LLM RL Training-Rollout Mismatch

Xinyi Wang
1/13/2026

Your Efficient RL Framework Secretly Brings You Off-Policy RL Training

Feng Yao* Liyuan Liu* Dinghuai Zhang Chengyu Dong Jingbo Shang Jianfeng Gao

*: Equal Contributions (Work in Progress)

Last Updated on October 13, 2025 | First Published on August 5, 2025 | [: Github]

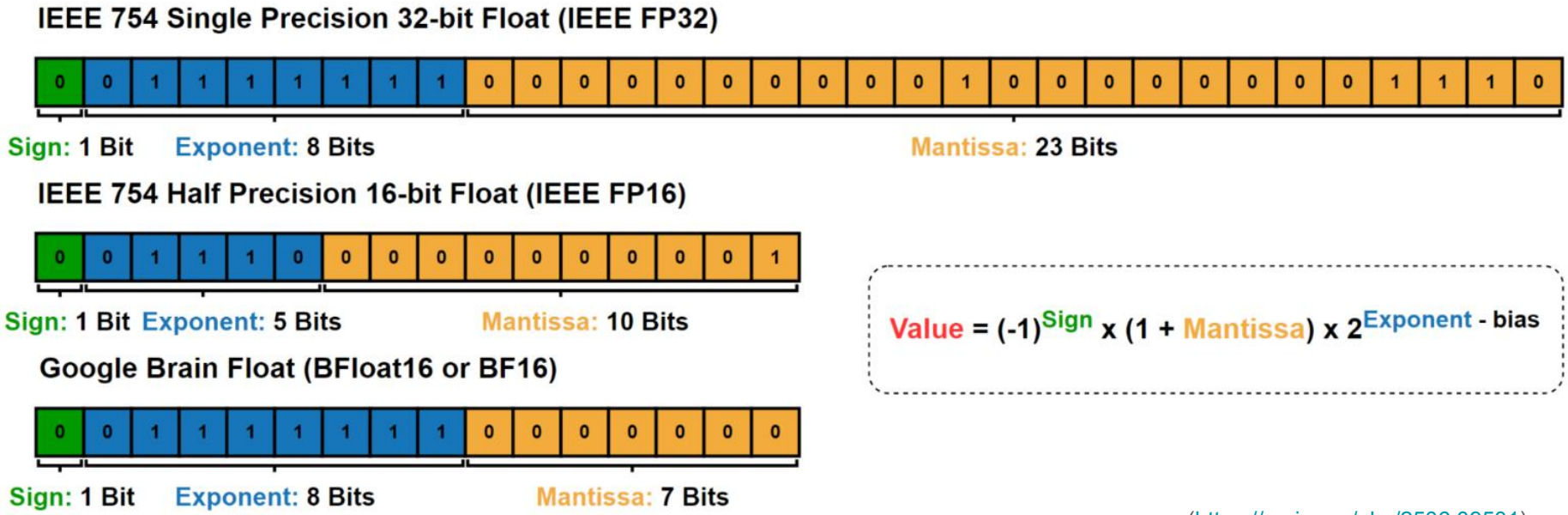
TL;DR

In modern RL training frameworks (e.g., VeRL), different implementations are used for rollout generation (e.g., vLLM) and model training (e.g., FSDP). Here, we show the **implementation gap** implicitly turns the on-policy RL to be **off-policy**, and discuss a simple yet effective importance sampling technique for handling such discrepancy.

(https://fengyao.notion.site/off-policy-rl)

Root cause of training-rollout mismatch

- Floating point (especially BF16) numbers have limited precision



(https://arxiv.org/abs/2506.09501)

Root cause of training-rollout mismatch

- Because of rounding errors, floating-point arithmetic is not associative.
- i.e. (a + b) +c != a + (b + c)

Example	Sum Order	FP32	BF16
$a, b, c = 0.1, -0.1, 0.2$	$a+b+c$	00111110010011001100110011001101	0011111001001101
	$a+c+b$	00111110010011001100110011001110	0011111001001110

But what happens when we add two floating-point numbers with different exponents, such as 1230 and 23.4? In this case, the exact result is 1253.4. However, we can only maintain 3 digits of precision at a time. Floating-point addition will thus *drop* the last 2 digits and obtain the value $1.25 * 10^3$ (or 1250).

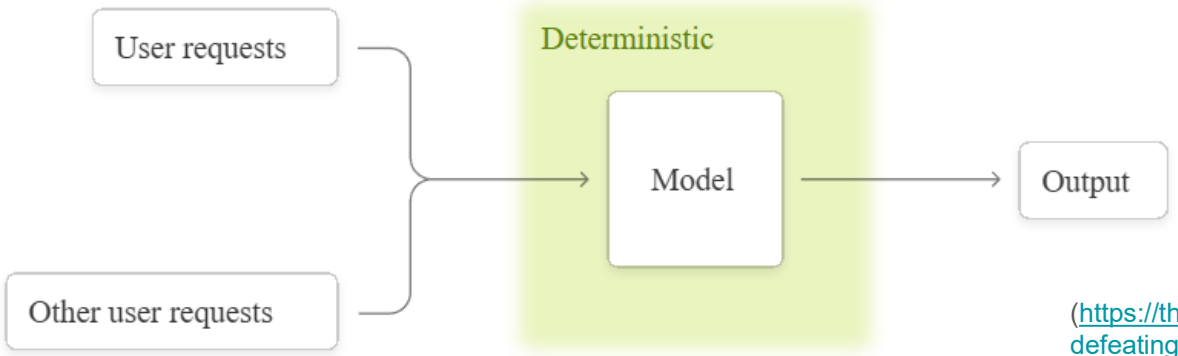
(https://arxiv.org/abs/2506.09501
https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

1230 + 23.4 = 1.2534 × 10³

1.23 × 10³ 2.34 × 10¹ exact: 1253.4

Nondeterminism in LLM Inference

- For an LLM forward pass, if the hardware, input batch, and inference algorithm is completely fixed, the output is usually **deterministic**.
  - Reason: concurrent atomic adds are not necessary for the vast majority of kernels



(https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

Figure 3: From the perspective of the inference server, it *is* deterministic. Given the exact same user requests, it will always provide the same deterministic output.

Nondeterminism in LLM Inference

- Within one (same) GPU:
  - Nondeterminism caused by different **batch size**, as matmul implementations are not batch-invariant.
  - i.e. the same input placed in different batches might produce different outputs

```python
import torch
torch.set_default_device('cuda')

B = 2048
D = 4096
a = torch.linspace(-1000, 1000, B*D).reshape(B, D)
b = torch.linspace(-1000, 1000, D*D).reshape(D, D)
# Doing a matrix vector multiplication by taking
# the first element of the batch
out1 = torch.mm(a[:1], b)
# Doing a matrix matrix multiplication and then taking
# the first element of the batch
out2 = torch.mm(a, b)[:1]
print((out1 - out2).abs().max()) # tensor(1669.2500, device='cuda:0')
```

(https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

Nondeterminism in LLM Inference

BIO: batch invariant operation

- Across several GPUs:
  - Nondeterminism caused by different **tensor parallelism (TP) sizes**, due to different reduction order (communication protocol) and varied batch size per GPU.
  - i.e. the same batch of inputs split across different number of GPUs will produce different outputs

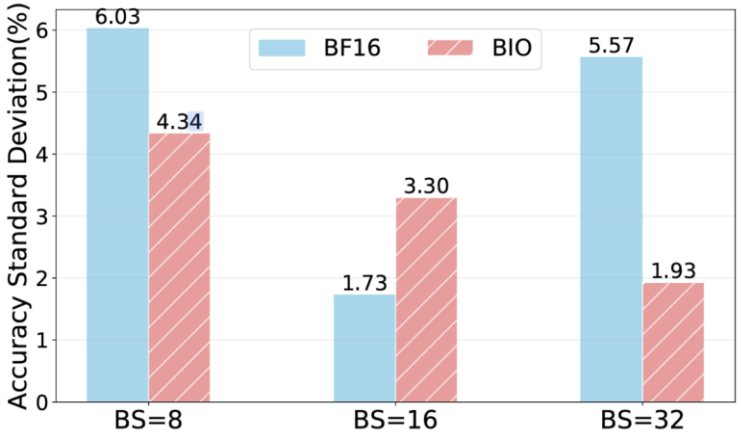


Figure 4. Accuracy standard deviation of Qwen3-8B on AIME24 dataset under different tensor parallelism (TP = 1/2/4/8) settings. While BIO enhances output determinism relative to vanilla BF16 inference, the accuracy variance can still reach over 4%.

