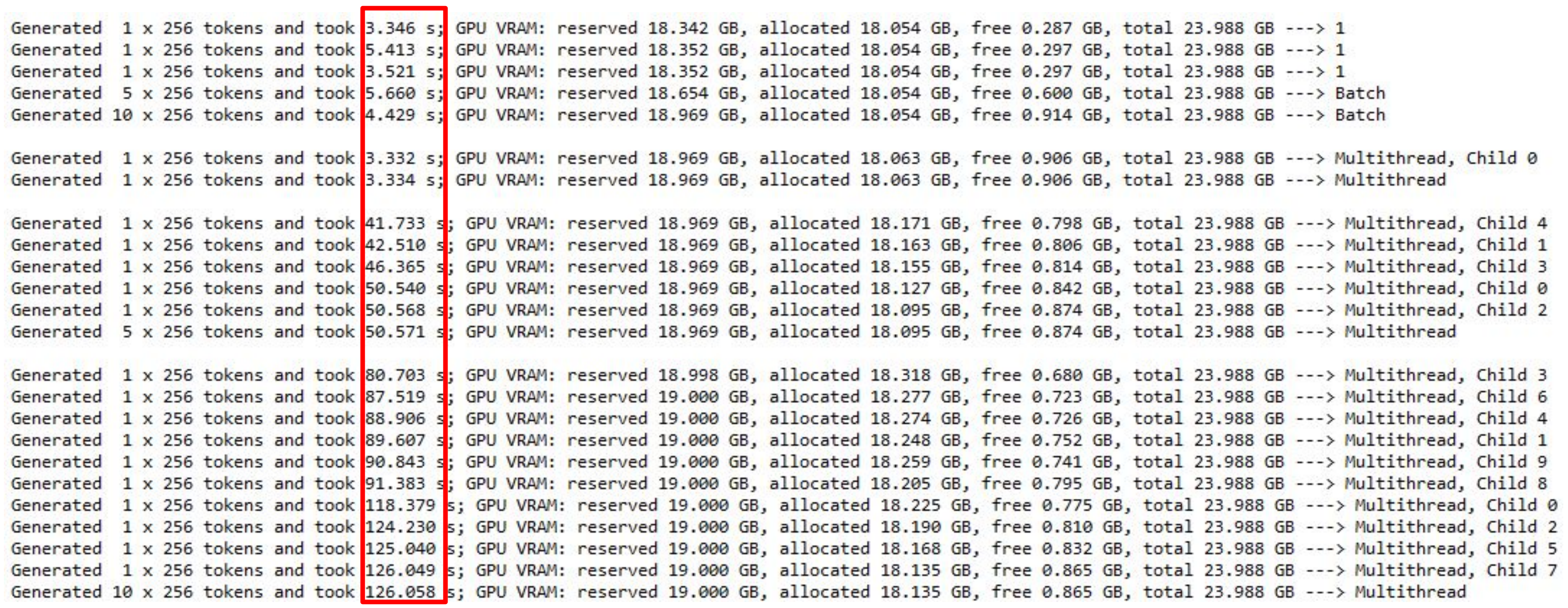
The following table lists the known limitations on WSL 2 that may affect CUDA applications that use some of these features that are fully supported on Linux.

Limitations	Impact
Maxwell GPU is not supported.	Maxwell GPUs are not officially supported in WSL 2, but it may still work. Pascal and later GPU is recommended.
Unified Memory - Full Managed Memory Support is not available on Windows native and therefore WSL 2 will not support it for the foreseeable future.	UVM full features will not be available and therefore applications relying on UVM full features may not work. If your application is using Managed Memory, your application could see reduced performance and high system memory usage. Concurrent CPU/GPU access is not supported. CUDA queries will say whether it is supported or not and applications are expected to check this.
Pinned system memory (example: System memory that an application makes resident for GPU accesses) availability for applications is limited.	For example, some deep learning training workloads, depending on the framework, model and dataset size used, can exceed this limit and may not work.

https://docs.nvidia.com/cuda/wsl-user-guide/index.html
https://github.com/dotnet/runtime/issues/42994
https://www.reddit.com/r/bashonubuntuonwindows/comments/mm567j/management_of_wsl2_cpu_affinity_self_promotion/

