Speeding NVLM Decoder Inference Computation

NYCU-HPC-Team2

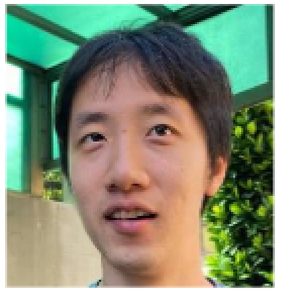
NYCU HPC team2

Nvidia Mentors · Jay Chen ·Shijie Wang

NYCU SDC HPC team

Dr.Meng-Hsun Tsai

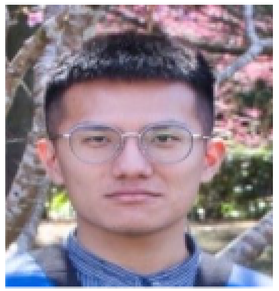
Shun-Yu Yang



Chen-Kai Chang

Zong-Hua Wu

Jia-Hui Shen



Cheng-Wei Lin

Chu-Siang Tseng

Speeding NVLM Decoder Inference Computation

- **Problem the team is trying to solve.**
  - Our team aims to accelerate the inference speed of the NVLM model to meet the increasing demand for real-time applications while reducing latency and memory resource usage.

- **Scientific driver for the chosen algorithm.**
  - **Reduce Computational Costs**
    - With efficient algorithms decrease the computational load on hardware, leading to lower energy consumption and operational expenses, which is crucial for sustainable AI deployment.
  - **Speed up the inference time**
    - Reduced inference time allows handling more requests within the same time frame, making the system more scalable and efficient.

- **What's the algorithmic motif?**
  - **Tensor Parallelism , Chat function , Generate function, KV Cache, Flash Attention**

- **What parts are you focused on?**
  - The team focused on four main areas:
  - **Pipeline Parallelism**: Improving throughput by utilizing larger batches in the pipeline.
  - **Tensor Parallelism**: Splitting the weights of the linear layers across GPUs to reduce latency.
  - **Flash Attention**: Highly optimized kernels for faster, memory-efficient attention computations.
  - **KV Cache**: Leveraging caching mechanisms to significantly enhance inference speed.

Speeding NVLM Decoder Inference Computation

- **What's NVLM?**
  - NVLM (NVIDIA Vision Language Model) is designed to excel in both vision-language and text-only tasks by integrating advanced techniques that combine visual and textual information, enabling it to perform a wide range of applications such as OCR, visual reasoning, document understanding, and mathematical reasoning in a visual context; furthermore, it improves its text-only performance after multimodal training by incorporating a high-quality text dataset during fine-tuning.

- **Why NVLM?**
  - By first optimizing the inference speed of this vision-language model, we can then apply the same techniques to improve performance in other large language models down the line.
  - Ensuring Broad Applicability of Techniques
  - Leveraging Tight Integration with NVIDIA Hardware

Evolution and Strategy

- **What was your goal for coming here?**
  - Learn about different **speedup methods** and be able to successfully implement them
  - Interact with and learn from experts of diverse fields from other teams
  - Learning new knowledge from the mentor and applying it to this project.

- **What was your initial strategy?**
  - Using JAX to rewrite NVLM decoder only model to increase inference speed.

- **How did this strategy change?**
  - Since NVLM is a merge of many models (such as Qwen2, InternVision), we figured that rewriting them all using JAX would be too much work.

Results and Final Profile

- **What were you able to accomplish?**
  - Developed a scalable pipeline that utilizes parallelism to maximize GPU usage.
  - Using Tensor Parallelism to split weight of linear layers to different GPUs to maximize GPU utilization
  - Implemented Flash Attention to reduce memory usage while improving speed.
  - Integrated KV Cache to achieve a substantial reduction in redundant computations during inference.

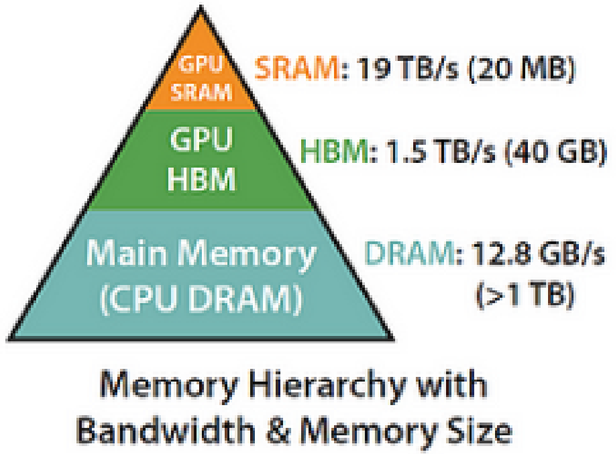
- **Did you achieve a speed up?**
  - We achieved a 3.3x speedup in inference

- **What did you learn?**
  - How to using Nsight Profiling.
  - Different ways to speed up LLM inference.
  - ~~Testing how many days my body can go without sleep.~~

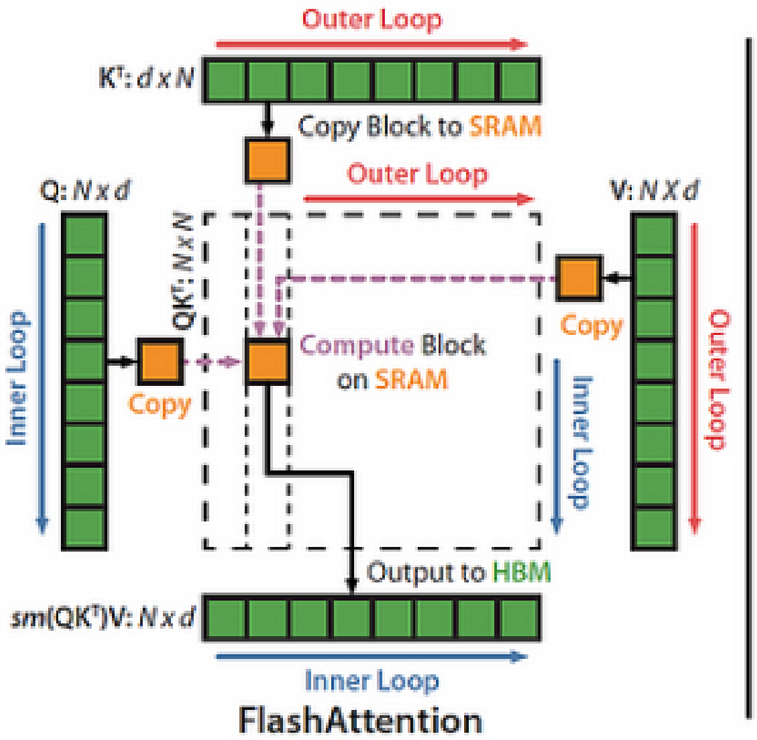
Method

	Flash Attention	KV Cache	Tensor Parallelism	Pipeline Parallelism	Batch Scheduling	Thread Pool Executor
Benchmark	✓	✓		✓		
Dataloader Handle Batch input	✓	✓		✓	✓	
Thread-based concurrency	✓	✓		✓		✓
vllm	✓	✓	✓		✓	