(https://arxiv.org/abs/2511.17826)

Nondeterminism in LLM Inference

(https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Training-Inference-Mismatch-271211a558b7808d8b12d403fd15edda)

- Different GPU types will also produce different outputs because of kernel implementation differences.
  - e.g. H200 is more stable for RL training

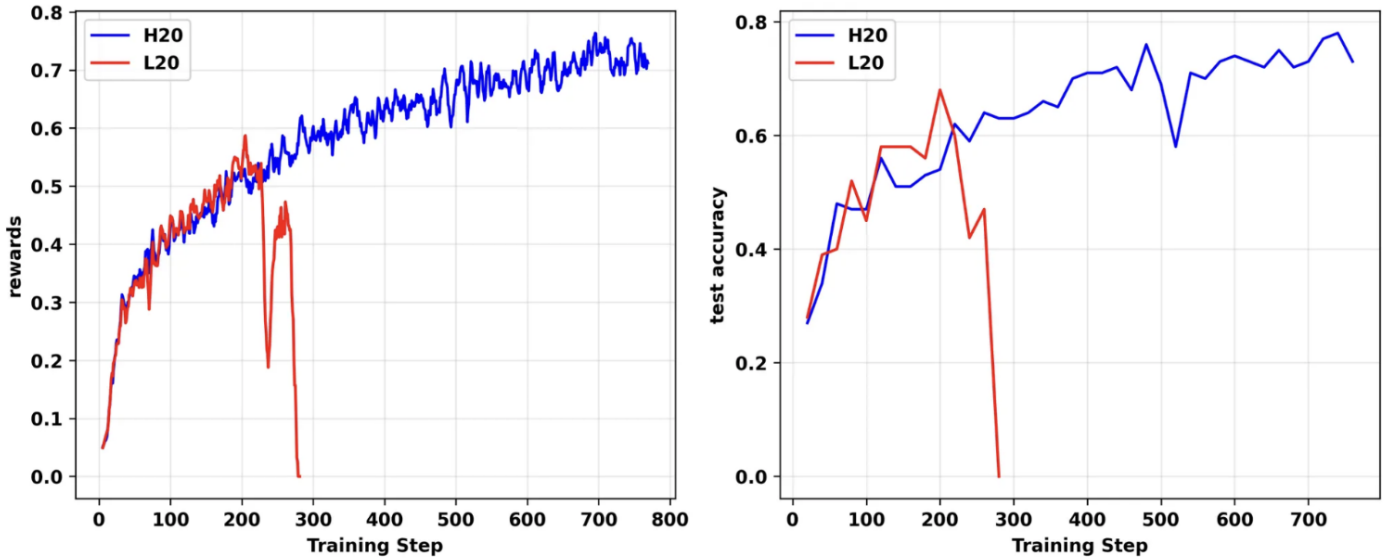
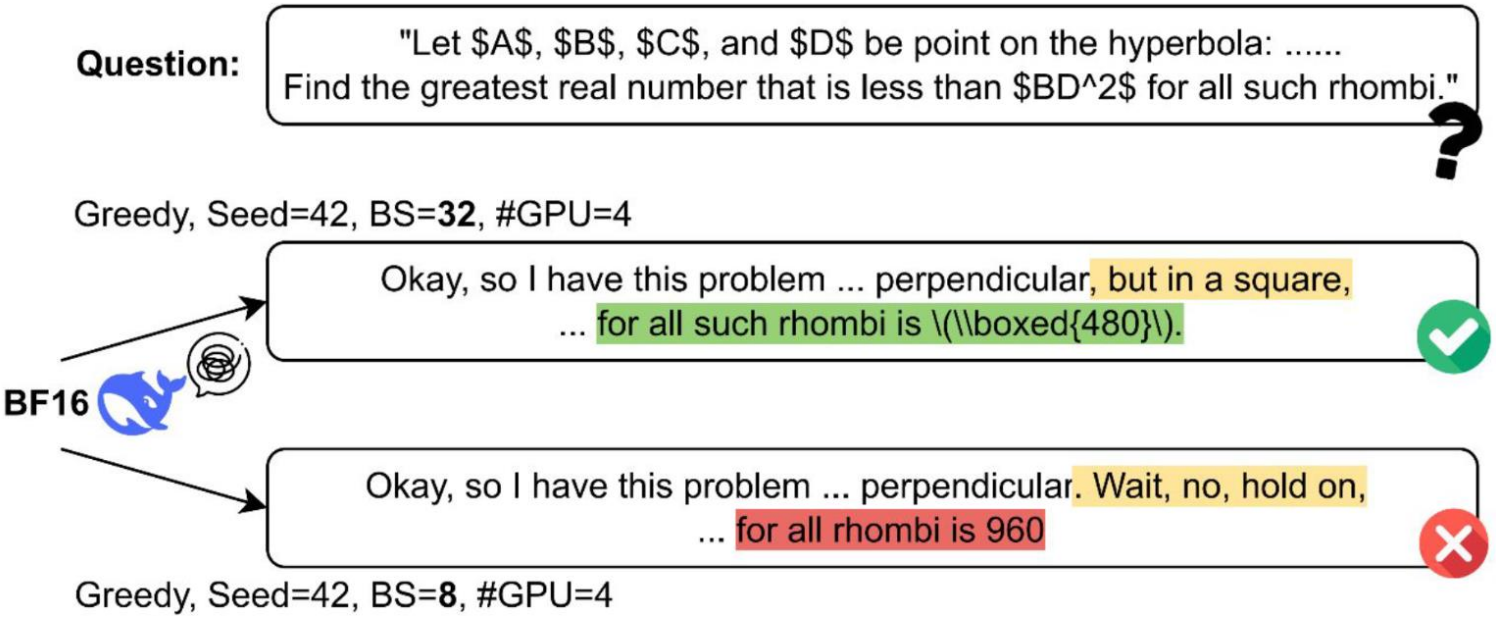


Figure 20: *Training rewards (**Left**) and test accuracy (**Right**) of two on-policy GRPO experiments conducted on L20 and H20 GPUs, respectively.*

Example of nondeterministic greedy decoding

- Small errors accumulate across long generations …



(https://arxiv.org/abs/2506.09501)

Accuracy variance

- When evaluate LLMs, using FP32 can greatly reduce numerical instability.
- However, for RL rollouts, we need to use 16 bits for training efficiency, which would produce large variance.

Table 3: Std@Acc of greedy decoding across 12 different settings (GPU types, GPU counts, and batch sizes) under BF16, FP16, and FP32 Numerical Precisions. Reasoning models also exhibit larger variance compared to non-reasoning counterparts. More results can be found in Appendix C.

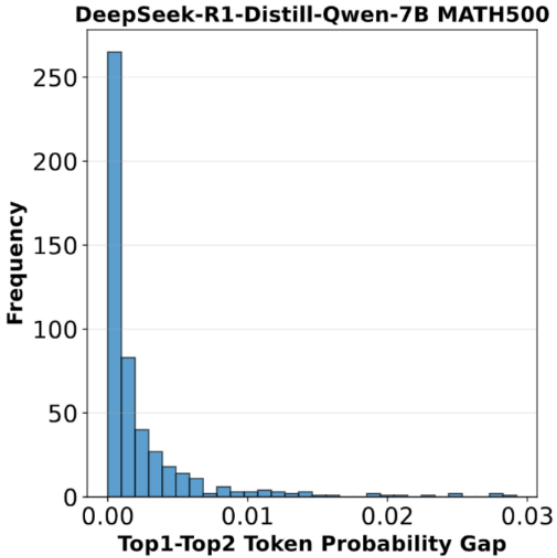
	AIME'24			MATH500			LiveCodeBench-Easy		
	BF16	FP16	FP32	BF16	FP16	FP32	BF16	FP16	FP32
DeepSeek-R1-Distill-Qwen-7B	9.15%	5.74%	0	1.04%	1.12%	0.12%	1.67%	1.28%	0.37%
DeepSeek-R1-Distill-Llama-8B	4.60%	6.00%	5.8e-17	1.59%	0.73%	0.23%	2.31%	1.92%	0.29%
Qwen2.5-7B-Instruct	1.71%	1.45e-17	1.45e-17	0.83%	0.36%	1.16e-16	0.79%	0.48%	1.16e-16
Llama-3.1-8B-Instruct	1.92%	1.30%	0	0.94%	0.34%	0.13%	1.00%	0.67%	0.25%

(https://arxiv.org/abs/2506.09501)

Why reasoning is more sensitive to numerical instability?