Multithreading Inference: WSL2, Yi-34B-Chat-4bits, RTX 4090

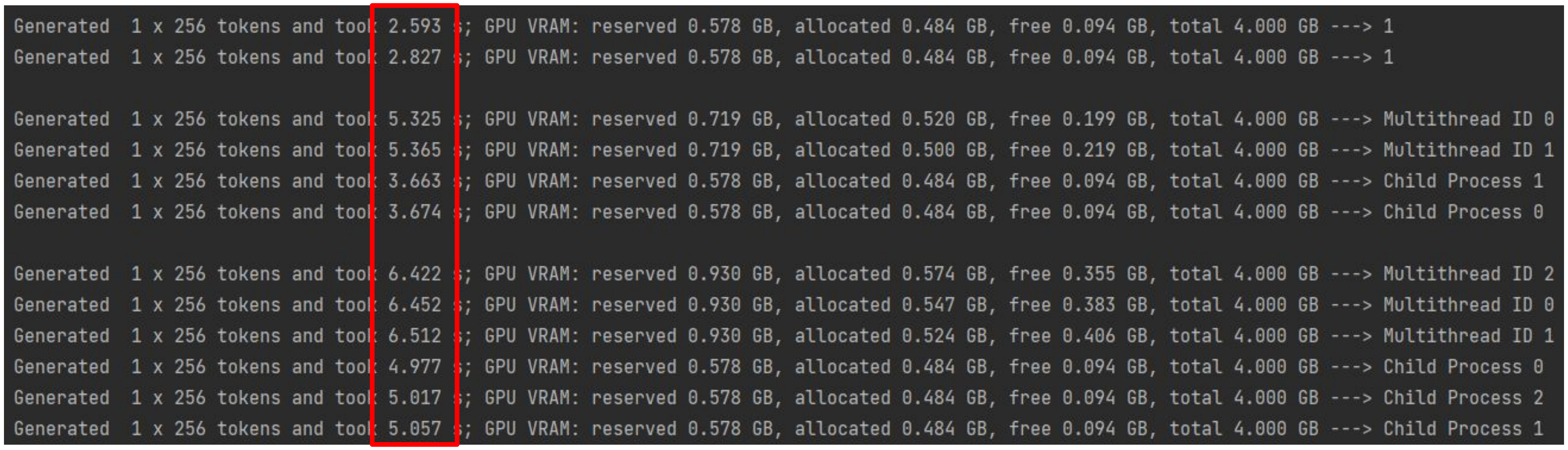
Model	Yi-34B-Chat-4bits, around **18 GB**
Environment (Cloud, shared)	WSL2 of Windows GPU: RTX 4090, 24GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	The similar phenomenon is observed: the inference time (120 s) for 10 Python threads is approximately 40 times that of one inference (3.3 s). The GPU utilization exceeds 70% when processing batch inputs (size 10) but decreases to approximately 25% during multithreading inference.



```
Generated  1 x 256 tokens and took 3.346 s; GPU VRAM: reserved 18.342 GB, allocated 18.054 GB, free 0.287 GB, total 23.988 GB ---> 1
Generated  1 x 256 tokens and took 5.413 s; GPU VRAM: reserved 18.352 GB, allocated 18.054 GB, free 0.297 GB, total 23.988 GB ---> 1
Generated  1 x 256 tokens and took 3.521 s; GPU VRAM: reserved 18.352 GB, allocated 18.054 GB, free 0.297 GB, total 23.988 GB ---> 1
Generated  5 x 256 tokens and took 5.660 s; GPU VRAM: reserved 18.654 GB, allocated 18.054 GB, free 0.600 GB, total 23.988 GB ---> Batch
Generated 10 x 256 tokens and took 4.429 s; GPU VRAM: reserved 18.969 GB, allocated 18.054 GB, free 0.914 GB, total 23.988 GB ---> Batch

Generated  1 x 256 tokens and took 3.332 s; GPU VRAM: reserved 18.969 GB, allocated 18.063 GB, free 0.906 GB, total 23.988 GB ---> Multithread, Child 0
Generated  1 x 256 tokens and took 3.334 s; GPU VRAM: reserved 18.969 GB, allocated 18.063 GB, free 0.906 GB, total 23.988 GB ---> Multithread

Generated  1 x 256 tokens and took 41.733 s; GPU VRAM: reserved 18.969 GB, allocated 18.171 GB, free 0.798 GB, total 23.988 GB ---> Multithread, Child 4
Generated  1 x 256 tokens and took 42.510 s; GPU VRAM: reserved 18.969 GB, allocated 18.163 GB, free 0.806 GB, total 23.988 GB ---> Multithread, Child 1
Generated  1 x 256 tokens and took 46.365 s; GPU VRAM: reserved 18.969 GB, allocated 18.155 GB, free 0.814 GB, total 23.988 GB ---> Multithread, Child 3
Generated  1 x 256 tokens and took 50.540 s; GPU VRAM: reserved 18.969 GB, allocated 18.127 GB, free 0.842 GB, total 23.988 GB ---> Multithread, Child 0
Generated  1 x 256 tokens and took 50.568 s; GPU VRAM: reserved 18.969 GB, allocated 18.095 GB, free 0.874 GB, total 23.988 GB ---> Multithread, Child 2
Generated  5 x 256 tokens and took 50.571 s; GPU VRAM: reserved 18.969 GB, allocated 18.095 GB, free 0.874 GB, total 23.988 GB ---> Multithread

Generated  1 x 256 tokens and took 80.703 s; GPU VRAM: reserved 18.998 GB, allocated 18.318 GB, free 0.680 GB, total 23.988 GB ---> Multithread, Child 3
Generated  1 x 256 tokens and took 87.519 s; GPU VRAM: reserved 19.000 GB, allocated 18.277 GB, free 0.723 GB, total 23.988 GB ---> Multithread, Child 6
Generated  1 x 256 tokens and took 88.906 s; GPU VRAM: reserved 19.000 GB, allocated 18.274 GB, free 0.726 GB, total 23.988 GB ---> Multithread, Child 4
Generated  1 x 256 tokens and took 89.607 s; GPU VRAM: reserved 19.000 GB, allocated 18.248 GB, free 0.752 GB, total 23.988 GB ---> Multithread, Child 1
Generated  1 x 256 tokens and took 90.843 s; GPU VRAM: reserved 19.000 GB, allocated 18.259 GB, free 0.741 GB, total 23.988 GB ---> Multithread, Child 9
Generated  1 x 256 tokens and took 91.383 s; GPU VRAM: reserved 19.000 GB, allocated 18.205 GB, free 0.795 GB, total 23.988 GB ---> Multithread, Child 8
Generated  1 x 256 tokens and took 118.379 s; GPU VRAM: reserved 19.000 GB, allocated 18.225 GB, free 0.775 GB, total 23.988 GB ---> Multithread, Child 0
Generated  1 x 256 tokens and took 124.230 s; GPU VRAM: reserved 19.000 GB, allocated 18.190 GB, free 0.810 GB, total 23.988 GB ---> Multithread, Child 2
Generated  1 x 256 tokens and took 125.040 s; GPU VRAM: reserved 19.000 GB, allocated 18.168 GB, free 0.832 GB, total 23.988 GB ---> Multithread, Child 5
Generated  1 x 256 tokens and took 126.049 s; GPU VRAM: reserved 19.000 GB, allocated 18.135 GB, free 0.865 GB, total 23.988 GB ---> Multithread, Child 7
Generated 10 x 256 tokens and took 126.058 s; GPU VRAM: reserved 19.000 GB, allocated 18.135 GB, free 0.865 GB, total 23.988 GB ---> Multithread
```

