Team-05 NYCU_Bocchi the CUDA!
Accelerate RL for LLM

Team-05 NYCU_Bocchi the CUDA!
Accelerate RL for LLM

NVIDIA Mentors

Shijie-Wang

Cheng-Zong Li

CHU-SIANG (Johnson) TSENG

Virginia Chen

Ryan Huang

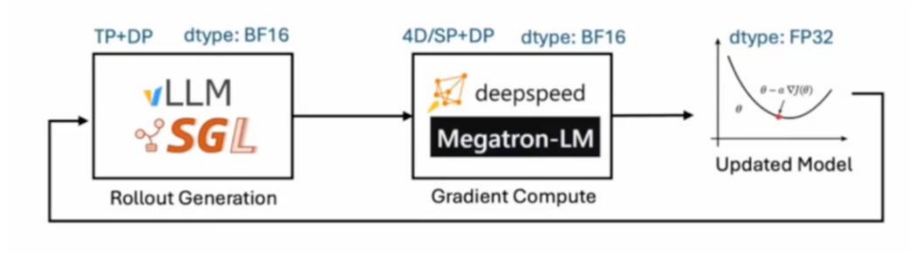
Kai Jie Lin

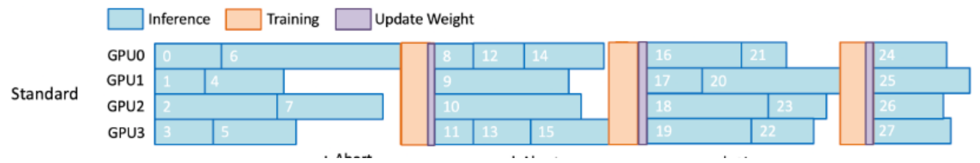
Accelerate RL for LLM

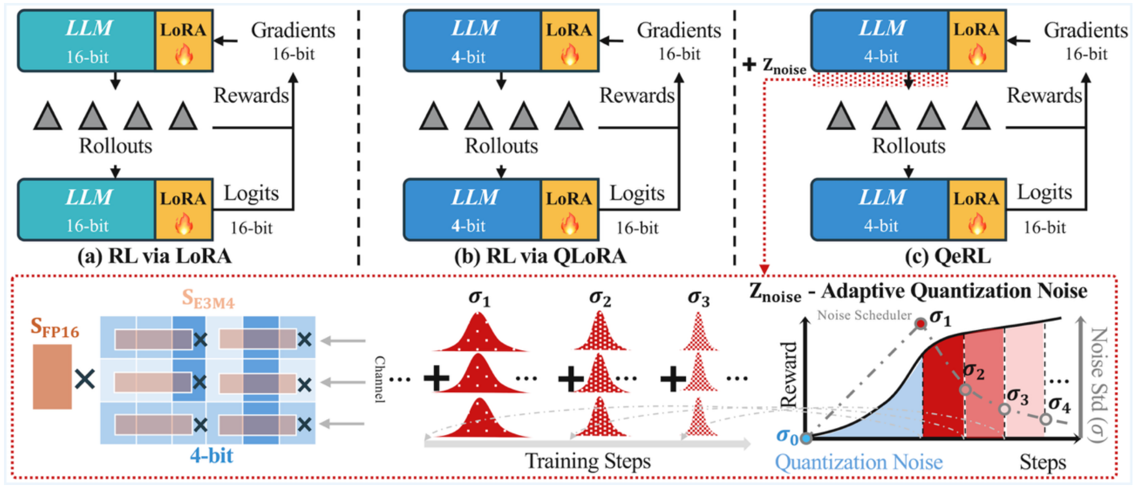
- Problem the team is trying to solve.
  - RL rollout is too slow and costly due to sampling-heavy token-by-token inference.
- Scientific driver for the chosen algorithm.
  - We selected DAPO/GRPO because they reflect the modern direction of LLM reasoning—stable, scalable, and efficient for multi-step reasoning, making them ideal for accelerated post-training.
- What's the algorithmic motif?
  - In this project we benchmark TRL's GRPO baseline against Quantization RL's GRPO / DAPO variants with LoRA / QLoRA, to measure both stability and system-level acceleration
- What parts are you focused on?
  - **Reducing inference latency** by eliminating MAGMA fallback on V100 and ensuring Tensor Core usage on H100 (FlashAttention, Marlin kernels, FP8).
  - Applying LoRA / QLoRA to reduce memory and enable faster fine-tuning; we're comparing how QeRL integrates lightweight adapters within rollout + policy-update loops.