- Token probabilities are similar, small rounding errors change the rank.
- Long CoT

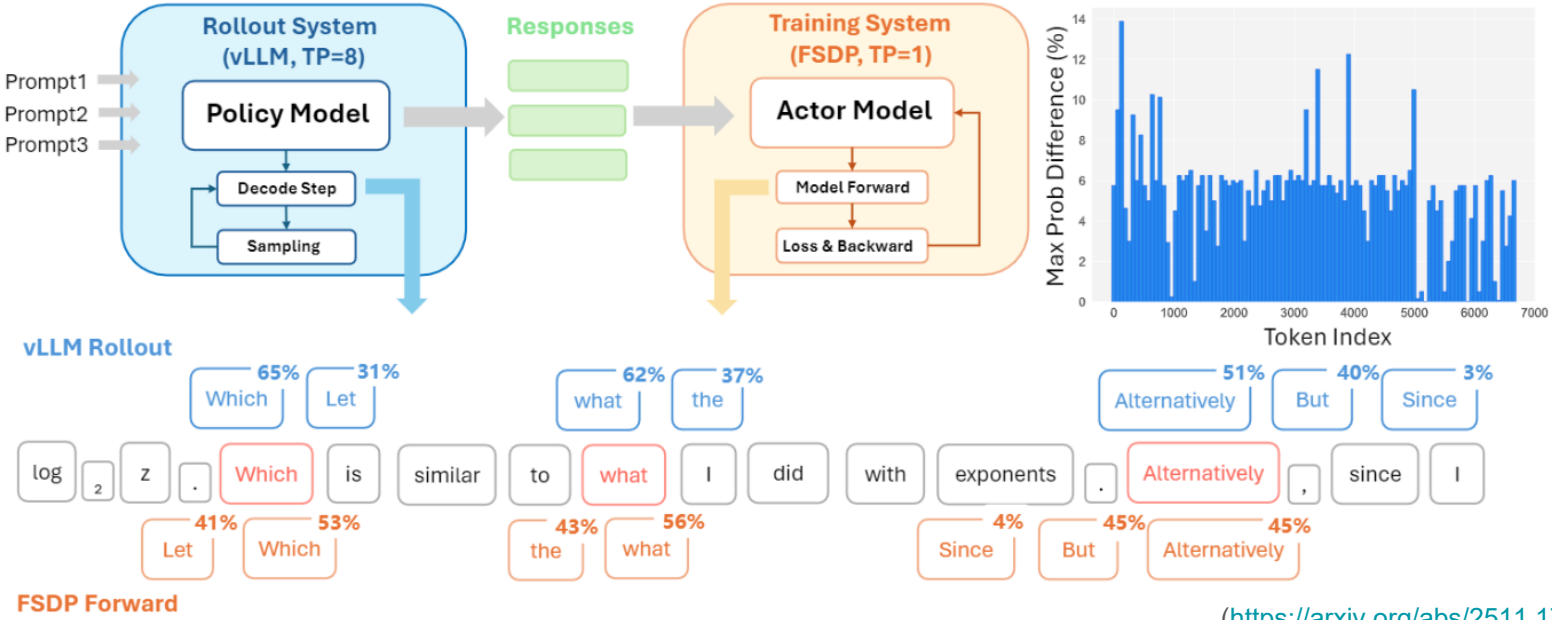
Answer 1		Answer 2	
Token	Prob.	Token	Prob.
know	49.75%	have	46.65%
have	43.91%	know	46.64%
need	3.18%	need	3.39%
'm	2.47%	'm	2.63%
've	0.49%	've	0.52%
...	...	...	...



(https://arxiv.org/abs/2506.09501)

Figure 3: **Left:** the top-5 tokens and their predicted probabilities at the divergence index for two different answers to the same question in BF16. **Right:** The gap between the top-two competing tokens probability. We observe the token probability gap are often minimal in reasoning models.

Training-rollout mismatch in RL



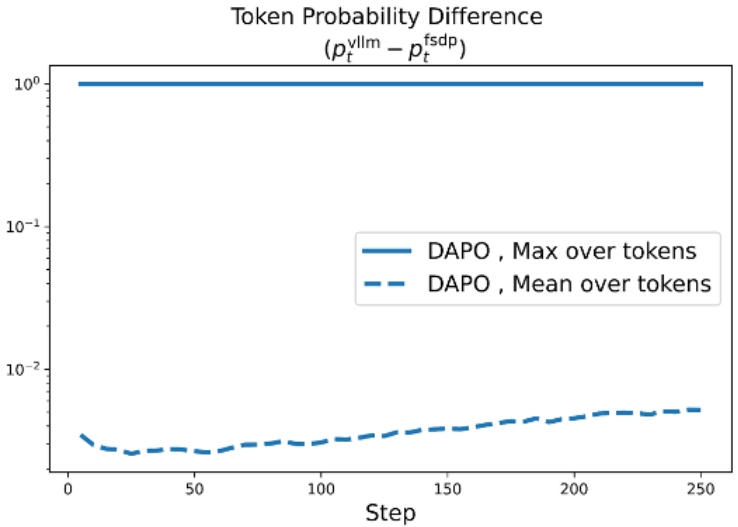
(https://arxiv.org/abs/2511.17826)

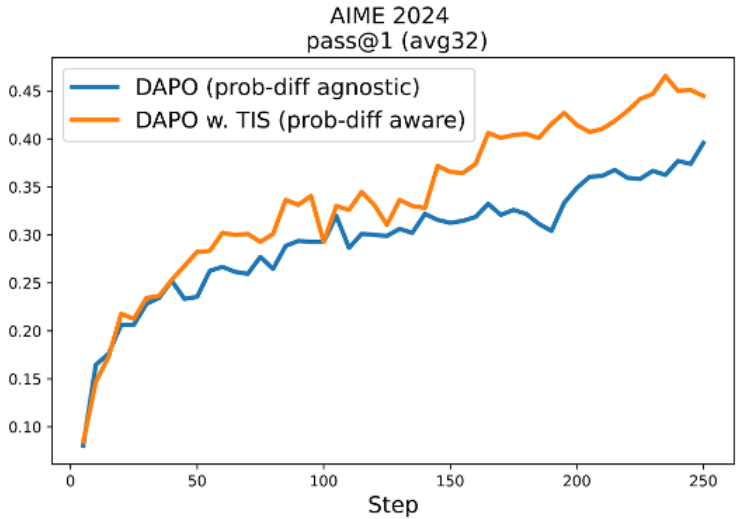
Figure 1. The illustration of numerical variance in LLM-based RL training. Different frameworks and tensor-parallel settings lead to noticeable probability discrepancies for the same model, making it difficult to achieve stable and truly on-policy reinforcement learning.

Off policy RL due to training-rollout mismatch

- Large gap between vLLM and FSDP probabilities breaks the on-policy assumption of REINFORCE-based RL algorithms (e.g. GRPO, DAPO), and hurts their performance.

$$\theta \leftarrow \theta + \mu \cdot \mathbb{E}_{a \sim \pi_{\text{vllm}}(\theta)}[R(a) \cdot \nabla_\theta \log \pi_{\text{fsdp}}(a, \theta)]$$





(https://fengyao.notion.site/off-policy-rl)

Off policy RL induce training instability

(https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Training-Inference-Mismatch-271211a558b7808d8b12d403fd15edda)

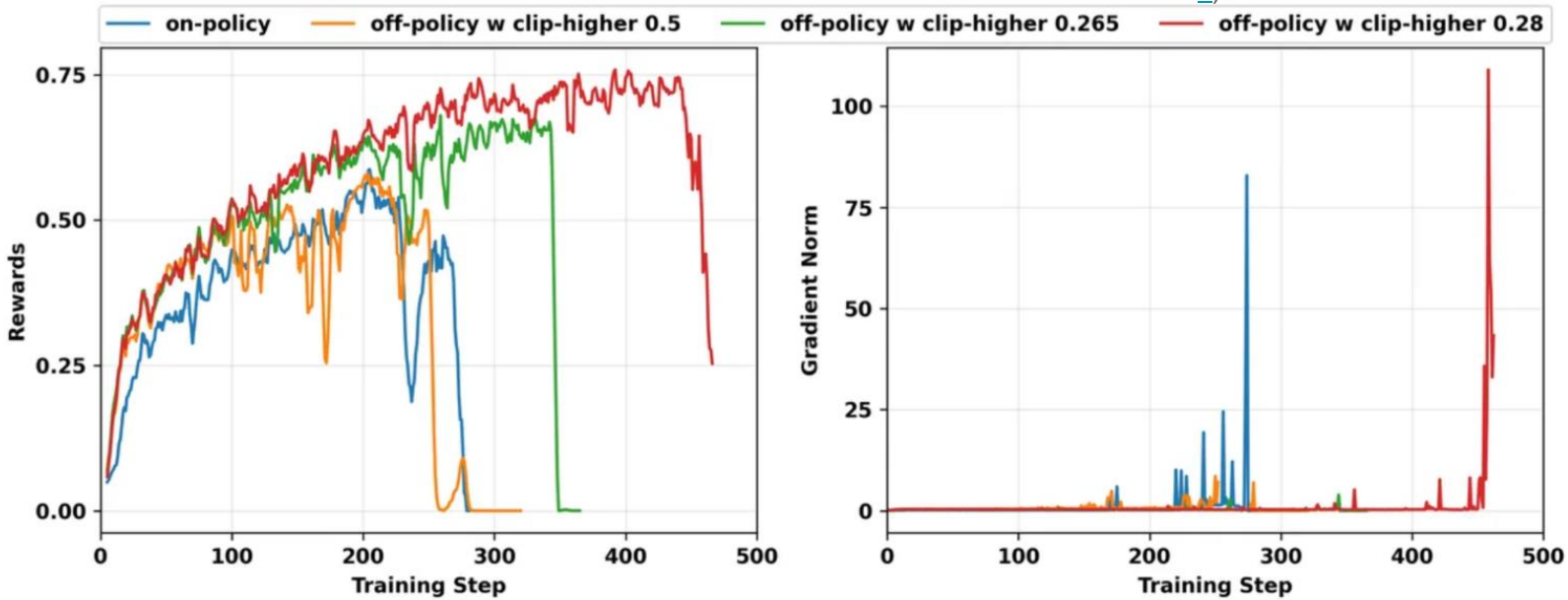


Figure 1. *Rewards (**Left**) and gradient norms (**Right**) from our four failed GRPO TIR experiments on Qwen3-14B-Base. All experiments sample* `1024` *trajectories (* `64` *prompts ×* `16` *responses) at each training step and use a learning rate of* `1e-6` *. The* `ppo_mini_batch_size` *is set to* `1024` *and* `256` *for on-policy and off-policy experiments, respectively.*

Warning Signs of training-rollout mismatch

(https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Training-Inference-Mismatch-271211a558b7808d8b12d403fd15edda)

- Anomalous fluctuations in the entropy of FSDP policy and rewards.