Multiprocessing Inference: Windows, GPT2, A2000

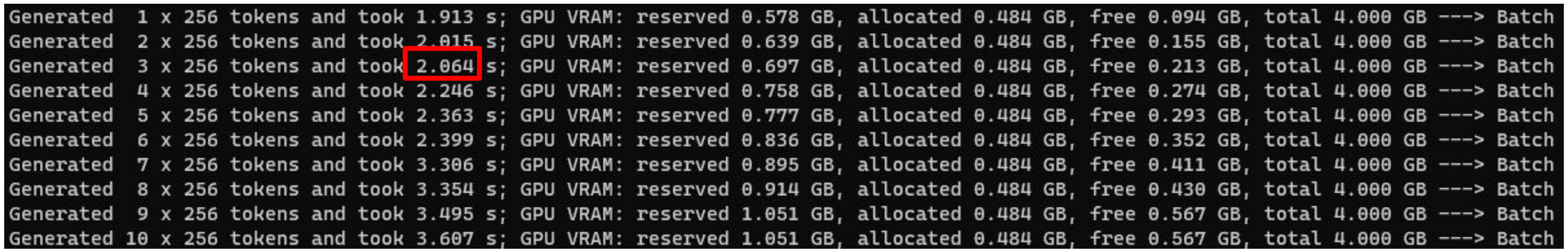
Model	GPT2, around 500 MB
Environment	Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	Multiprocessing effectively circumvents the Python GIL issue. With multiple processes running in parallel, each equipped with its own CUDA default stream, multiprocessing can deliver better performance in inference compared to multithreading. A few challenges arise: 1)Multiple processes performing inferences at different layers/stages on the GPU limit the optimal utilization of the GPU cache. 2)Sharing GPU memory among processes is not straightforward. Each process must independently load its own model into GPU VRAM for inference, resulting in increased VRAM usage. 3)Every process has its own CUDA context (which also consumes VRAM, a few hundred MB). The GPU can only execute one CUDA context at a time and context switching can reduce the GPU utilization. To address this, the NVIDIA MPS server is required to minimize the context switching and the GPU should be set to EXCLUSIVE_PROCESS mode.