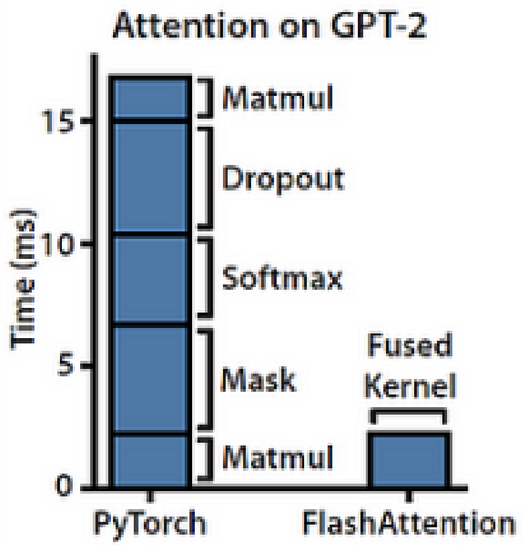
Method: Flash Attention



Memory Hierarchy with Bandwidth & Memory Size



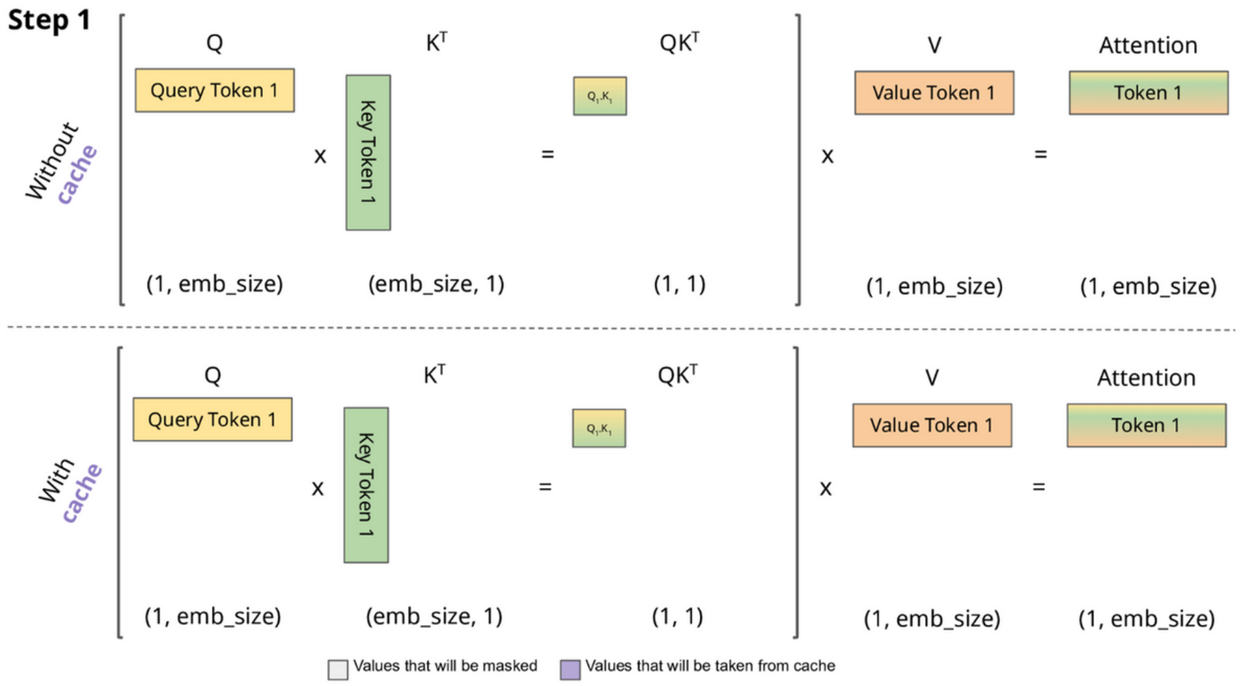
FlashAttention



Attention mechanism optimized for faster and more memory-efficient computation of attention. It utilizes **tiling strategies to keep intermediate activations in fast/on-chip memory**, and minimizes memory usage by avoiding explicit computation and storage of large attention matrices. With flash attention, we can efficiently handle softmax and attention, allowing larger models and longer sequence lengths.

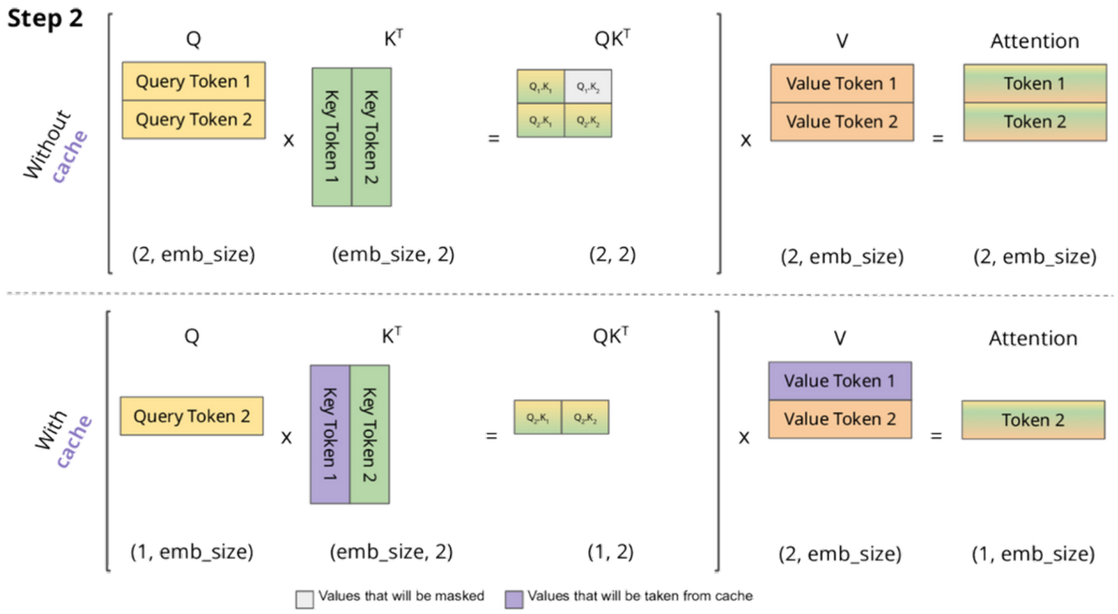
Method :KV Cache

Memory optimization(Key-Value (KV) Cache): The **Key-Value Cache** is a mechanism used during inference to store intermediate computation results (attention keys and values) for reuse in subsequent steps of a sequence