Project Overview

- **Goal:** Speed up **Reinforcement Learning (RL) for Large Language Models (LLMs)** using efficient training and inference pipelines.
- **QeRL focuses on**
  - **Accelerating the RL for LLM post-training stage** by integrating **vLLM + LoRA** for fast rollout and policy updates.
  - **Reducing GPU memory and compute cost** through **quantization (NVFP4/FP8)** and optimized distributed execution.







https://github.com/NVlabs/QeRL?tab=readme-ov-file

Loss function: GRPO vs. DAPO

- DAPO increases the clipping upper bound (1 + ϵ_high) to address the issue of limited exploration in the early stages of the algorithm.
- It introduces a dynamic sampling mechanism that enforces diversity among samples (excluding those with acc = 0 or 1).
- DAPO modifies the loss function aggregation from a sequence-level sum to a token-level sum to prevent gradient dilution in long sequences.

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{q\sim P(Q),\, \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(O|q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(r_{i,t}(\theta)A_i,\ \text{clip}\Big(r_{i,t}(\theta),1-\varepsilon,1+\varepsilon\Big)A_i\right) - \beta\,\mathbb{D}_{KL}\left(\pi_\theta \,\|\, \pi_{\text{ref}}\right)\right)\right]$$

$$\mathcal{J}_{DAPO}(\theta) = \mathbb{E}_{(q,a)\sim P(Q),\, \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(O|q)}\left[\frac{1}{\sum_{i=1}^{G}|o_i|}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}\min\left(r_{i,t}(\theta)A_i,\ \text{clip}\Big(r_{i,t}(\theta),1-\varepsilon_{\text{low}},1+\varepsilon_{\text{high}}\Big)A_i\right)\right]$$

Data Format – NVFP4 (1/3)

- Reducing model size and compute cost is essential for deploying LLMs
- Quantization converts high-precision weights (and/or activations) into lower-precision formats, reducing memory bandwidth and improving throughput
- Recently, low-precision formats such as NF4, MXFP4, and NVFP4 have emerged, enabling efficient LLM inference with minimal accuracy loss through advanced scaling and quantization techniques
- BF16 and FP16 are commonly used in training models and saving checkpoints for the model
  - FP16: 1 sign bit, 5 exponent bits, 10 mantissa bits (E5M10)
  - BF16: 1 sign bit, 8 exponent bits, 7 mantissa bits (E8M7)

Data Format – NVFP4 (2/3)

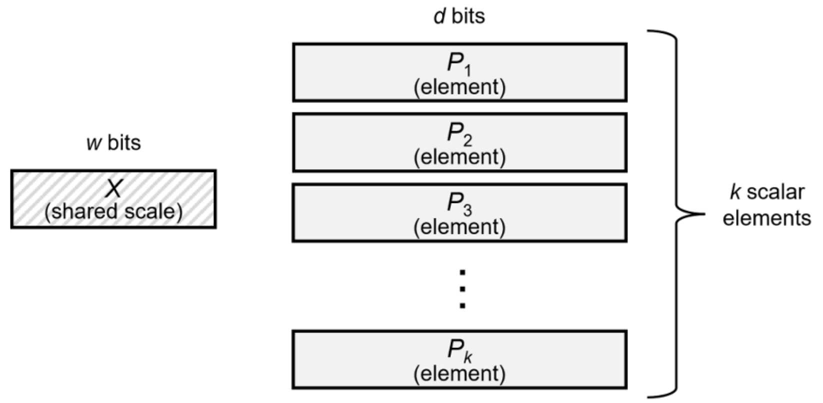
- NF4 (NormalFloat4, arXiv:2305.14314)
  - Pretrained neural network weights usually have a zero-centered normal distribution
  - Transform all weights to a single fixed distribution by scaling
  - 16 codebook levels, each bin under a normal distribution N(0,1) contains equal probability mass

- MXFP4 (MicroXcaling FP4, arXiv:2310.10537)
  - Group weight elements into small blocks,
    elements in the same block share a scaling factor
  - Element: E2M1(1-bit mantissa and a 2-bit exponent)
  - Unsigned 8-bit (E8M0) scaling factor, block size = 32