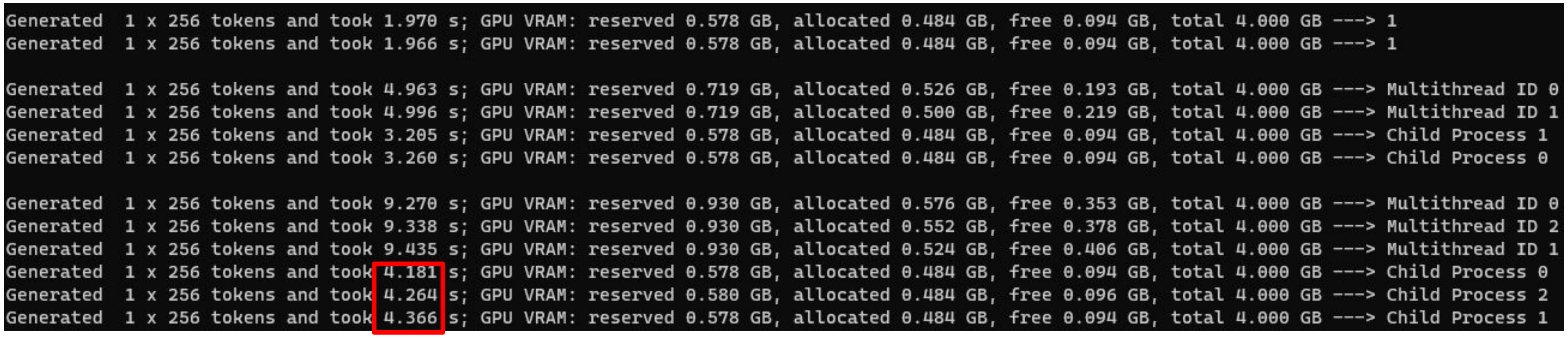
```
Generated  1 x 256 tokens and took 2.593 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 2.827 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1

Generated  1 x 256 tokens and took 5.325 s; GPU VRAM: reserved 0.719 GB, allocated 0.520 GB, free 0.199 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 5.365 s; GPU VRAM: reserved 0.719 GB, allocated 0.500 GB, free 0.219 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 3.663 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 1
Generated  1 x 256 tokens and took 3.674 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 0

Generated  1 x 256 tokens and took 6.422 s; GPU VRAM: reserved 0.930 GB, allocated 0.574 GB, free 0.355 GB, total 4.000 GB ---> Multithread ID 2
Generated  1 x 256 tokens and took 6.452 s; GPU VRAM: reserved 0.930 GB, allocated 0.547 GB, free 0.383 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 6.512 s; GPU VRAM: reserved 0.930 GB, allocated 0.524 GB, free 0.406 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 4.977 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 0
Generated  1 x 256 tokens and took 5.017 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 2
Generated  1 x 256 tokens and took 5.057 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 1
```