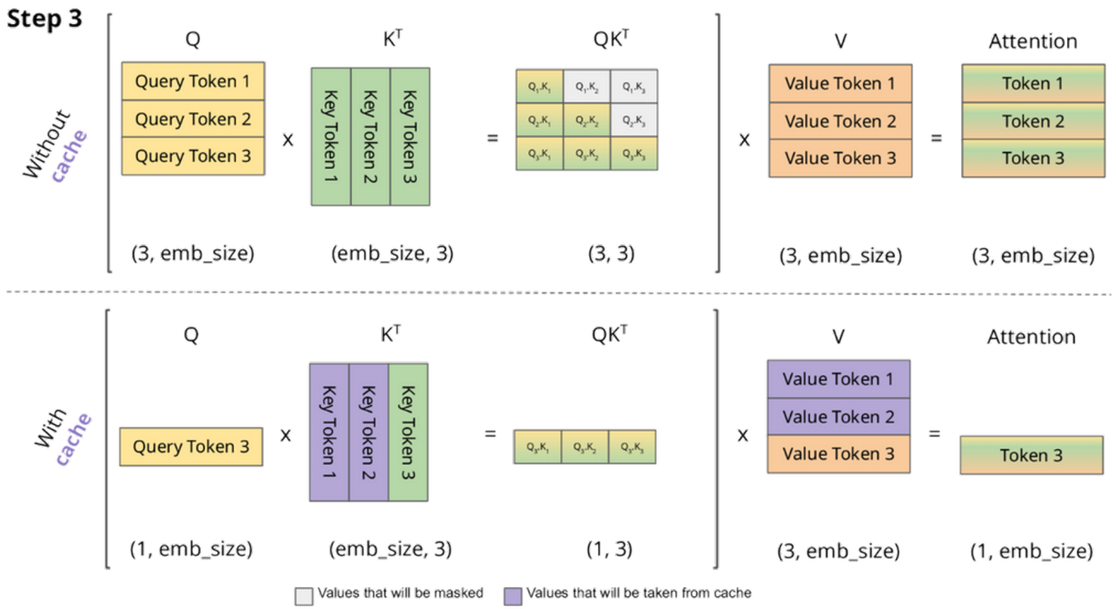
Method :KV Cache

Memory optimization(Key-Value (KV) Cache): The **Key-Value Cache** is a mechanism used during inference to store intermediate computation results (attention keys and values) for reuse in subsequent steps of a sequence



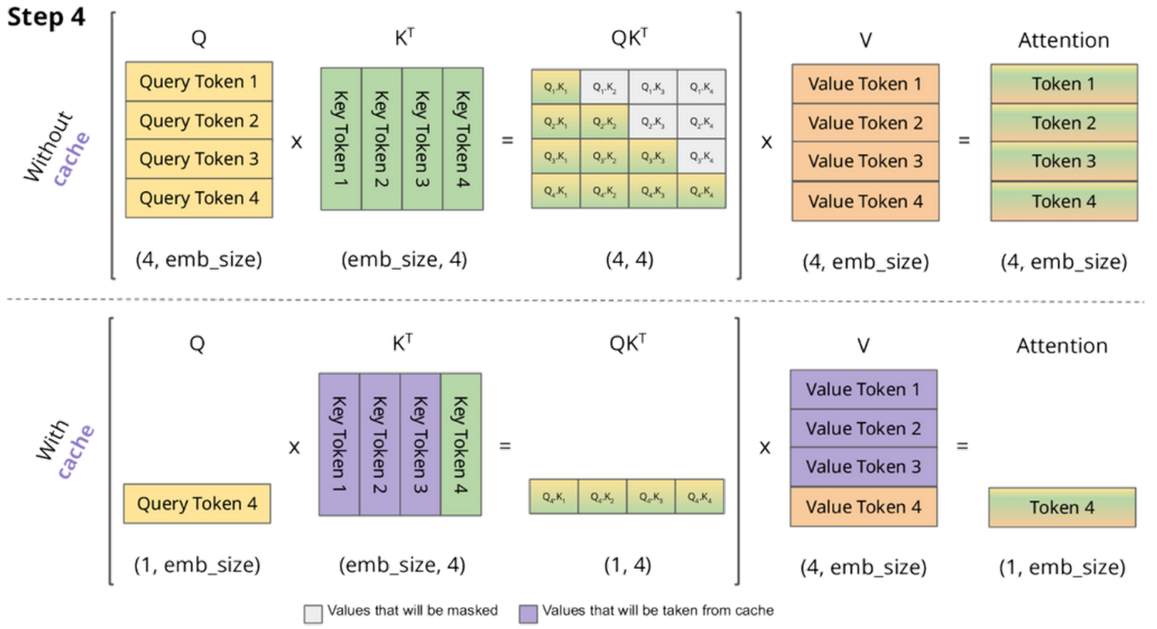
Method :KV Cache

Memory optimization(Key-Value (KV) Cache): The **Key-Value Cache** is a mechanism used during inference to store intermediate computation results (attention keys and values) for reuse in subsequent steps of a sequence



Method :KV Cache

Memory optimization(Key-Value (KV) Cache): The **Key-Value Cache** is a mechanism used during inference to store intermediate computation results (attention keys and values) for reuse in subsequent steps of a sequence