- NVFP4
  - Element: E2M1(1-bit mantissa and a 2-bit exponent)
  - FP8 (E4M3) scaling factor, block size = 16

An MX-compliant format is characterized by three components:

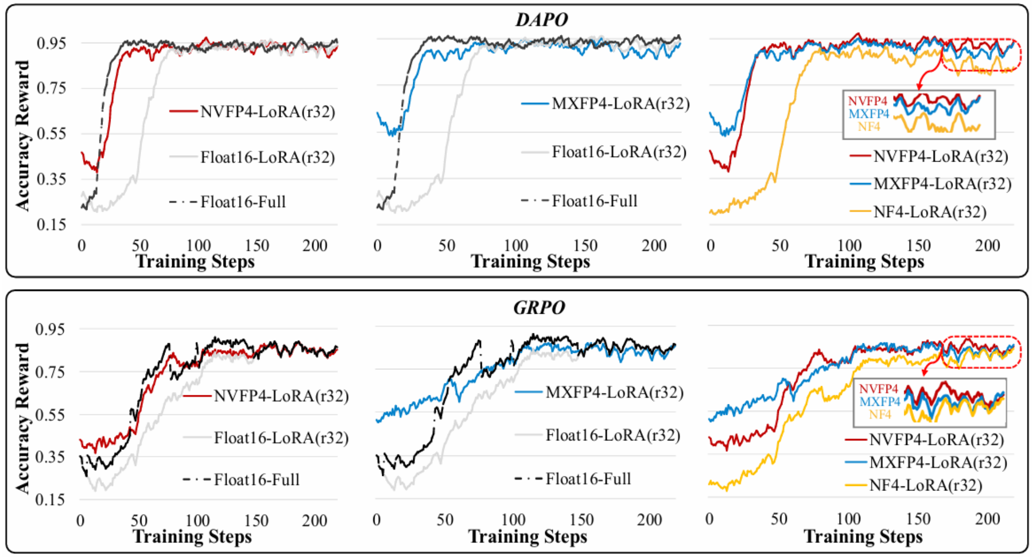
- Scale (X) data type / encoding
- Private elements (P_i) data type / encoding
- Scaling block size (k)



Data Format – NVFP4 (3/3)

- In hardware execution
  - NF4 uses a codebook table, adding latency and requiring high-precision arithmetic
  - MXFP4 uses block-wise quantization, enabling simple scaling operations and efficient hw execution
  - NVFP4 is natively supported on NVIDIA Blackwell Tensor Cores, delivering higher throughput

- Experiments (arXiv:2510.11696)
  - RLHF w/ LoRA adapter, r=32
  - W4A16 (Weight: 4bits, Activation: 16bits)
  - NVFP4 ultimately converges to better rewards
  - NVFP4 on H100 is executed by the Marlin kernel