Multiprocessing Inference: WSL2, GPT2, A2000

Model	GPT2, around 500 MB
Environment	Windows, 16 Cores and 64GB RAM GPU: A2000, 4GB VRAM PyTorch 2.1.2, Python 3.10, CUDA 11.8, cnDNN 8.7
Result	Inference performance through multiprocessing on WSL2 (4.3 s for 3 processes) surpasses that (5.0 s for 3 processes) on Windows and is far beyond the performance (9.3 s for 3 threads) of multithreading inference on WSL2, but it is much lower than the batch inference (2.0 s when the batch size is 3).



```
Generated  1 x 256 tokens and took 1.913 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Batch
Generated  2 x 256 tokens and took 2.015 s; GPU VRAM: reserved 0.639 GB, allocated 0.484 GB, free 0.155 GB, total 4.000 GB ---> Batch
Generated  3 x 256 tokens and took 2.064 s; GPU VRAM: reserved 0.697 GB, allocated 0.484 GB, free 0.213 GB, total 4.000 GB ---> Batch
Generated  4 x 256 tokens and took 2.246 s; GPU VRAM: reserved 0.758 GB, allocated 0.484 GB, free 0.274 GB, total 4.000 GB ---> Batch
Generated  5 x 256 tokens and took 2.363 s; GPU VRAM: reserved 0.777 GB, allocated 0.484 GB, free 0.293 GB, total 4.000 GB ---> Batch
Generated  6 x 256 tokens and took 2.399 s; GPU VRAM: reserved 0.836 GB, allocated 0.484 GB, free 0.352 GB, total 4.000 GB ---> Batch
Generated  7 x 256 tokens and took 3.306 s; GPU VRAM: reserved 0.895 GB, allocated 0.484 GB, free 0.411 GB, total 4.000 GB ---> Batch
Generated  8 x 256 tokens and took 3.354 s; GPU VRAM: reserved 0.914 GB, allocated 0.484 GB, free 0.430 GB, total 4.000 GB ---> Batch
Generated  9 x 256 tokens and took 3.495 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
Generated 10 x 256 tokens and took 3.607 s; GPU VRAM: reserved 1.051 GB, allocated 0.484 GB, free 0.567 GB, total 4.000 GB ---> Batch
```



```
Generated  1 x 256 tokens and took 1.970 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1
Generated  1 x 256 tokens and took 1.966 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> 1

Generated  1 x 256 tokens and took 4.963 s; GPU VRAM: reserved 0.719 GB, allocated 0.526 GB, free 0.193 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 4.996 s; GPU VRAM: reserved 0.719 GB, allocated 0.500 GB, free 0.219 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 3.205 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 1
Generated  1 x 256 tokens and took 3.260 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 0

Generated  1 x 256 tokens and took 9.270 s; GPU VRAM: reserved 0.930 GB, allocated 0.576 GB, free 0.353 GB, total 4.000 GB ---> Multithread ID 0
Generated  1 x 256 tokens and took 9.338 s; GPU VRAM: reserved 0.930 GB, allocated 0.552 GB, free 0.378 GB, total 4.000 GB ---> Multithread ID 2
Generated  1 x 256 tokens and took 9.435 s; GPU VRAM: reserved 0.930 GB, allocated 0.524 GB, free 0.406 GB, total 4.000 GB ---> Multithread ID 1
Generated  1 x 256 tokens and took 4.181 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 0
Generated  1 x 256 tokens and took 4.264 s; GPU VRAM: reserved 0.580 GB, allocated 0.484 GB, free 0.096 GB, total 4.000 GB ---> Child Process 2
Generated  1 x 256 tokens and took 4.366 s; GPU VRAM: reserved 0.578 GB, allocated 0.484 GB, free 0.094 GB, total 4.000 GB ---> Child Process 1
```

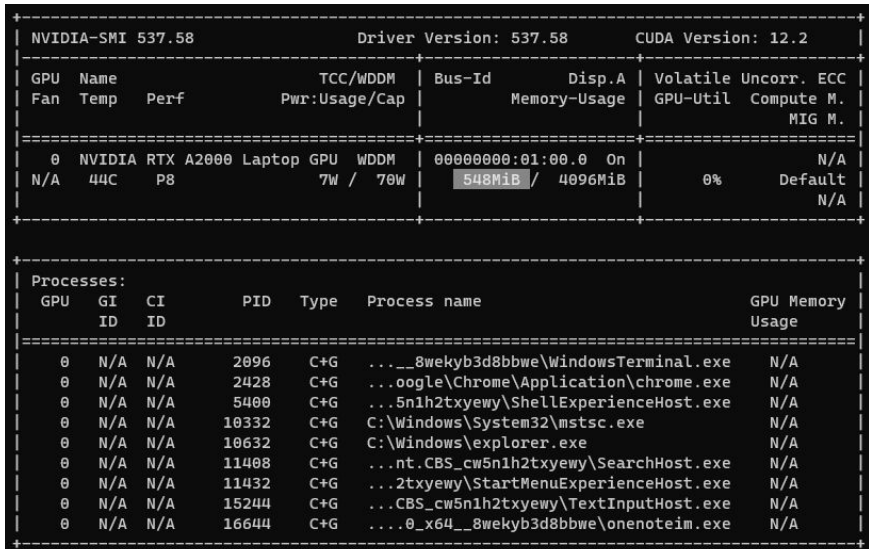
How high the GPU VRAM usage could be on idle?

Data Center GPU is optimized for scientific computing, AI and ML.