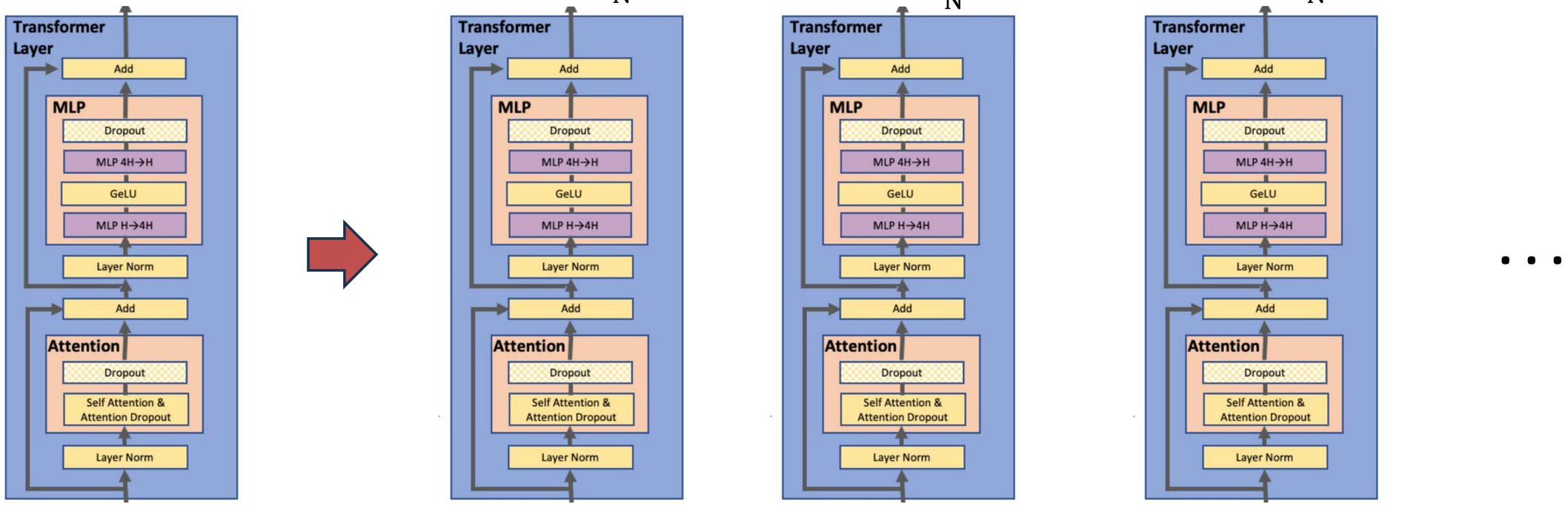


Method: Batch Scheduling

Batch Scheduling :
Batch scheduling is a technique used to improve the efficiency of inference in machine learning models by grouping multiple input queries into a single batch for processing. This approach leverages parallelism, optimizes hardware utilization, and reduces the overhead of processing queries individually..

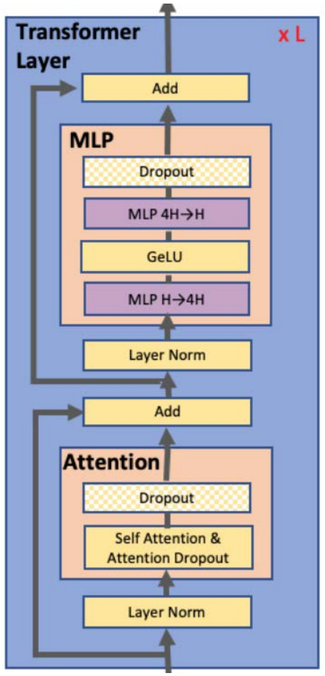
Method :Pipeline Parallelism

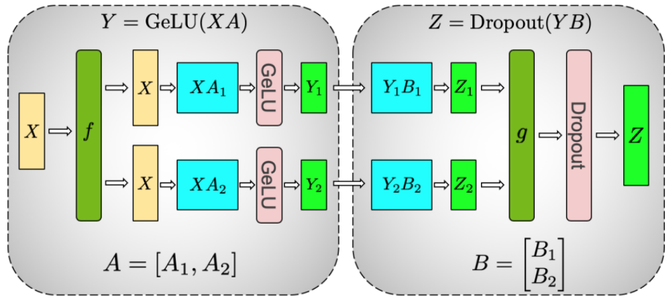
Pipeline parallelism : is a technique to optimize the execution of large models by dividing the model into sequential chunks (stages) and assigning each chunk to a separate GPU. Each GPU handles a specific portion of the computation pipeline and passes its output to the next GPU, much like an assembly line.



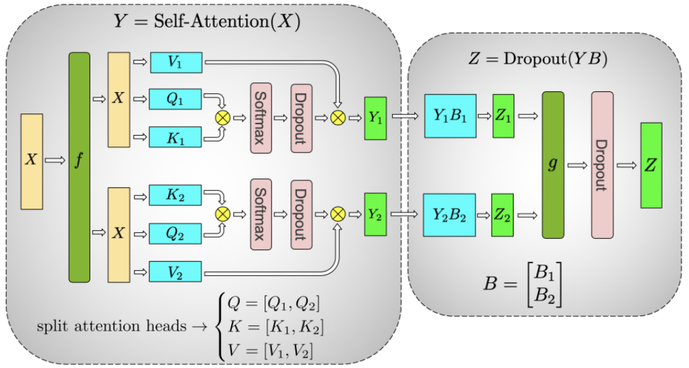
Method :Tensor Parallelism

Parallelization(Tensor Parallelism): Tensor Parallelism is a technique that splits computations within a single model layer across multiple GPUs. Unlike pipeline parallelism, which divides layers across GPUs, tensor parallelism focuses on parallelizing the operations inside a single layer





(a) MLP



(b) Self-Attention

Benchmark

- Implement pipeline parallelism to split the model.
- Utilize the KV cache to enhance memory efficiency and speed up inference.
- Enable Flash Attention (originally set to False) by changing it to True, achieving a 10% performance improvement.
- The chat function just could handle single question

Method : Dataloader Handle Batch input

Using the Dataloader to Handle Multiple Inputs from Users and Increase Total Throughput.

Adjust the **chat** function and **generate** function in the source code to support batch processing of user questions.

1. **Dataloader for Batch Inputs**: Add the dataloader to accept multiple inputs and group them into batches. This will enable parallel processing and increase throughput.
2. **Chat Function Adjustment**: Redesign the chat function to process a batch of inputs in one go, rather than handling each input individually. Ensure the function supports asynchronous or parallel processing if applicable.
3. **Generate Function Update**: Modify the generate function to handle batch processing. Ensure tokenization, model inference, and response decoding are optimized for batch execution. This involve padding, truncation, and leveraging efficient model inference frameworks.

Method :Thread-based concurrency

Parallel Execution Using ThreadPoolExecutor

1. **Concurrency**: Enables concurrent processing of multiple user inputs, significantly reducing response time for batch questions.
2. **Efficiency**: Threads are reused within the pool, avoiding the overhead of creating and destroying threads repeatedly.

Method :vllm

vLLM is a high-performance method designed to optimize large language models for efficient inference.

1. **Memory optimization(Key-Value (KV) Cache) :** The **Key-Value Cache** is a mechanism used during inference to store intermediate computation results (attention keys and values) for reuse in subsequent steps of a sequence.
2. **Parallelization(Tensor Parallelism) :** Tensor parallelism splits computations within a single layer across multiple GPUs
3. **Batch Scheduling** : Groups multiple queries into a single batch for parallel processing.