Evolution and Strategy

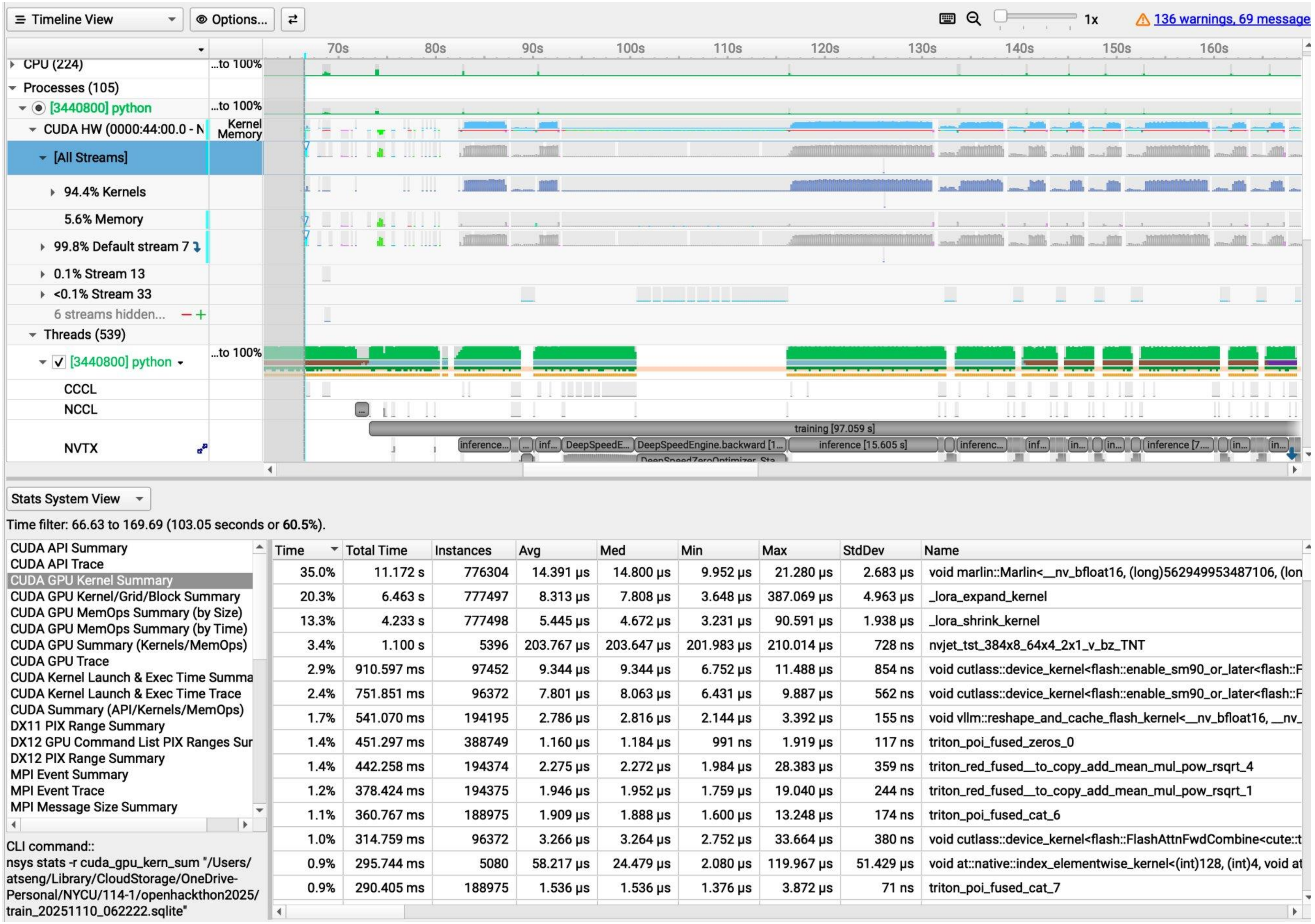
- What was your goal for coming here?
  - Our goal was to gain practical experience in accelerating RL training pipelines and to better understand how to profile long-running inference workloads in a real environment.
- What was your initial strategy?
  - We initially planned to build the entire setup from the ground up using NeMo-RL or SLIME (combining Megatron-LM for training, SGLang for inference, and Ray for distributed data exchange).
- How did this strategy change?
  - We ran into several infrastructure challenges, so we switched to a more practical approach. Using QeRL for faster iteration and single-GPU profiling.

Results and Final Profile

- What were you able to accomplish?
  - We successfully ran QeRL on a single H100 GPU using LoRA with DAPO, and compared its rollout sampling performance against the TRL GRPO baseline.

- Did you achieve a speed up?
  - 4.6× speed-up in rollout sampling compared to the original TRL GRPO implementation.

- What did you learn?
  - We realized that there is still so much more to learn in this field. This experience taught us to make the most of the limited time we have to keep learning, experimenting, and improving step by step.



Time filter: 66.63 to 169.69 (103.05 seconds or 60.5%).

CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
CUDA GPU Trace
CUDA Kernel Launch & Exec Time Summa
CUDA Kernel Launch & Exec Time Trace
CUDA Summary (API/Kernels/MemOps)
DX11 PIX Range Summary
DX12 GPU Command List PIX Ranges Sur
DX12 PIX Range Summary
MPI Event Summary
MPI Event Trace
MPI Message Size Summary

Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Name
35.0%	11.172 s	776304	14.391 μs	14.800 μs	9.952 μs	21.280 μs	2.683 μs	void marlin::Marlin<__nv_bfloat16, (long)562949953487106, (lon
20.3%	6.463 s	777497	8.313 μs	7.808 μs	3.648 μs	387.069 μs	4.963 μs	_lora_expand_kernel
13.3%	4.233 s	777498	5.445 μs	4.672 μs	3.231 μs	90.591 μs	1.938 μs	_lora_shrink_kernel
3.4%	1.100 s	5396	203.767 μs	203.647 μs	201.983 μs	210.014 μs	728 ns	nvjet_tst_384x8_64x4_2x1_v_bz_TNT
2.9%	910.597 ms	97452	9.344 μs	9.344 μs	6.752 μs	11.488 μs	854 ns	void cutlass::device_kernel<flash::enable_sm90_or_later<flash::F
2.4%	751.851 ms	96372	7.801 μs	8.063 μs	6.431 μs	9.887 μs	562 ns	void cutlass::device_kernel<flash::enable_sm90_or_later<flash::F
1.7%	541.070 ms	194195	2.786 μs	2.816 μs	2.144 μs	3.392 μs	155 ns	void vllm::reshape_and_cache_flash_kernel<__nv_bfloat16, __nv_
1.4%	451.297 ms	388749	1.160 μs	1.184 μs	991 ns	1.919 μs	117 ns	triton_poi_fused_zeros_0
1.4%	442.258 ms	194374	2.275 μs	2.272 μs	1.984 μs	28.383 μs	359 ns	triton_red_fused__to_copy_add_mean_mul_pow_rsqrt_4
1.2%	378.424 ms	194375	1.946 μs	1.952 μs	1.759 μs	19.040 μs	244 ns	triton_red_fused__to_copy_add_mean_mul_pow_rsqrt_1
1.1%	360.767 ms	188975	1.909 μs	1.888 μs	1.600 μs	13.248 μs	174 ns	triton_poi_fused_cat_6
1.0%	314.759 ms	96372	3.266 μs	3.264 μs	2.752 μs	33.664 μs	380 ns	void cutlass::device_kernel<flash::FlashAttnFwdCombine<cute::t
0.9%	295.744 ms	5080	58.217 μs	24.479 μs	2.080 μs	119.967 μs	51.429 μs	void at::native::index_elementwise_kernel<(int)128, (int)4, void at
0.9%	290.405 ms	188975	1.536 μs	1.536 μs	1.376 μs	3.872 μs	71 ns	triton_poi_fused_cat_7

CLI command::

```
nsys stats -r cuda_gpu_kern_sum "/Users/
atseng/Library/CloudStorage/OneDrive-
Personal/NYCU/114-1/openhackthon2025/
train_20251110_062222.sqlite"
```

Energy Efficiency

INPUTS	
Baseline	GPU
Baseline GPU Type	8x V100 32GB SXM
Baseline GPU # GPUs	1
Final GPU Node	8x H100 80GB SXM5
Final # GPUs	1
Application Speedup	4.6x

Node Replacement	4.6x

GPU NODE POWER SAVINGS			
	Baseline	Comparison	
	8x V100 32GB SXM	8x H100 80GB SXM5	Power Savings
Compute Power (W)	16,135	10,298	5,837
Networking Power (W)	590	880	-290
Total Power (W)	**16,725**	**11,178**	5,547