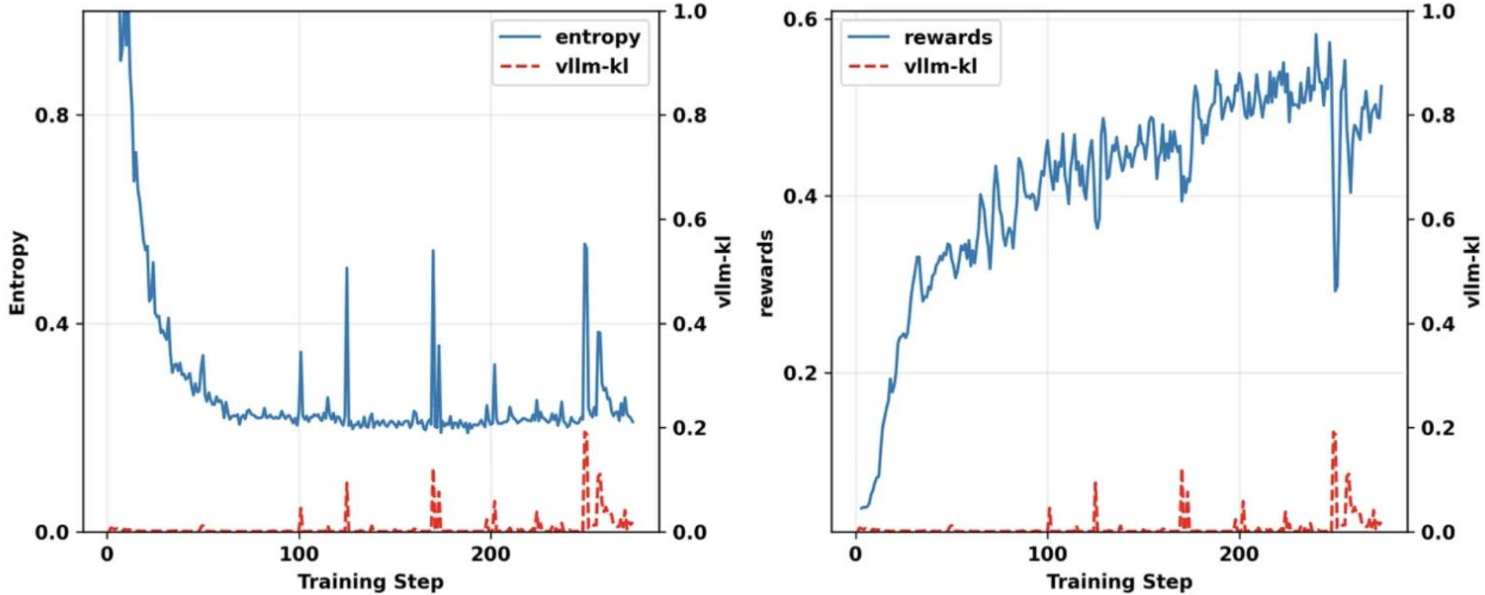


*Figure 2. Comparative presentation of an on-policy experiment result showing **entropy** (**Left**) and **rewards** (**Right**) alongside `vllm-kl` values, illustrating their correlation during the training phase.*

Warning Signs of training-rollout mismatch

(https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Training-Inference-Mismatch-271211a558b7808d8b12d403fd15edda)

- Simultaneous explosion in both the fsdp-ppl metric and the gradient norm

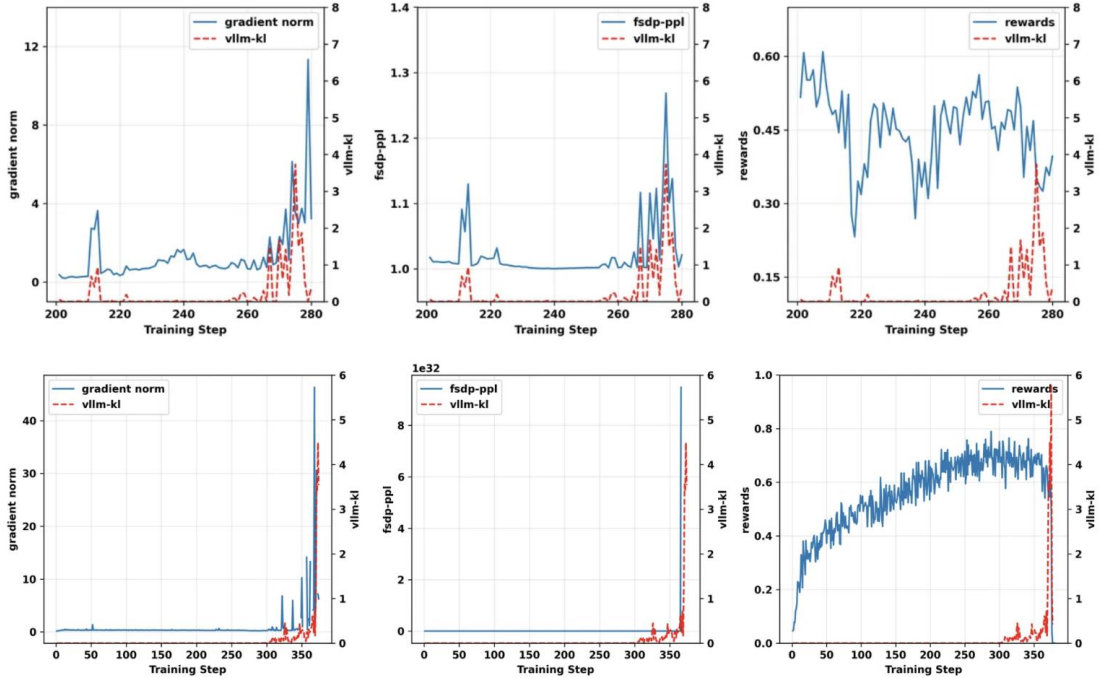


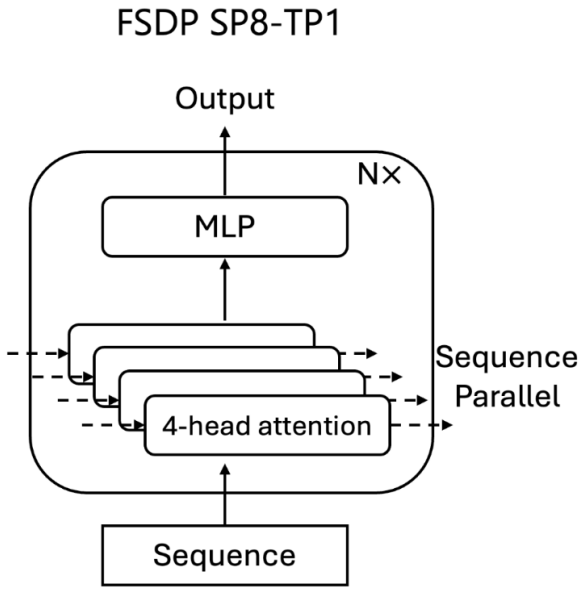
*Figure 3. Results of an on-policy experiment from step 200 - 280 (**Top**) and an off-policy experiment with clip-higher 0.28 (**Bottom**), showing **gradient norm** (**Left**), **fsdp-ppl** (**Middle**) and **rewards** (**Right**) alongside `vllm-kl` values, illustrating their correlation during the training.*

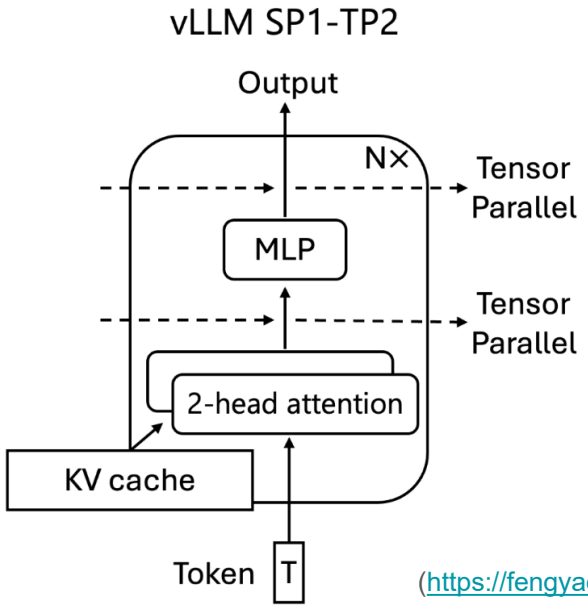
Causes of training-rollout mismatch in RL

- Compute mismatch between training and inference (matrices dimension mismatch)
  - Training: prefill
  - Rollout: prefill + generate + KV cache
- Different parallel strategy (TP mismatch)
  - Training: Fully sharded data parallel (FSDP), layer-wise shard
  - Rollout: tensor parallel (TP), matrix-wise shard
- Framework implementation difference (minor)
  - vLLM v1 engine does not support directly returning the adjusted probabilities used for sampling (fixed)
  - vLLM lm_head's precision does not match that of HuggingFace transformers