Method

	Flash Attention	KV Cache	Tensor Parallelism	Pipeline Parallelism	Batch Scheduling	Thread Pool Executor
Benchmark	✓	✓		✓		
Dataloader Handle Batch input	✓	✓		✓	✓	
Thread-based concurrency	✓	✓		✓		✓
vllm	✓	✓	✓		✓	

Inference Questions (Text Only Version)

question1 = 'You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.Merge nums1 and nums2 into a single array sorted in non-decreasing order.The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.'

question2 = 'Given an array nums of size n, return the majority element.The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.'

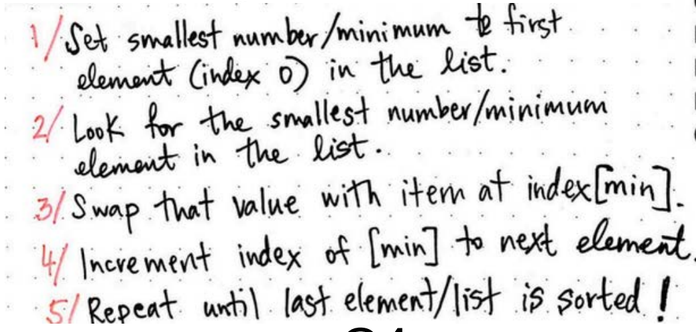
question3 = 'Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.'

question4 = 'Given a string s containing just the characters (,), {, }, [and], determine if the input string is valid.An input string is valid if:Open brackets must be closed by the same type of brackets.Open brackets must be closed in the correct order.Every close bracket has a corresponding open bracket of the same type.'

Inference Questions (Image + text)

question1 = ["<image>\nExplain why this meme is funny"]
question2 = ["<image>\nWhat is the dierence between the left, middle and right object in the image?"]
question3 = ["<image>\nYou are a helpful driving assistant. In this scene, which lane should I choose and why?"]
question4 = ["<image>\nWrite code based on the provided pseudo code"]
question5 = ["<image>\nAccording to the table, explain which food is the most likely cause of the outbreak of food poisoning?",]
question6 = ["<image>\nWhat percentage of market share does NVIDIA have for data center GPUs in 2023?"]

Q1



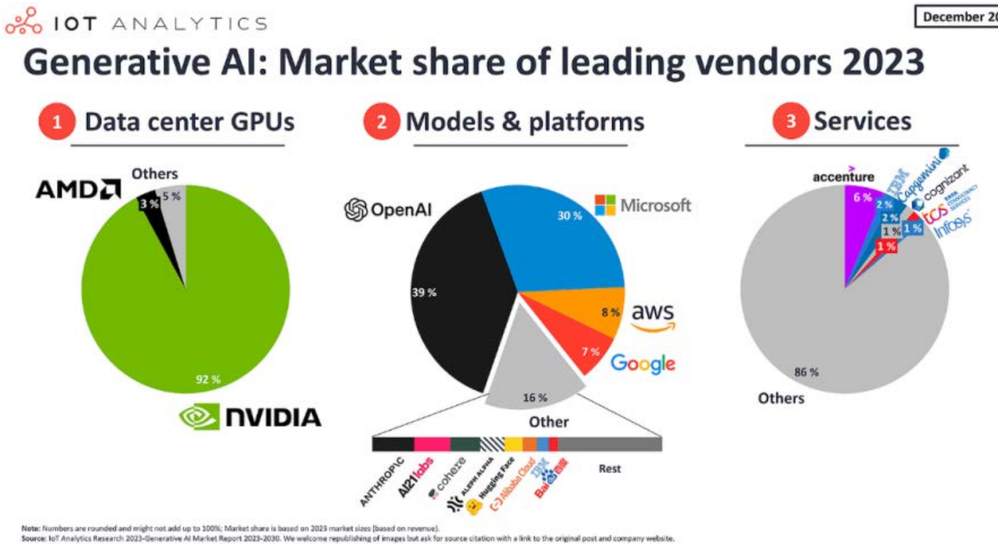
Q4

Q2

Food	Number of people who ate that food	Number of people who ate the food and got sick
Cold chicken	86	34
Potato salad	54	38
Egg sandwiches	76	40
Fruit pie and cream	32	12
Cheese	48	12