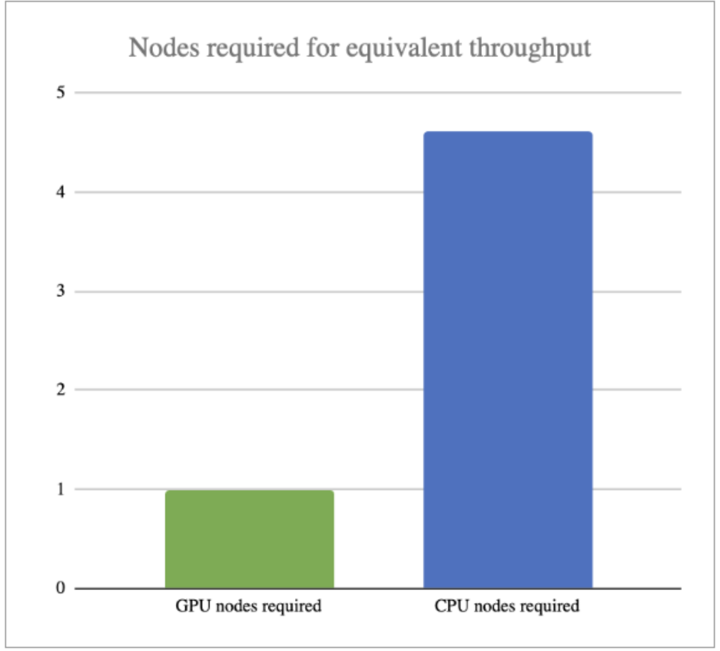
Node Power efficiency	**1.5x**

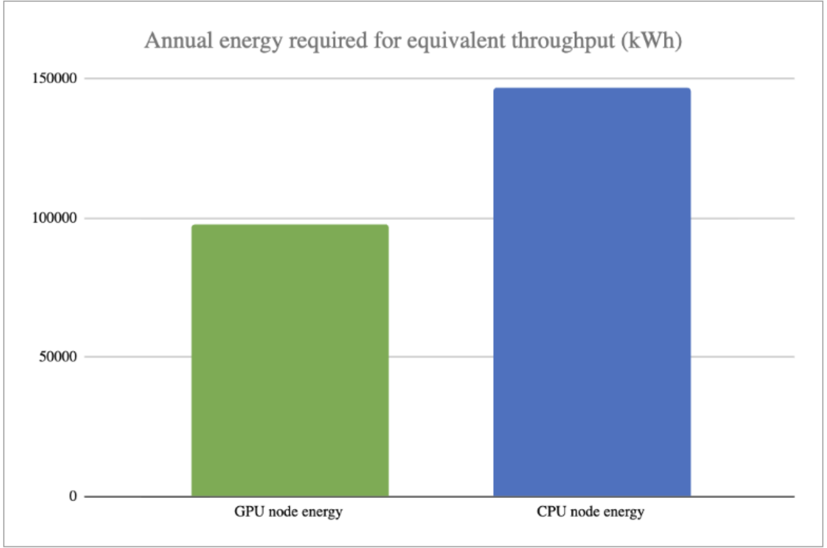
ANNUAL ENERGY SAVINGS PER GPU NODE			
	8x V100 32GB SXM	8x H100 80GB SXM5	Power Savings
Compute Power (kWh/year)	141,343	90,210	51,132
Networking Power (kWh/year)	5,169	7,709	(2,540)
Total Power (kWh/year)	**146,512**	**97,919**	**48,592**

$/kWh	$ 0.18
Annual Cost Savings	$ 8,746.64
3-year Cost Savings	$ 26,239.91

Metric Tons of CO2	34
Gasoline Cars Driven for 1 year	7
Seedlings Trees grown for 10 years	570

(source: Link)





Energy Efficiency

7

Gasoline cars driven for a year

Trees growing for 10 years

570

34

Metric tons of CO2

What problems have you encountered?

1. Build & installation issues

- **transformer_engine failed to build** — wheel compilation repeatedly failed with error: subprocess-exited-with-error and missing CUDA/CMake dependencies.
- Needed specific environment setup (CUDA ≥ 12.1, cuDNN ≥ 9.3, GCC ≥ 9).
- Inconsistent versions between **PyTorch**, **Triton**, and **TransformerEngine** caused installation failure.

2. Import & version mismatch

- ImportError: cannot import name 'entropy_from_logits' from trl.trainer.utils
  → TRL's internal API changed, QeRL trainer code was outdated.
- Frequent **dependency drift** between Hugging Face trl and accelerate versions.

3. Runtime & DeepSpeed errors

- **ChildFailedError** and **SIGSEGV (segmentation fault)** during launch via Accelerate + DeepSpeed.
- Process exited with exitcode: -11, sometimes without traceback.
- **Destroy process group not called** warnings → resource leaks on multi-GPU nodes.

Wishlist

Event

- Need More time make further progress and achieve better results.

Final Thoughts

- Was this Open Hackathon worth it?
  - Sure, I learned a lot from the mentors, especially about practical optimization techniques and profiling.
  - The discussions and feedback helped me better understand how to analyze GPU performance bottlenecks and design more efficient training and inference pipelines.

- Will you continue development?
  - Yes, because it is also part of my school project, and I want to gain deeper knowledge through further exploration.