Training-rollout compute & parallelism difference

- Training
  - Prefill
  - Data parallelism + layer-wise shard
- Rollout
  - Prefill + KV cache + single token generation
  - Tensor parallelism (matrix-wise shard)





(https://fengyao.notion.site/off-policy-rl)

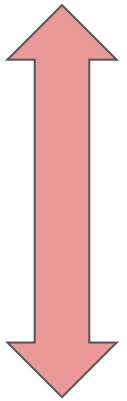
Fixing training-rollout mismatch

- Numerically stable RL algorithm (for MoE)
  - GSPO, SAPO: sequence level importance ratio
  - Router Replay: cache the activated experts
  - Issue: expert-activation volatility of MoE models
- Correct importance sampling ratio
  - Truncated Importance Sampling (TIS)
  - FlashRL: 8Bit Rollouts (vLLM patch)
  - Masked Importance Sampling (MIS)
- Higher floating point precision
  - Use FP16 instead of BF16
  - LayerCast (vLLM patch): optimized FP32 inference pipeline
- Deterministic kernels
  - Batch-invariant kernel
  - TP-invariant kernel: Tree-Based Invariant Kernels (TBIK)

Fixing training-rollout mismatch

- Numerically stable RL algorithm (for MoE)
  - GSPO, SAPO: sequence level importance ratio
  - Router Replay: cache the activated experts
  - Issue: expert-activation volatility of MoE models
- Correct importance sampling ratio
  - Truncated Importance Sampling (TIS)
  - FlashRL: 8Bit Rollouts (vLLM patch)
  - Masked Importance Sampling (MIS)
- Higher floating point precision
  - Use FP16 instead of BF16
  - LayerCast (vLLM patch): optimized FP32 inference pipeline
- Deterministic kernels
  - Batch-invariant kernel
  - TP-invariant kernel: Tree-Based Invariant Kernels (TBIK)

- Lightweight
- Cannot completely close the gap

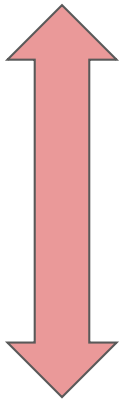


- Large overhead
- Completely close the gap

Fixing training-rollout mismatch

- Numerically stable RL algorithm (for MoE)
  - GSPO, SAPO: sequence level importance ratio
  - Router Replay: cache the activated experts
  - Issue: expert-activation volatility of MoE models
- Correct importance sampling ratio
  - Truncated Importance Sampling (TIS)
  - FlashRL: 8Bit Rollouts (vLLM patch)
  - Masked Importance Sampling (MIS)
- Higher floating point precision
  - Use FP16 instead of BF16
  - LayerCast (vLLM patch): optimized FP32 inference pipeline
- Deterministic kernels
  - Batch-invariant kernel
  - TP-invariant kernel: Tree-Based Invariant Kernels (TBIK)

- Lightweight
- Cannot completely close the gap



- Large overhead
- Completely close the gap

Correct importance sampling ratio

- Truncated Importance Sampling (TIS):

Have: $$\mathbb{E}_{a\sim\pi_{\text{vllm}}(\theta)}[R(a)\cdot\nabla_\theta\log\pi_{\text{fsdp}}(a,\theta)]$$

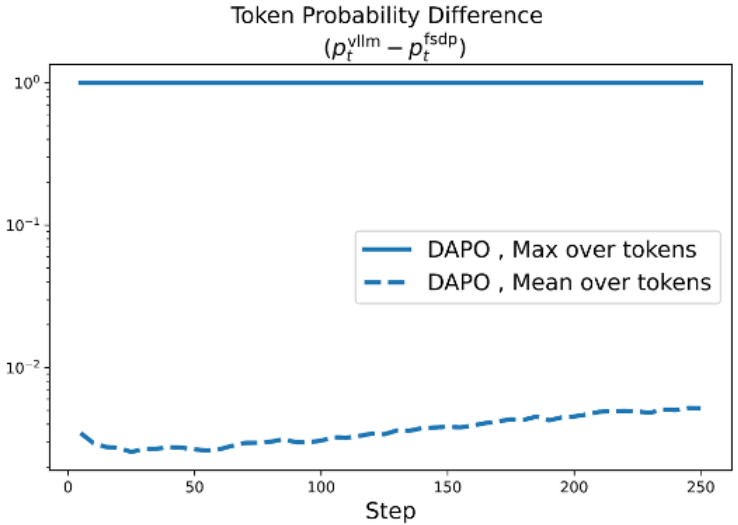
Want: $$\mathbb{E}_{a\sim\pi_{\text{fsdp}}(\theta)}[R(a)\cdot\nabla_\theta\log\pi_{\text{fsdp}}(a,\theta)]$$

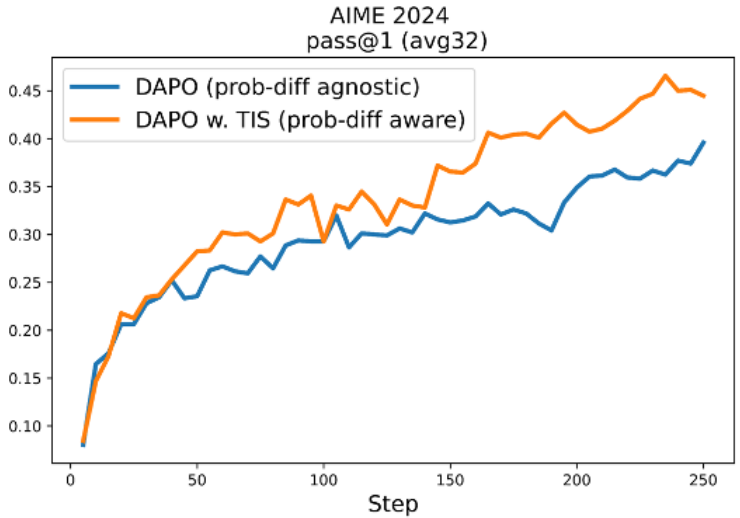
Approximate: $$\mathbb{E}_{a\sim\pi_{\text{vllm}}(\theta)}\Big[\underbrace{\min\Big(\frac{\pi_{\text{fsdp}}(a,\theta)}{\pi_{\text{vllm}}(a,\theta)},C\Big)}_{\text{truncated importance ratio}}\cdot R(a)\cdot\nabla_\theta\log\pi_{\text{fsdp}}(a,\theta)\Big]$$

(https://fengyao.notion.site/off-policy-rl)

Truncated Importance Sampling (TIS)

- Improve performance with negligible overhead
- Does not completely close the distribution gap





(https://fengyao.notion.site/off-policy-rl)

Speed up with 8bit rollouts

- Vanilla 8bit quantization hurts performance

$$\underbrace{\mathbb{E}_{a\sim\pi_{\text{bf16}}(\theta_{\text{old}})}}_{\text{int8 Rollout: } \pi_{\text{bf16}} \to \pi_{\text{int8}}} \left[\nabla_\theta \min\left(\frac{\pi_{\text{bf16}}(a,\theta)}{\pi_{\text{bf16}}(a,\theta_{\text{old}})}\hat{A},\ \text{clip}\left(\frac{\pi_{\text{bf16}}(a,\theta)}{\pi_{\text{bf16}}(a,\theta_{\text{old}})},\ 1-\epsilon,\ 1+\epsilon\right)\hat{A}\right)\right]$$