Q5

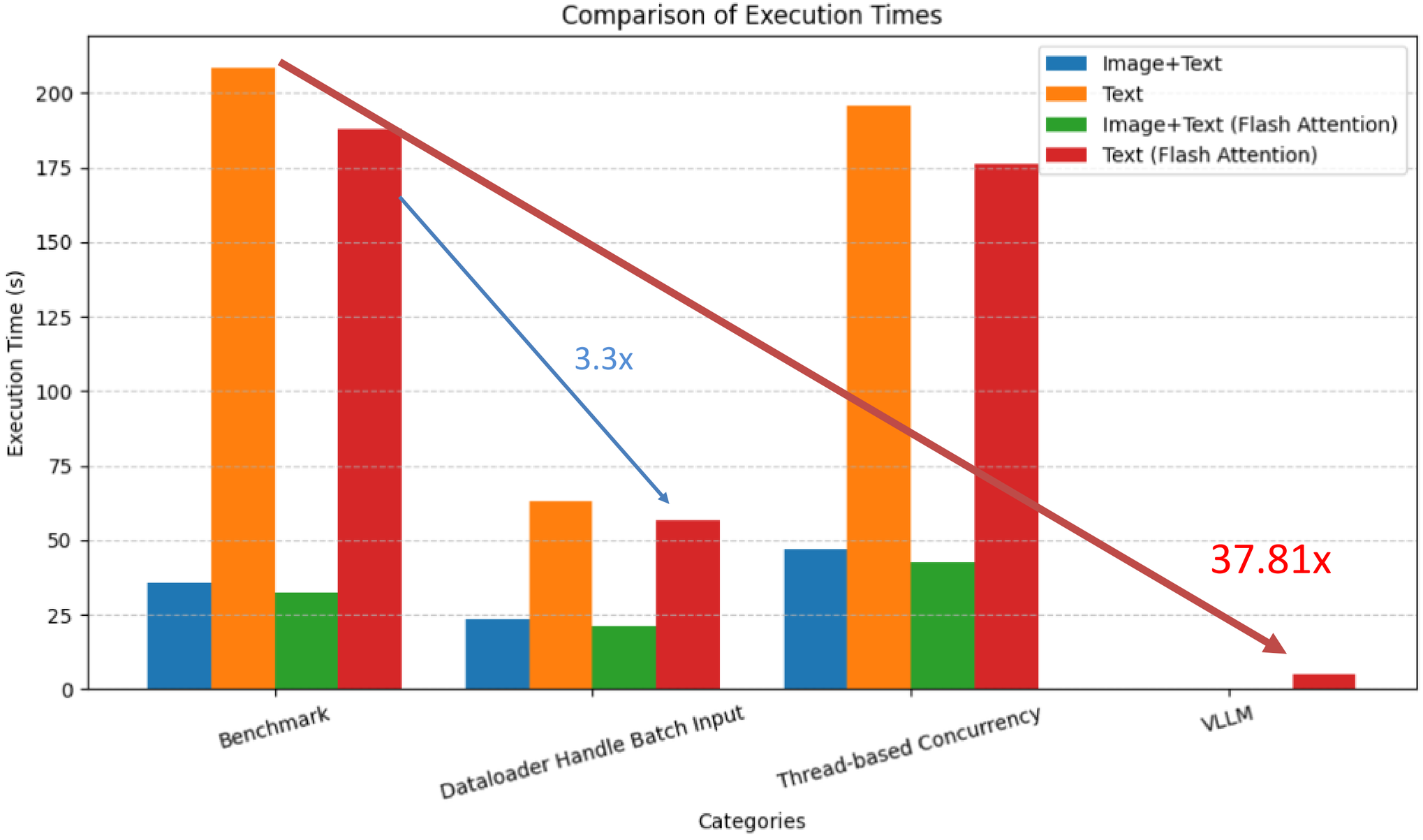
Q3



OpenACC
Make Science, Less Programming

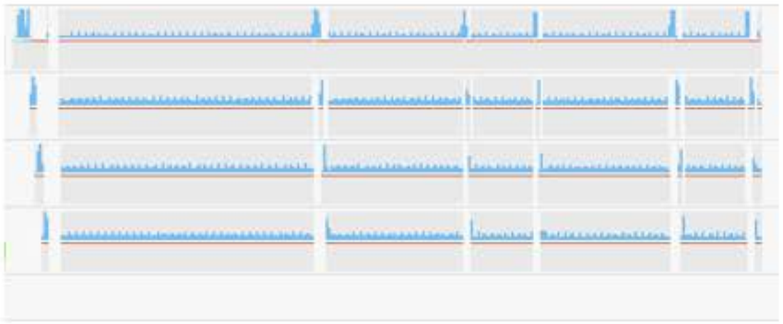
Result

	TEXT + IMAGE	TEXT ONLY (4 questions)	TEXT ONLY+Flash attention (4 questions)
Benchmark	35.94s	208.60s	187.80s
Dataloader Handle Batch input	23.72s	63.07s	56.76s
Thread-based concurrency	47.18s	195.93s	177.3s
vLLM	----------	----------	5.50s

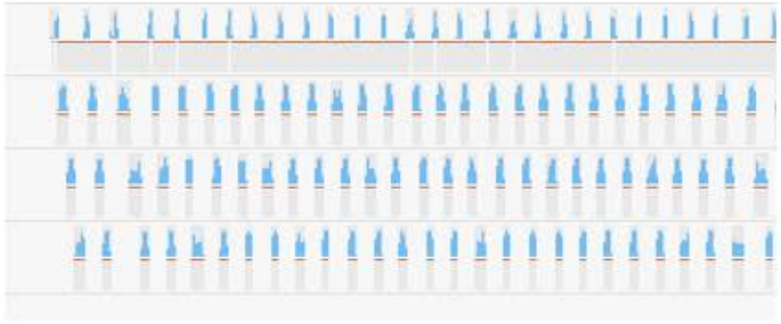


Result : Nsight Profile

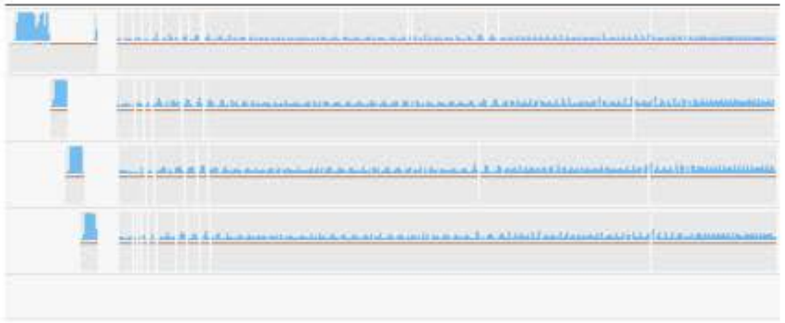
Benchmark



Zoom in

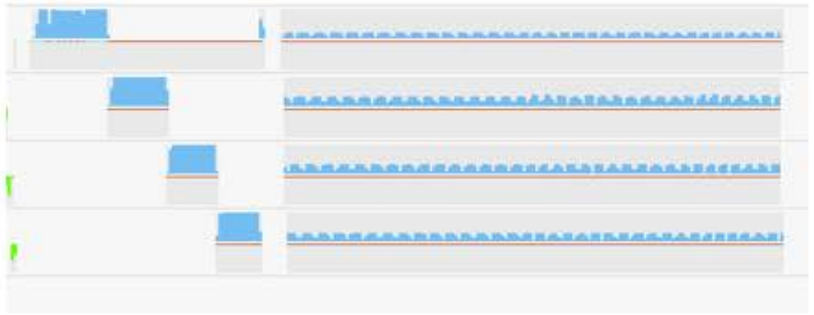


Thread-based concurrency

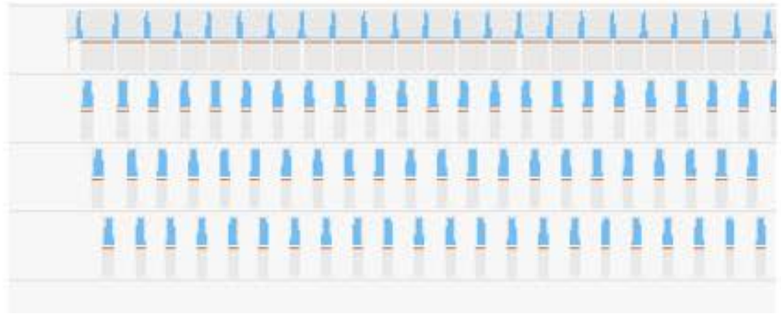


Zoom in

Dataloader Handle Batch input



Zoom in



We also try...

deepspeed

- why choose deepspeed
  - DeepSpeed brings together in parallelism technology such as tensor, pipeline and ZeRO-parallelism, and combines them with high performance custom inference kernels
  - don’t require any change on the modeling side
  - automatically partition the model as necessary