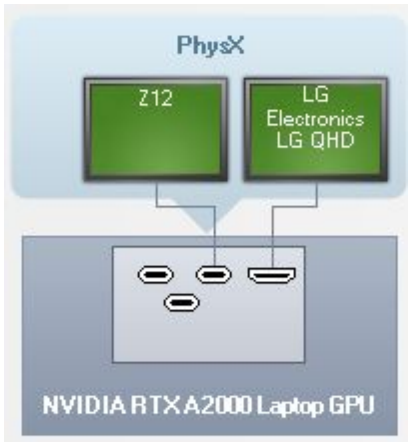
Gaming GPU is designed specifically for rendering high-quality graphics in video games.

Gaming GPU may have video outputs while Data Center GPU does not, which would take 300~1000 MB of VRAM, depending on how many screens that are connected to the GPU, their resolutions and opening apps that have a display output.

Example: the GPU VRAM usage is 0 after my laptop reboots. After being connected to 2 monitors and running some apps (not games or ML jobs), the 500+ MB out of 4 GB VRAM in the laptop is used.

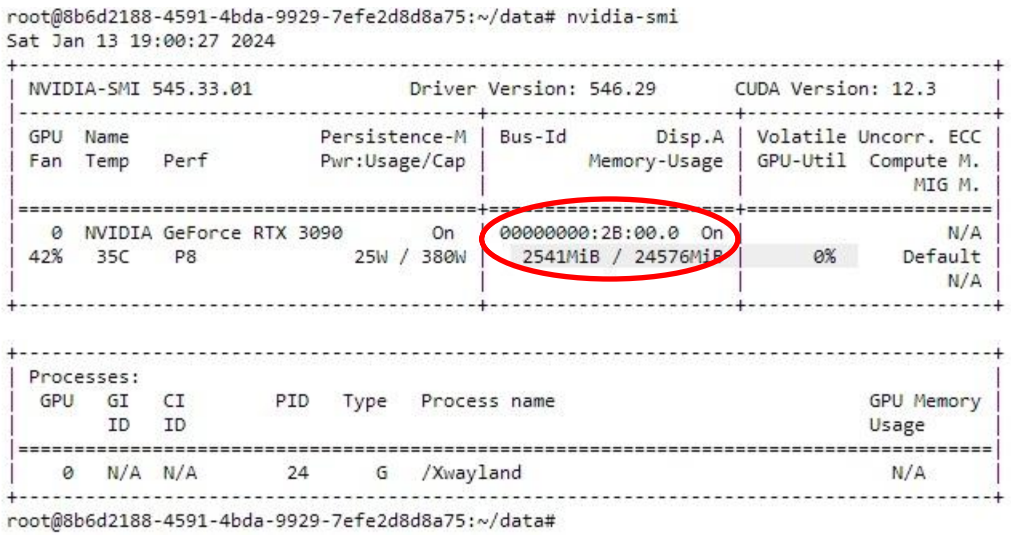


```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 537.58                 Driver Version: 537.58       CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                     TCC/WDDM  | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA RTX A2000 Laptop GPU  WDDM  | 00000000:01:00.0  On |                  N/A |
| N/A   44C    P8               7W /  70W |    548MiB /  4096MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      2096    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A      2428    C+G   ...oogle\Chrome\Application\chrome.exe    N/A      |
|    0   N/A  N/A      5400    C+G   ...5n1h2txyewy\ShellExperienceHost.exe    N/A      |
|    0   N/A  N/A     10332    C+G   C:\Windows\System32\mstsc.exe             N/A      |
|    0   N/A  N/A     10632    C+G   C:\Windows\explorer.exe                   N/A      |
|    0   N/A  N/A     11408    C+G   ...nt.CBS_cw5n1h2txyewy\SearchHost.exe    N/A      |
|    0   N/A  N/A     11432    C+G   ...2txyewy\StartMenuExperienceHost.exe    N/A      |
|    0   N/A  N/A     15244    C+G   ...CBS_cw5n1h2txyewy\TextInputHost.exe    N/A      |
|    0   N/A  N/A     16644    C+G   ....0_x64__8wekyb3d8bbwe\onenoteim.exe    N/A      |
+---------------------------------------------------------------------------------------+
```