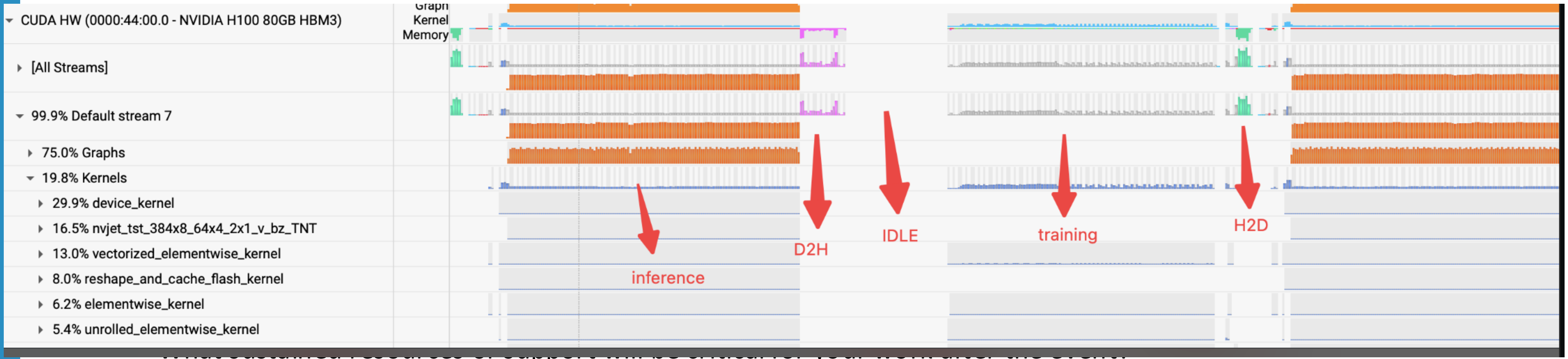
- Next steps, future plans.
  - Keep working on addressing the feedback and issues pointed out by the mentors.
  - Integrate **Transformer Engine** into both the training and inference pipelines to explore **FP8 acceleration** and further improve overall performance.

- What sustained resources or support will be critical for your work after the event?
  - Continued access to **NVIDIA Blackwell or Hopper architecture's GPU**, would be helpful for future experiments. These resources will allow us to further test FP8 optimization and multi-GPU scaling.

Final Thoughts

- Was this Open Hackathon worth it?
  - Sure, I learned a lot from the mentors, especially about practical optimization techniques and profiling.
  - The discussions and feedback helped me better understand how to analyze GPU performance bottlenecks and design more efficient training and inference pipelines.

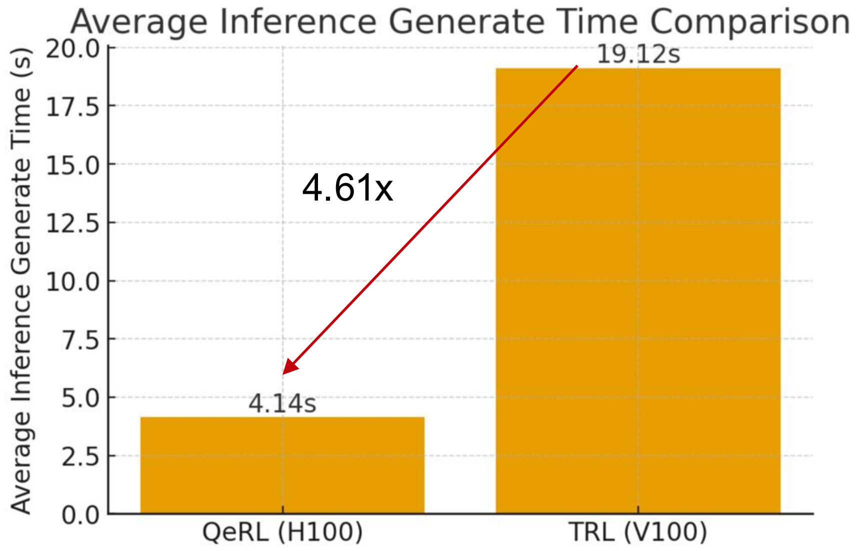
Solve following issues.


CUDA HW (0000:44:00.0 - NVIDIA H100 80GB HBM3)
Graph
Kernel
Memory
[All Streams]
99.9% Default stream 7
75.0% Graphs
19.8% Kernels
29.9% device_kernel
16.5% nvjet_tst_384x8_64x4_2x1_v_bz_TNT
13.0% vectorized_elementwise_kernel
8.0% reshape_and_cache_flash_kernel
6.2% elementwise_kernel
5.4% unrolled_elementwise_kernel
inference
D2H
IDLE
training
H2D

- What sustained resources or support will be critical for your work after the event?
  - Continued access to **NVIDIA Blackwell or Hopper architecture's GPU**, would be helpful for future experiments. These resources will allow us to further test FP8 optimization and multi-GPU scaling.