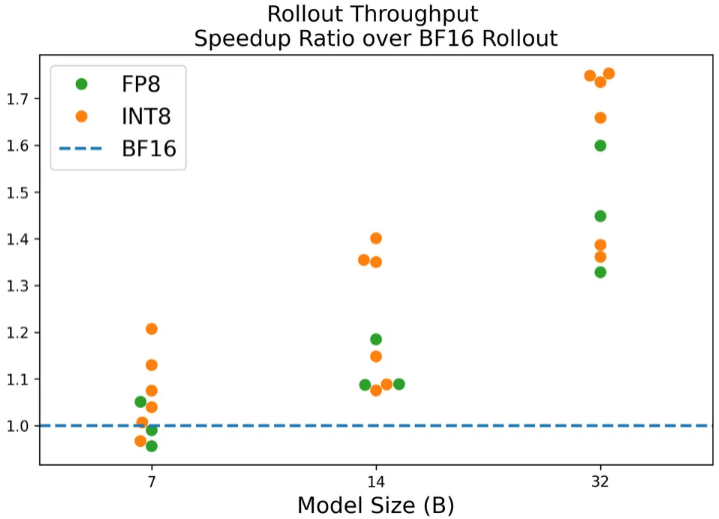
- FlashRL: use TIS

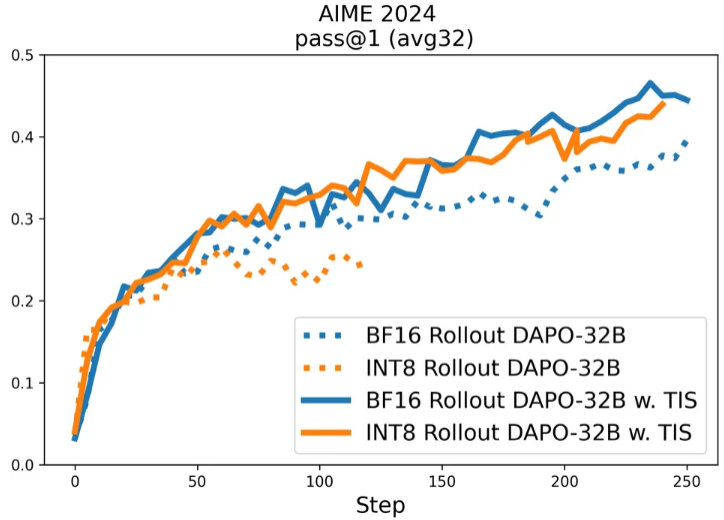
$$\mathbb{E}_{a\sim\pi_{\text{int8}}(\theta_{\text{old}})}\left[\underbrace{\min\left(\frac{\pi_{\text{bf16}}(a,\theta_{\text{old}})}{\pi_{\text{int8}}(a,\theta_{\text{old}})}, C\right)}_{\text{truncated importance ratio}} \cdot \nabla_\theta \min\left(\frac{\pi_{\text{bf16}}(a,\theta)}{\pi_{\text{bf16}}(a,\theta_{\text{old}})}\hat{A}, \text{clip}\left(\frac{\pi_{\text{bf16}}(a,\theta)}{\pi_{\text{bf16}}(a,\theta_{\text{old}})}, 1-\epsilon,\ 1+\epsilon\right)\hat{A}\right)\right]$$

(https://fengyao.notion.site/flash-rl)

Speed up with 8bit rollouts

- With DAPO and Qwen2.5-32B model, FlashRL INT8 rollout
  - matches the performance of BF16 rollout with TIS
  - Outperforms naive BF16 rollout (without TIS)





(https://fengyao.notion.site/flash-rl)

Masked Importance Sampling (MIS)

(https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Training-Inference-Mismatch-271211a558b7808d8b12d403fd15edda)

- mask the policy loss for sequences where the IS ratio exceeds the threshold C

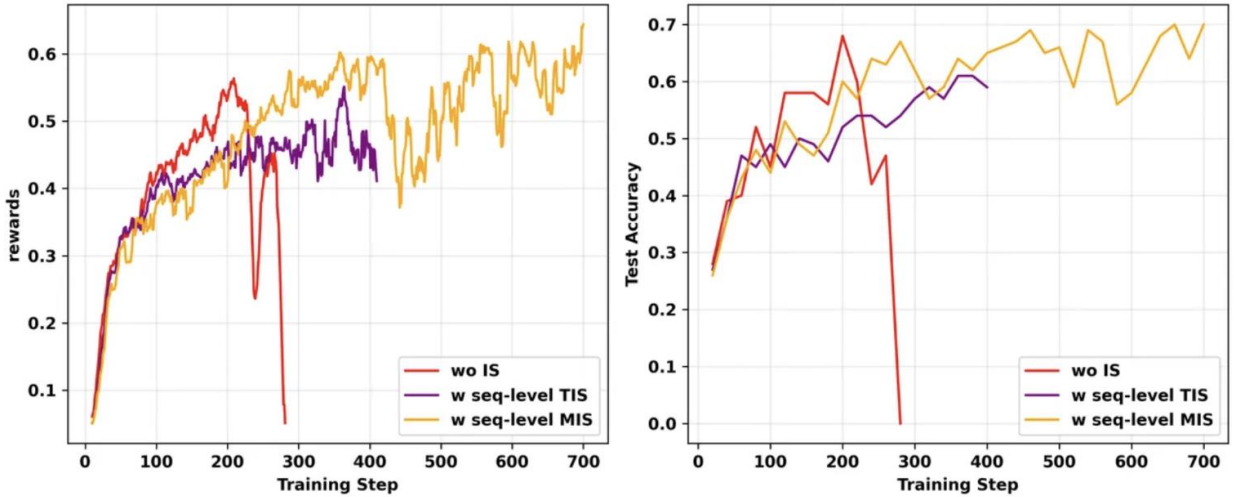
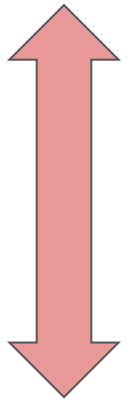


Figure 17: *Comparison of on-policy GRPO with sequence level MIS, TIS, and the vanilla version without IS. All three experiments were conducted on L20 GPUs. Compared to the vanilla on-policy experiment, the introduction of both MIS and TIS extended the training duration, but both also suffered from training instability in the later stages. Compared to TIS, MIS shows improvements in both training reward and test set accuracy, also surpassing the peak performance of the vanilla experiment without IS.* ***Left:*** *the dynamics of training rewards during RL training.* ***Right:*** *the dynamics of test accuracy during RL training.*

Fixing training-rollout mismatch

- Numerically stable RL algorithm (for MoE)
  - GSPO, SAPO: sequence level importance ratio
  - Router Replay: cache the activated experts
  - Issue: expert-activation volatility of MoE models
- Correct importance sampling ratio
  - Truncated Importance Sampling (TIS)
  - FlashRL: 8Bit Rollouts (vLLM patch)
  - Masked Importance Sampling (MIS)
- Higher floating point precision
  - Use FP16 instead of BF16
  - LayerCast (vLLM patch): optimized FP32 inference pipeline
- Deterministic kernels
  - Batch-invariant kernel
  - TP-invariant kernel: Tree-Based Invariant Kernels (TBIK)

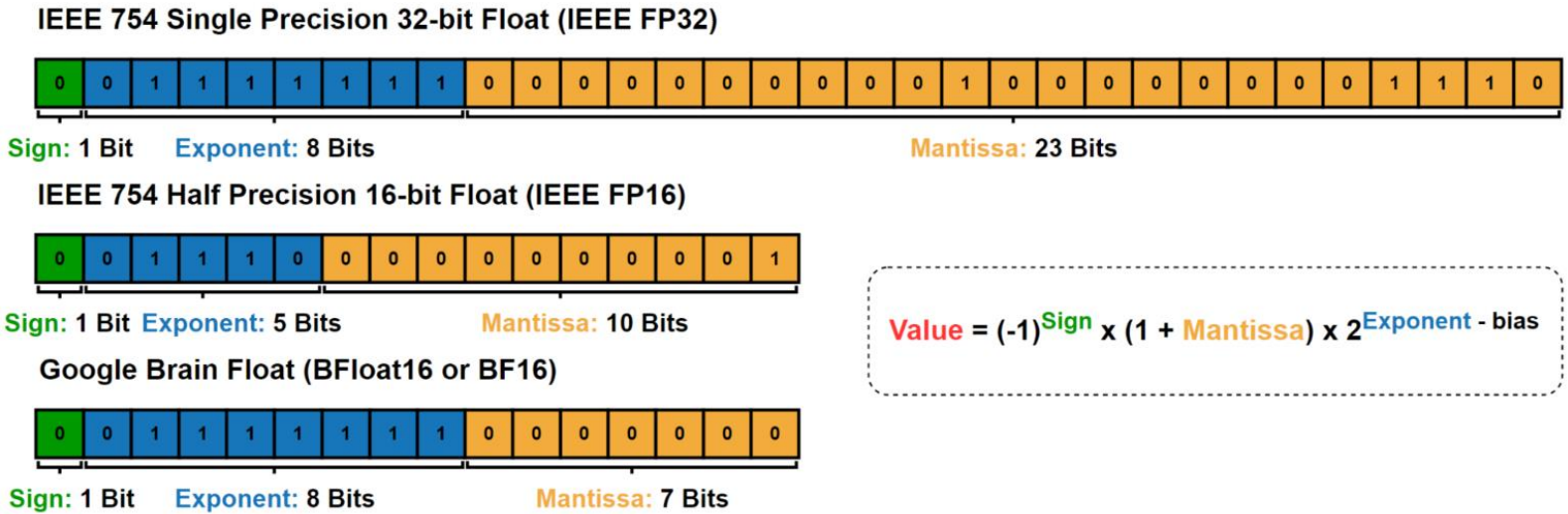
- Lightweight
- Cannot completely close the gap



- Large overhead
- Completely close the gap

Use higher floating point precision

- FP16 has higher precision than BF16 while has a smaller range
- BF16 is more stable for training (same exponent as FP32), while at inference time FP16 gives better precision



(https://arxiv.org/abs/2506.09501)

Use higher floating point precision

- Consider switch to FP16/FP32 when performing rollouts

Table 3: Std@Acc of greedy decoding across 12 different settings (GPU types, GPU counts, and batch sizes) under BF16, FP16, and FP32 Numerical Precisions. Reasoning models also exhibit larger variance compared to non-reasoning counterparts. More results can be found in Appendix C.