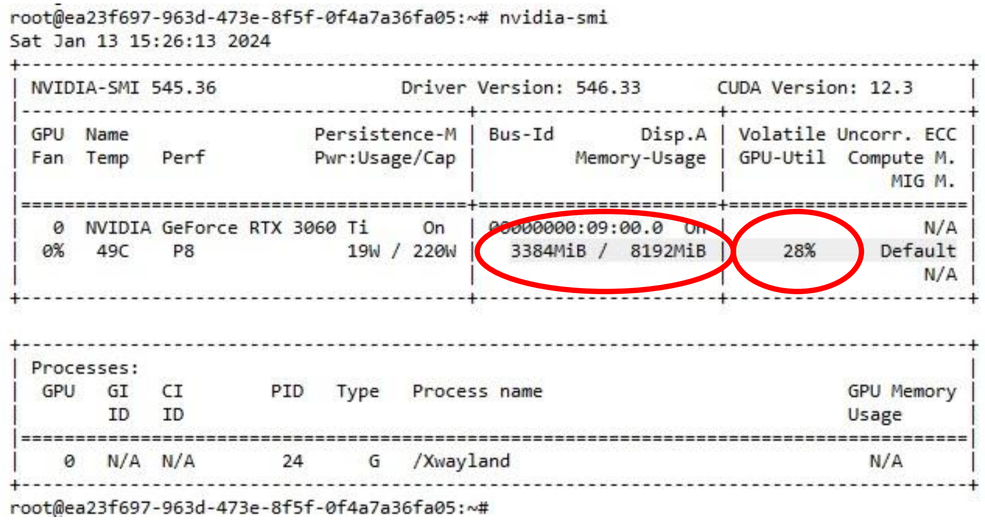


https://www.reddit.com/r/overclocking/comments/uf0rga/how_high_should_be_gpu_vram_usage_on_idle/
https://forums.tomshardware.com/threads/minimizing-gpu-memory-usage-on-windows.3697187/
https://www.daz3d.com/forums/discussion/610226/windows-vram-usage
https://www.techtarget.com/searchstorage/definition/video-RAM

The initial VRAM usage on some nodes



```
root@8b6d2188-4591-4bda-9929-7efe2d8d8a75:~/data# nvidia-smi
Sat Jan 13 19:00:27 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 545.33.01              Driver Version: 546.29       CUDA Version: 12.3     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3090        On  | 00000000:2B:00.0  On |                  N/A |
| 42%   35C    P8              25W / 380W |   2541MiB / 24576MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A        24      G   /Xwayland                                 N/A      |
+---------------------------------------------------------------------------------------+
root@8b6d2188-4591-4bda-9929-7efe2d8d8a75:~/data#
```



```
root@ea23f697-963d-473e-8f5f-0f4a7a36fa05:~# nvidia-smi
Sat Jan 13 15:26:13 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 545.36                 Driver Version: 546.33       CUDA Version: 12.3     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3060 Ti     On  | 00000000:09:00.0  On |                  N/A |
|  0%   49C    P8              19W / 220W |   3384MiB /  8192MiB |     28%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A        24      G   /Xwayland                                 N/A      |
+---------------------------------------------------------------------------------------+
root@ea23f697-963d-473e-8f5f-0f4a7a36fa05:~#
```

- The JupyterLab container workloads are just running without loading models...
- With the Windows GPU-PV technology, Windows and WSL2 applications can share the same GPU dynamically, based on who need what.
- It appears these nodes are running some apps by their owners that make use of its GPU.
- When the VRAM on a GPU is fully utilized, the Windows system may use the RAM as a substitute, leading to decreased performance compared to the GPU operating within its dedicated VRAM.
- Among the selected nodes, most are deemed suitable, with at least 90% of their VRAM available for container workloads.

https://www.reddit.com/r/StableDiffusion/comments/x308i4/how_does_shared_gpu_memory_work/

The fixed overhead on VRAM usage to run a PyTorch App

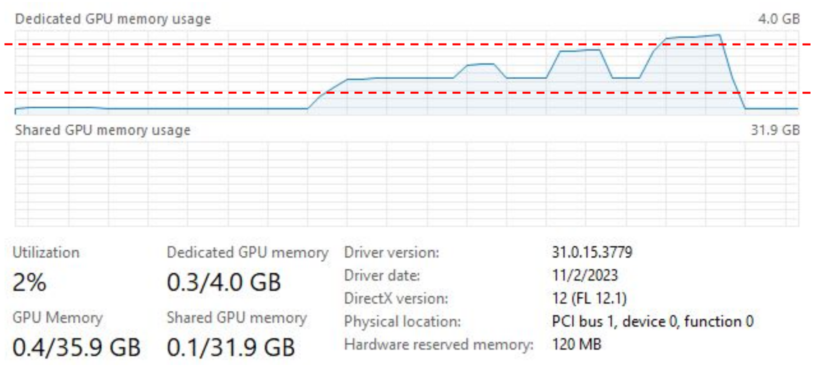
- Each process establishes a CUDA context on the GPU, consuming approximately a few hundred MB, which could be different on different platforms.
- Additionally, both the PyTorch and cuDNN libraries collectively occupy a few hundred MB, and this memory is shared across all processes.
- While these VRAM usages (CUDA context and libraries) are not directly visible within the individual processes, they can be observed through tools such as the Windows Task Manager or ‘nvidia-smi’.
- In the context of inference for each process, VRAM usage encompasses the model, input tensors, and the self-attention KV cache.