Application Background

This project focuses on optimizing large language model (LLM) training and inference efficiency. By analyzing GPU performance and reducing data transfer overhead, it aims to make model fine-tuning faster and more cost-effective. In the future, this system can help anyone fine-tune their own smaller models more efficiently using techniques like FP8 quantization, CUDA Graphs, and multi-GPU parallelism.



Hackathon Objectives and Approach

Programming models. : CUDA/torch
Profiling/hot spots : Nsight Systems
Libraries : transformer_engine, TRL, llmcompressor,open-rl,Marlin.

Technical Accomplishments and Impact

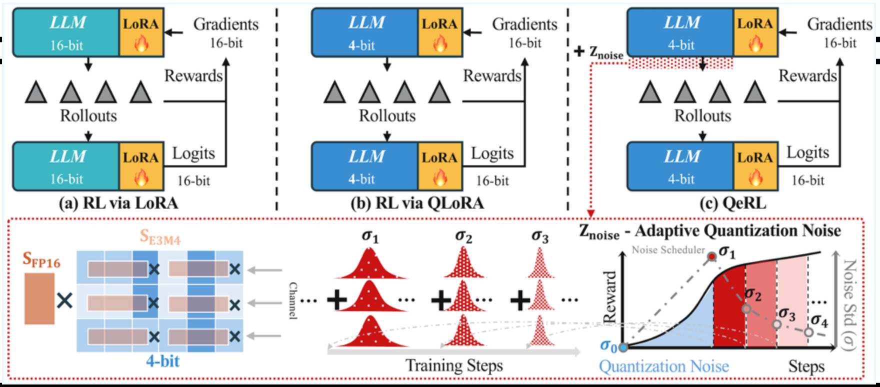
What were you able to achieve at the hackathon?
• Successfully ran **QeRL (GRPO/DAPO)** with **LoRA** and on a single H100 GPU.

• Profiled and analyzed rollout performance, identifying major inference bottlenecks in the RLHF pipeline.

• Compared **TRL's GRPO baseline** with **QeRL's optimized setup**, validating measurable performance gains.

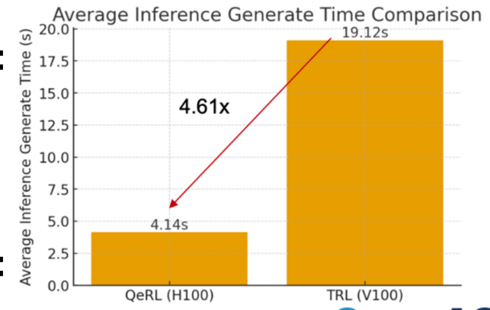
(Required) Create a storyline for publication on NCHC's website.

Accelerate RL for LLM



{_陽明交大_} 團隊來自 將 {_RL for LLM_} 加速了 {_4.6_} 倍！！

在現今人工智慧快速發展的時代，模型的規模越來越大、運算量也成倍增加。訓練或推論一個大型語言模型（LLM）往往需要耗費數天甚至數週的時間，因此「加速 AI 模型運算」成為一個極其重要的研究方向。這不只是讓程式跑得更快，而是直接影響**開發效率、能源消耗與成本效益**。例如，若能讓訓練速度提升兩倍，就能在同樣的時間內完成更多實驗，或用相同的硬體資源達成更好的成果。
本次題目聚焦在 GPU 加速與模型效能最佳化，特別是透過低精度運算（如 FP8、FP4）與效能分析工具（profiling）來找出瓶頸並進行優化。這個主題的重要性在於：隨著模型規模持續增長，傳統高精度運算將浪費大量記憶體與頻寬，而低精度計算能在保持準確度的前提下，**大幅降低成本與延遲**。
透過這次的加速成果，我們發現使用 FP8 訓練與 FP4 推論不僅能節省記憶體與能源，也能讓模型在相同硬體上達到更高吞吐量（throughput）。這樣的技術不僅對研究社群有幫助，在產業應用上更具潛力——當未來各行各業擁有自己的專屬資料集時，若能結合加速與低精度技術，就能**更快速地完成線上（online）或離線（offline）微調（finetune）**，讓企業能以更低成本、更高效率地部署並更新自家的 AI 模型，實現真正靈活且持續演進的智慧系統。



報告投影片連結 (由國網上傳到 github)