"total_time": 35
"throughput": 33.9,
"num_questions": 6

```
Tue Dec  3 17:43:54 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.154.05             Driver Version: 535.154.05   CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------+
| 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 A100-SXM4-40GB          On  | 00000000:07:00.0 Off |                    0 |
| N/A   36C    P0              62W / 400W |  33555MiB / 40960MiB |      9%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM4-40GB          On  | 00000000:0F:00.0 Off |                    0 |
| N/A   33C    P0              60W / 400W |  25955MiB / 40960MiB |     12%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM4-40GB          On  | 00000000:47:00.0 Off |                    0 |
| N/A   33C    P0              62W / 400W |  25955MiB / 40960MiB |     12%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM4-40GB          On  | 00000000:4E:00.0 Off |                    0 |
| N/A   36C    P0              67W / 400W |  25955MiB / 40960MiB |     12%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   4  NVIDIA A100-SXM4-40GB          On  | 00000000:B7:00.0 Off |                    0 |
| N/A   37C    P0              60W / 400W |  25955MiB / 40960MiB |     24%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   5  NVIDIA A100-SXM4-40GB          On  | 00000000:BD:00.0 Off |                    0 |
| N/A   37C    P0             123W / 400W |  19177MiB / 40960MiB |      9%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

can run question parallel

but costs more time and memory

need future study

"total_time": 66.84
"throughput": 18.15
"num_questions": 6

```
Every 0.1s: nvidia-smi

Mon Dec  2 12:32:15 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.154.05             Driver Version: 535.154.05   CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------+
| 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 A100-SXM4-40GB          On  | 00000000:07:00.0 Off |                    0 |
| N/A   42C    P0             189W / 400W |  35170MiB / 40960MiB |     89%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM4-40GB          On  | 00000000:0F:00.0 Off |                    0 |
| N/A   38C    P0             181W / 400W |  35314MiB / 40960MiB |     90%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM4-40GB          On  | 00000000:47:00.0 Off |                    0 |
| N/A   38C    P0             188W / 400W |  35572MiB / 40960MiB |     90%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM4-40GB          On  | 00000000:4E:00.0 Off |                    0 |
| N/A   41C    P0             200W / 400W |  35572MiB / 40960MiB |     90%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   4  NVIDIA A100-SXM4-40GB          On  | 00000000:B7:00.0 Off |                    0 |
| N/A   43C    P0             180W / 400W |  35572MiB / 40960MiB |     90%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   5  NVIDIA A100-SXM4-40GB          On  | 00000000:BD:00.0 Off |                    0 |
| N/A   42C    P0             187W / 400W |  35300MiB / 40960MiB |     89%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

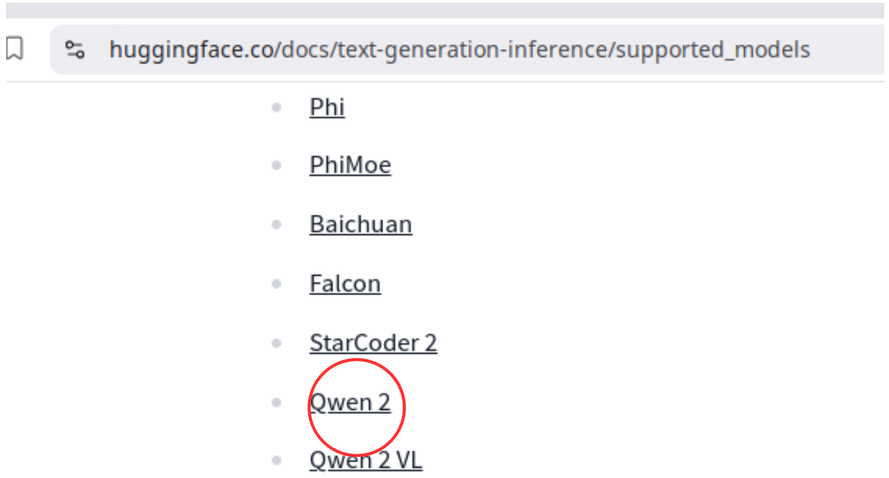
OpenACC
More Science, Less Programming

National Center for High-performance Computing

NVIDIA

What problems have you encountered?

- Issue with hugging face API
  - Unable to use the hugging face API to enable tensor parallel (even though the model we want to accelerate is listed as a supported model).
  - Initially thought to be because of improper splitting of model (pipeline parallel). Digging deeper found that it's more complicated.



What problems have you encountered?

- Issue with hugging face API
  - Unable to use the hugging face API to enable tensor parallel (even though the model we want to accelerate is listed as a supported model).
  - Initially thought to be because of improper splitting of model (pipeline parallel). Digging deeper found that it's more complicated.

```
  path = "nvidia/NVLM-D-72B"
  device_map = split_model()
  model = AutoModel.from_pretrained(
      path,
      torch_dtype=torch.bfloat16,
      low_cpu_mem_usage=True,
      use_flash_attn=False,
      tp_plan="auto",
      trust_remote_code=True,
      device_map=device_map).eval()
```

```
  File "/workspace/NCHC_hackthon/inference_qwen2.py", line 4,
    model = AutoModelForCausalLM.from_pretrained(
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/workspace/NCHC_hackthon/transformers/src/transformers
    return model_class.from_pretrained(
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/workspace/NCHC_hackthon/transformers/src/transformers
    assert tp_device is not None, "tp_device not set!"
           ^^^^^^^^^^^^^^^^^^^^
AssertionError: tp_device not set!
(NVLM) root@0668254ef66a:/workspace/NCHC_hackthon#
```

What problems have you encountered?

No profiling report is generated in Nsight

Key Challenges:

- Want to using **Nvtx : start_range and end_range**
- Cannot generate the profiling report, but it runs accurately.