	AIME'24			MATH500			LiveCodeBench-Easy		
	BF16	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**
DeepSeek-R1-Distill-Qwen-7B	9.15%	5.74%	0	1.04%	1.12%	0.12%	1.67%	1.28%	0.37%
DeepSeek-R1-Distill-Llama-8B	4.60%	6.00%	5.8e-17	1.59%	0.73%	0.23%	2.31%	1.92%	0.29%
Qwen2.5-7B-Instruct	1.71%	1.45e-17	1.45e-17	0.83%	0.36%	1.16e-16	0.79%	0.48%	1.16e-16
Llama-3.1-8B-Instruct	1.92%	1.30%	0	0.94%	0.34%	0.13%	1.00%	0.67%	0.25%

(https://arxiv.org/abs/2506.09501)

Switch to FP16 for RL training

Loss scaling to keep backpropagation in the range of FP16:

1. The loss is multiplied by a large scaling factor S before backpropagation.

2. This scales up all gradients by S, shifting small gradient values out of the underflow region and into the representable range of FP16, thus preserving them.

3. Before updating the weights, the gradients are scaled back by dividing S.

(https://arxiv.org/abs/2510.26788)

Switch to FP16 for RL training

(https://arxiv.org/abs/2510.26788)
Verl patch: https://github.com/sail-sg/Precision-RL/blob/main/verl_fp16.patch

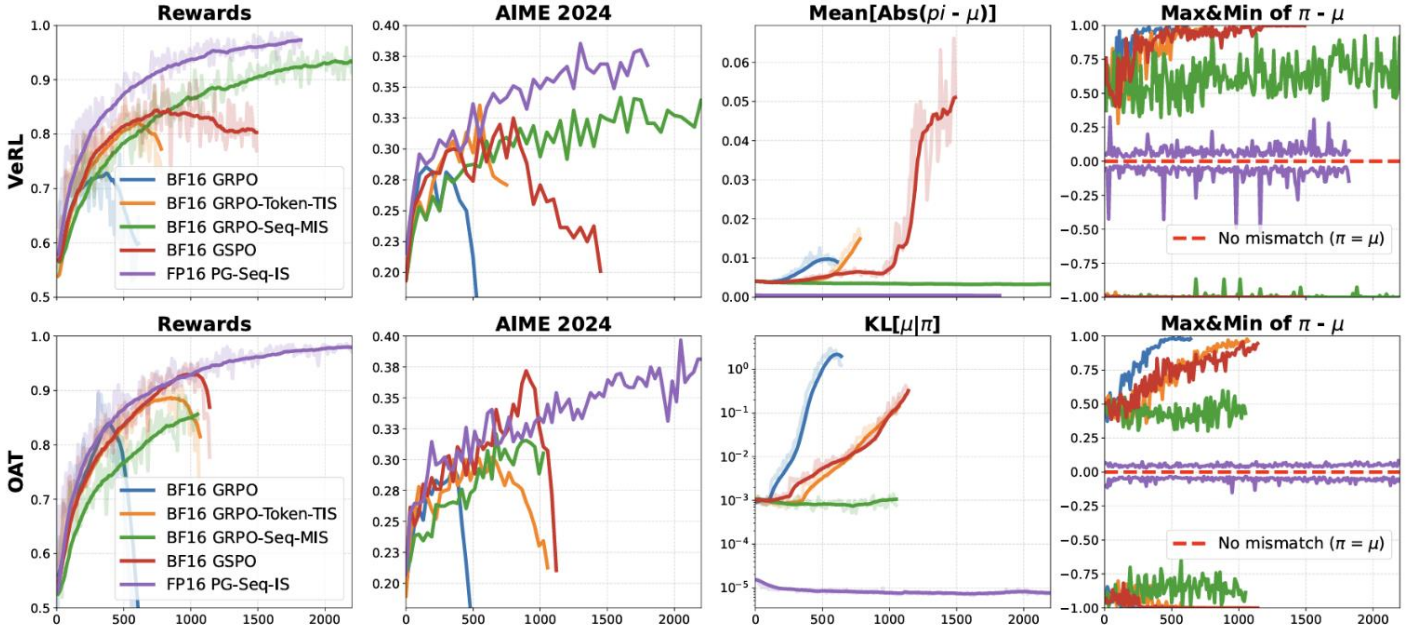


Figure 3: Simply switching from BF16 to FP16 stabilizes and prolongs RL training. The basic importance-weighted policy gradient algorithm in FP16 outperforms all baselines in BF16. Note that the third metric reported in each row slightly differs in implementation due to the use of separate codebases (VeRL and Oat). These metrics are semantically similar, and the minor differences do not affect our conclusions.

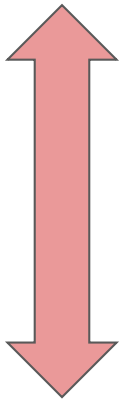
Switch to FP32 for inference

- Naively using FP32 doubles the memory usage and inference time compared to BF16
- LayerCast
  - Loading the model parameters initially in FP32 precision
  - Explicitly casting all linear layer weights and biases to BF16 for storage before inference
  - As inference runs, upcasting each weight back to FP32 just-in-time for matrix multiplication, one at a time.
- Achieve the same stability as FP32, with less memory
- Large inference time overhead (twice) compared to FB16

Fixing training-rollout mismatch

- Numerically stable RL algorithm (for MoE)
  - GSPO, SAPO: sequence level importance ratio
  - Router Replay: cache the activated experts
  - Issue: expert-activation volatility of MoE models
- Correct importance sampling ratio
  - Truncated Importance Sampling (TIS)
  - FlashRL: 8Bit Rollouts (vLLM patch)
  - Masked Importance Sampling (MIS)
- Higher floating point precision
  - Use FP16 instead of BF16
  - LayerCast (vLLM patch): optimized FP32 inference pipeline
- Deterministic kernels
  - Batch-invariant kernel
  - TP-invariant kernel: Tree-Based Invariant Kernels (TBIK)

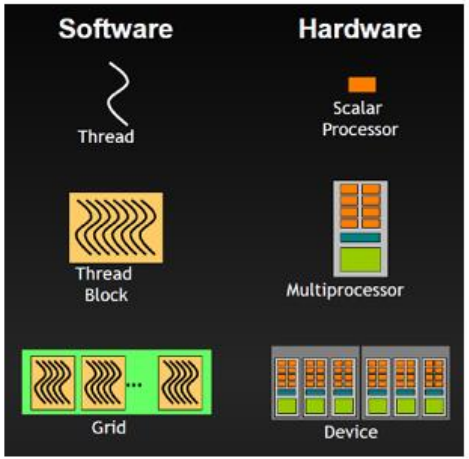
- Lightweight
- Cannot completely close the gap



- Large overhead
- Completely close the gap

GPU

- Scalar Processors execute parallel thread instructions
- Streaming Multiprocessors (SMs) each contain:
  - 64 Single Precision (FP32) units (“CUDA cores”)
  - 32 Double Precision (FP64) units
- 40-80 GB device memory (GDDR5)
- Execution Model:



A “core” (SM)

Kernel

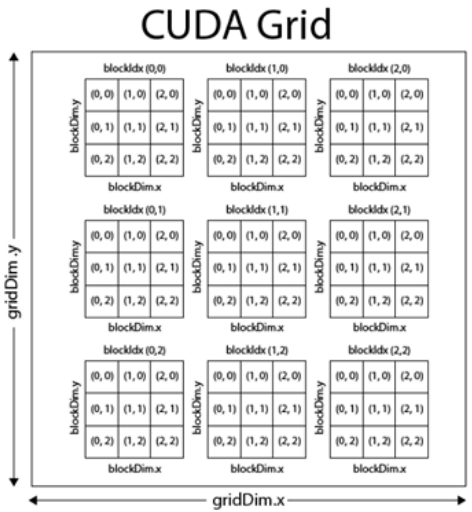
- A CUDA kernel is a function that runs on the GPU and applied independently to many elements at once.

```
// C[i][j] = A[i][j] + B[i][j]
__global__ void add2D(float *a, float *b, float *c,
int nx, int ny) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;

    if (i < nx && j < ny) {
        int idx = i + j * nx;
        c[idx] = a[idx] + b[idx];
    }
}


// Launch kernel
dim3 block(16, 16); // Often ThreadsPerBlock (16x16)
dim3 grid((nx + block.x - 1) / block.x,
          (ny + block.y - 1) / block.y);

add2D<<<grid, block>>>(d_a, d_b, d_c, nx, ny);
```



Batch-invariant kernels

- Parallelize the computation along the batch dimension and making each batch element compute independently, which guarantees a fixed reduction order regardless of the batch size.

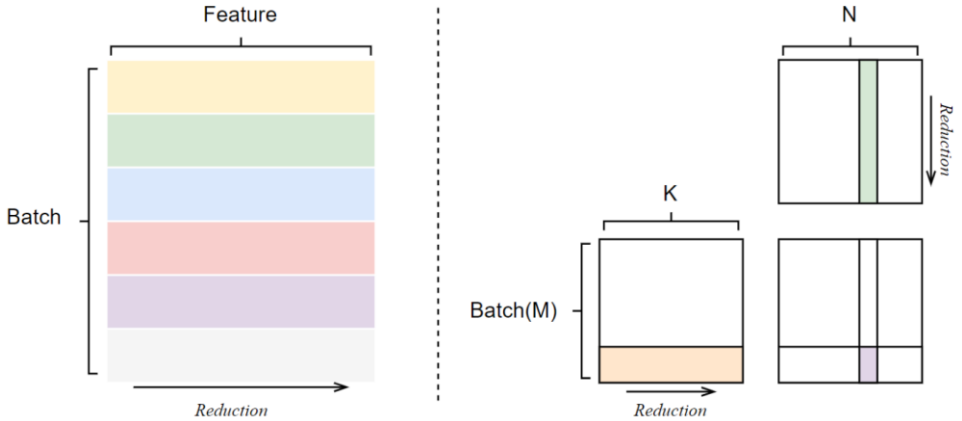


Figure 3. Implementations in Batch Invariant Operations. Left: RMSNorm. Right: MatMul.