https://discuss.pytorch.org/t/what-is-the-initial-1-3gb-allocated-vram-when-first-using-cuda/122079
https://discuss.pytorch.org/t/reduce-gpu-memory-blocked-by-context/142587

VRAM Usage for GPT2 Inference: WSL2 and Windows

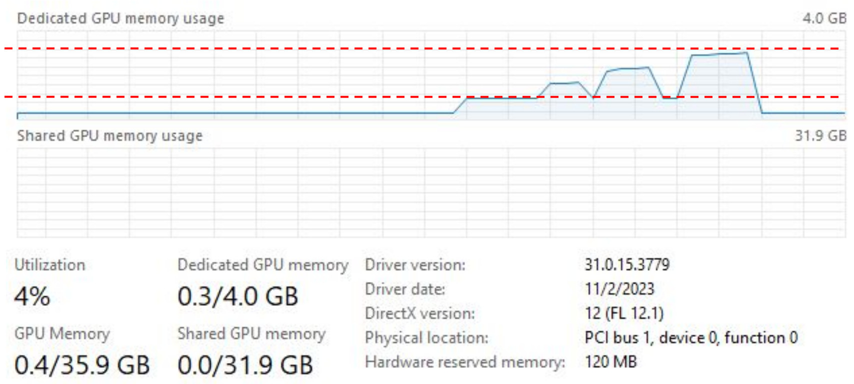
Within each process, the VRAM usage is similar on WSL2 and Windows, around 0.58 GB.
It seems Windows needs more VRAM for the CUDA context and libraries.

	Initial	**1st Process**	**2nd Process**	**3rd Process**	**4th Process**
	Video outputs and rendering	Model, Input & Cache, 0.58 GB; CUDA Context; Libraries	Model, Input & Cache, 0.58 GB; CUDA Context	Model, Input & Cache, 0.58 GB; CUDA Context	Model, Input & Cache, 0.58 GB; CUDA Context
Windows	0.3 GB	1.8 GB	2.4 GB	3.1 GB	3.8 GB
WSL2	0.3 GB	1.0 GB	1.7 GB	2.4 GB	3.1 GB



Utilization
2%
Dedicated GPU memory
0.3/4.0 GB
GPU Memory
0.4/35.9 GB
Shared GPU memory
0.1/31.9 GB

Driver version:	31.0.15.3779
Driver date:	11/2/2023
DirectX version:	12 (FL 12.1)
Physical location:	PCI bus 1, device 0, function 0
Hardware reserved memory:	120 MB



Utilization
4%
Dedicated GPU memory
0.3/4.0 GB
GPU Memory
0.4/35.9 GB
Shared GPU memory
0.0/31.9 GB

Driver version:	31.0.15.3779
Driver date:	11/2/2023
DirectX version:	12 (FL 12.1)
Physical location:	PCI bus 1, device 0, function 0
Hardware reserved memory:	120 MB

Running inference in Windows **Running inference in WSL2**

Summary

- The preferred approach involves grouping incoming requests and conducting batch inference, a strategy that can significantly boost performance and enhance resource utilization by efficiently utilizing the GPU cache and parallel processing capabilities.

- In practical terms, it's advisable to employ two types of processes: one dedicated to I/O- and CPU-bound tasks (downloading/uploading, preprocessing, and post-processing), and another focused on GPU inference to overcome various known limitations, such as Python GIL, VRAM usage issue with multiprocessing, and synchronous inference that is not asyncio-compatible.

- While WSL2 excels over Windows in several aspects, including single, batch, and multiprocessing inference, it does face significant performance issues with multithreading inference.

- When select the VRAM on nodes, several factors need to consider:

$$\boxed{\text{The available VRAM for model \& inference}} = \boxed{\text{Physical VRAM}} - \boxed{\text{The initial usage (video output, rending \& gaming)}} - \boxed{\text{The CUDA context and PyTorch/cuDNN libraries}}$$