Solution:

- Add environment variable -e NSYS_NVTX_PROFILER_REGISTER_ONLY=0

```
NSYS_NVTX_PROFILER_REGISTER_ONLY=0 nsys profile -t
cuda,nvtx,osrt,cudnn,cublas --stats=true --force-overwrite=true --capture-range nvtx --
nvtx-capture "range_name" -o nsysFile your_python
```

Energy Efficiency

ANNUAL ENERGY SAVINGS PER GPU NODE			
	Intel Dual SPR 8480c	8x H100 80GB SXM4	Power Savings
Compute Power (kWh/year)	1,136	81,380	(80,245)
Networking Power (kWh/year)	48	1,627	(1,579)
Total Power (kWh/year)	**1,184**	**83,008**	**(81,824)**
$/kWh	0.18		
Annual Cost Savings	(14,728.31)		
3-year Cost Savings	(44,184.92)		
Metric Tons of CO_2	(58)		
Gasoline Cars Driven for 1 year	(13)		
Seedlings Trees grown for 10 years	(959)		

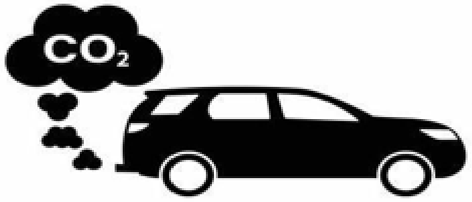
ASSUMPTIONS

(1) The workload being input will run 24/7/365 on the node in question

INPUTS	
# CPU Cores	2
# GPUs (H100)	4
Application Speedup	3.3x

GPU nodes required CPU nodes required

(13)



Gasoline cars driven for a year

(959)

Trees growing for 10 years

(58)

Metric tons of CO_2

Speical Thanks

Nvidia Mentors · Jay Chen ·Shijie Wang

- Johnson Sun

Final Thoughts

- Was this Open Hackathon worth it?
  - Yes, it was incredibly valuable. We gained deep insights into both the theoretical and practical aspects of model speedup methods. The experience not only enhanced our knowledge but also sparked our interest in this domain.

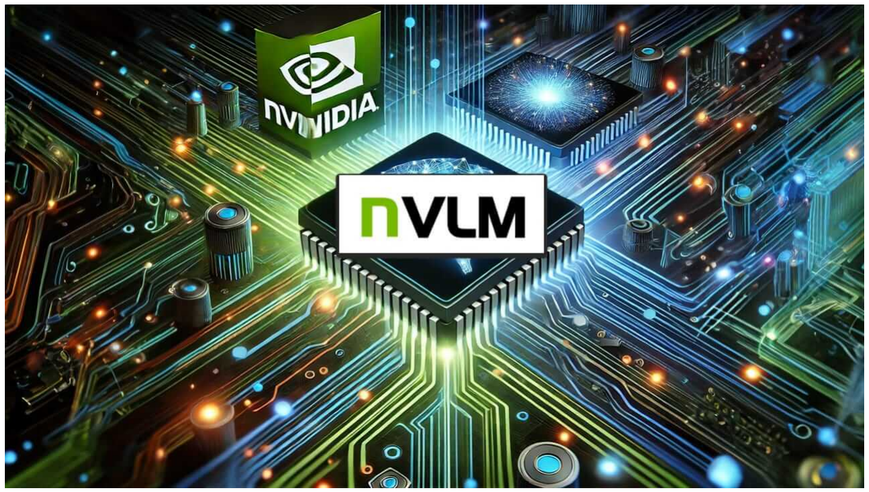
- Will you continue development?
  - Absolutely. The hackathon has inspired us to explore deeper into HPC.

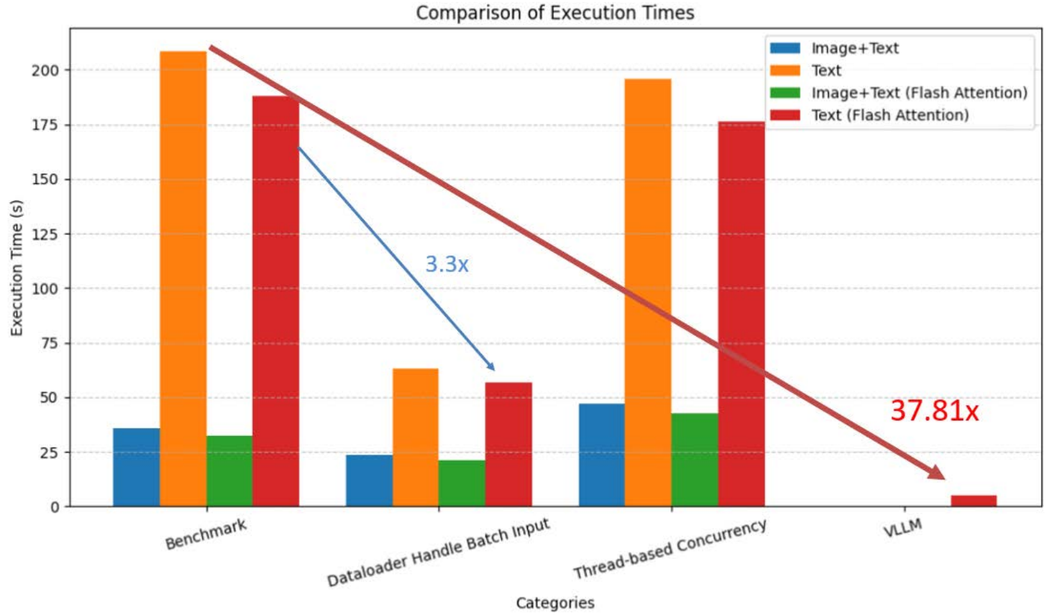
- Next steps, future plans.
  - Enhance our technical skills in HPC.
  - Continue to achieve the initial goal, rewrite the model in JAX framework.

- What sustained resources or support will be critical for your work after the event?
  - Continued mentorship and online resources would be incredibly useful
  - A100 && H100

In NCHC web

Speeding NVLM Decoder Inference Computation





在NVLM推理上的加速研究是一個極具潛力的領域。NVLM（NVIDIA Vision Language Model）是一種尖端的多模態大模型，其結合了文本和圖像的處理能力，並在視覺-語言任務上達到了最先進的成果。加速推理過程能夠顯著降低計算資源和時間的需求，使這些強大的模型能夠更快地應用於現實情境，例如圖像識別、文檔分析、以及視覺問答等應用。
這個領域的重要性在於多模態模型的廣泛應用潛力，尤其是那些需要即時回應的場景，如自動駕駛、醫療診斷或智能助理系統等。加速NVLM的推理能讓模型更加高效和實時，進一步推動其在工業界和日常生活中的普及和應用。
目前的加速研究已經顯示出明顯的成果，透過優化模型架構（例如混合架構和動態高解析度影像處理），推理速度提升了，並且能在保持甚至提升模型準確性的同時減少計算成本。這樣的加速帶來了多方面的影響，不僅降低了硬體資源的消耗，還使得這些多模態應用更為普及，進而推動整個AI領域向著更高效、更智能的方向發展。

	TEXT + IMAGE	TEXT ONLY (4 questions)	TEXT ONLY+Flash attention (4 questions)
Benchmark	35.94s	208.60s	187.80s
Dataloader Handle Batch input	23.72s	63.07s	56.76s
Thread-based concurrency	47.18s	195.93s	177.3s
vLLM	----------	----------	5.50s