(https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

(https://arxiv.org/abs/2511.17826)

Tree-Based Invariant Kernels

- By enforcing an identical binary-tree topology for both intra- and inter-GPU reductions, the computation order remains consistent across different TP sizes.

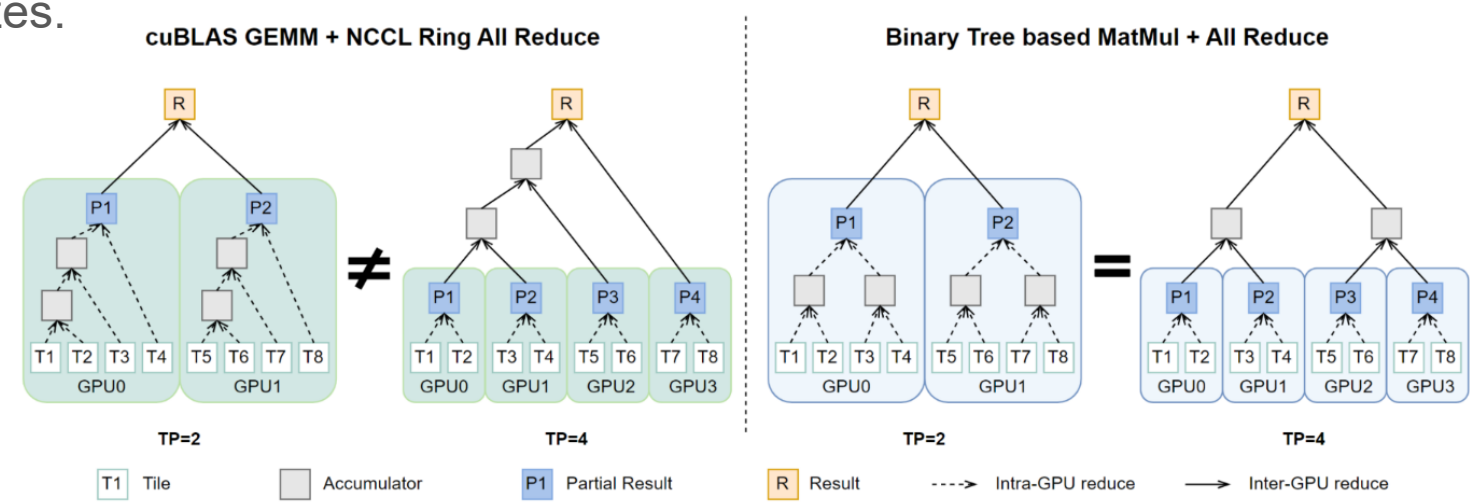
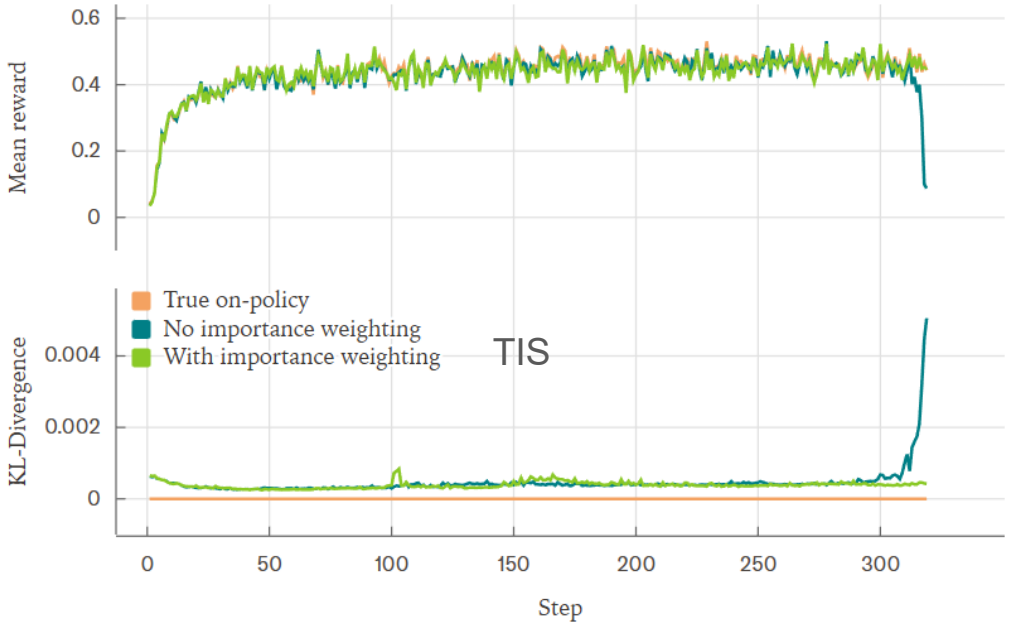


Figure 7. Comparison between the cuBLAS GEMM with NCCL Ring Reduce (left) and our Tree Reduce Kernel (right). In the Ring Reduce, inter-GPU communication follows a ring pattern, while intra-GPU reduction is performed sequentially. In contrast, our Tree Based Invariant Kernel adopts a hierarchical tree structure for both intra- and inter-GPU reductions, ensuring a consistent reduction order across different TP sizes.

Close the distribution gap

(https://arxiv.org/abs/2511.17826
https://arxiv.org/abs/2511.17826
https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

Batch invariant kernel



Tree-based invariant kernel

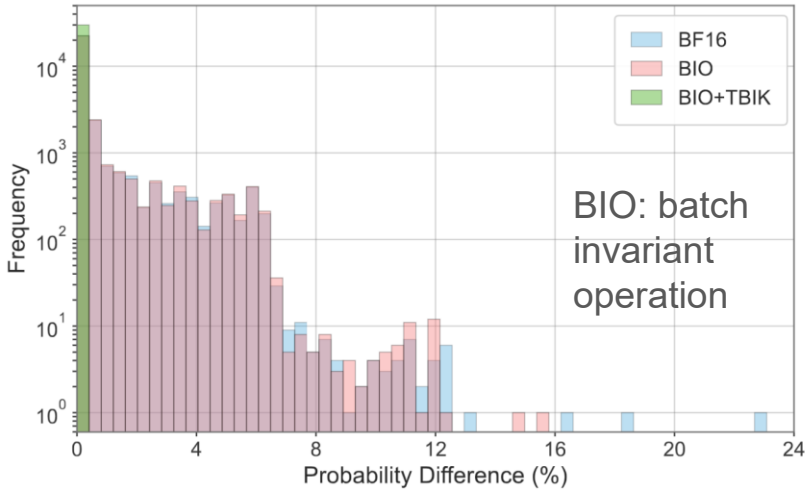


Figure 10. Maximum per-token probability difference calculated by vLLM (TP=4) and FSDP (TP=1) on Qwen3-8B using four NVIDIA L40S GPUs. Our method achieves zero probability gaps.

Large inference time overhead

Batch invariant kernel

Configuration	Time (seconds)
vLLM default	26
Unoptimized Deterministic vLLM	55
+ Improved Attention Kernel	42

Tree-based invariant kernel

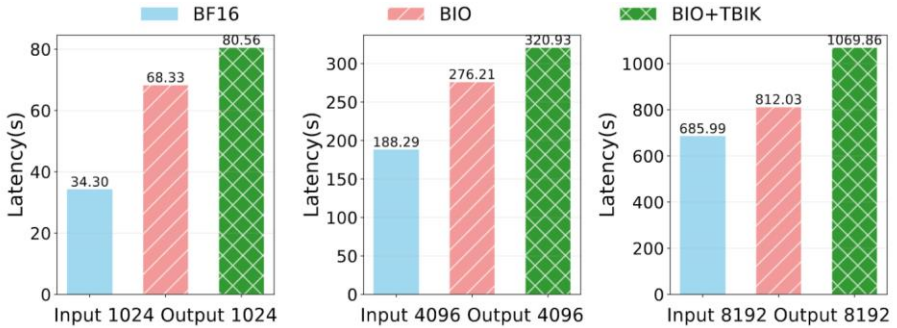


Figure 9. End-to-end latency of the Qwen3-8B model on four NVIDIA L40s GPUs under varying input/output lengths with a batch size of 64.

(https://arxiv.org/abs/2511.17826
https://arxiv.org/abs/2511.17826
https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/)

Takeaways

- Root cause of training-rollout mismatch: floating-point arithmetic is not associative
- Factors that affect LLM inference outcomes:
  - Batch size
  - Num GPUs/ TP size
  - GPU type
- Ways to reduce training-rollout mismatch:
  - Importance sampling: lightweight, good performance, still gap
  - Floating point precision choice: better performance at a cost
  - Deterministic kernels: completely close the gap, high overhead (may be greatly optimized in the near future)

Thank you!