Give me FP32 or give me death?
On the Impact of Numerical Precision to Reasoning Evaluation

Jiayi Yuan[1] Hao Li[2] Xinheng Ding[2] Wenya Xie[2] Yu-Jhe Li[3]
Wentian Zhao[3] Kun Wan[3] Jing Shi[3] Xia Hu[1] Zirui Liu[2]

1 2 3 Adobe

Code

Research Questions Covered in this Talk

- Numerical stability is a long standing problem, why reasoning model is much more sensitive to them compared to others?
- What factors impact the final results and introduce extra variance?
- What are the potential solutions to this problem?

Suggestions

- If use greedy decoding for token-level reproducibility, run it in FP32 (Unlike final acc, token length is extremely sensitive) !

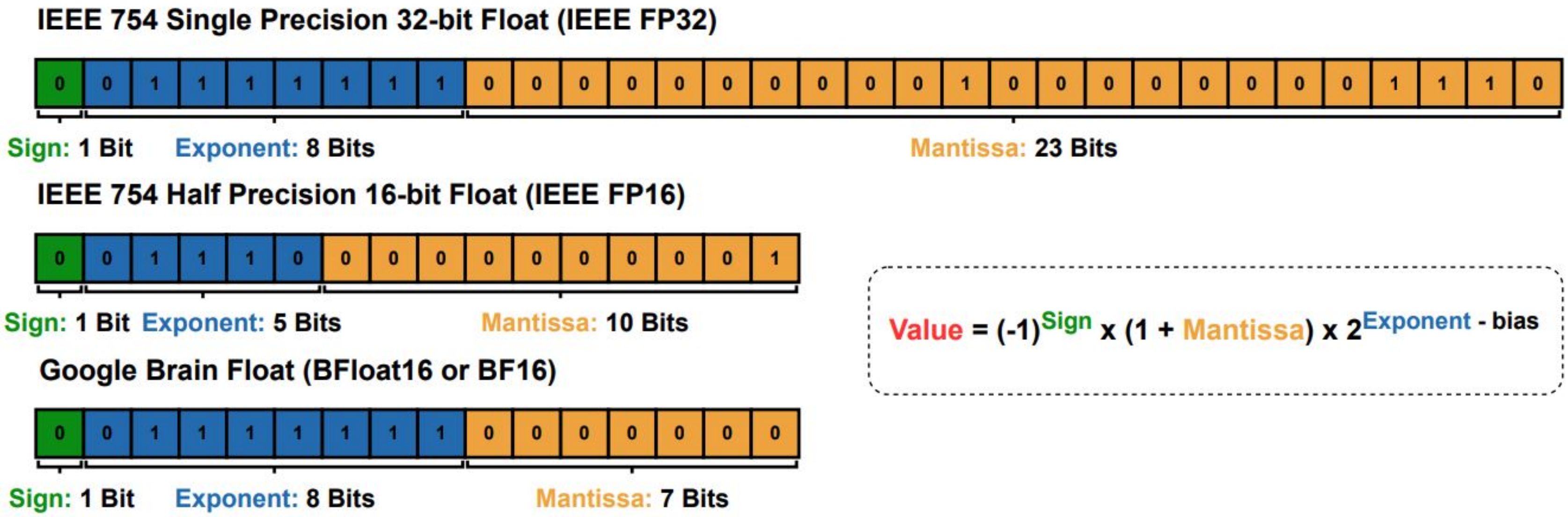
- For top-p/top-k or other random sampling based method,
    - Run it with more trials, especially for smaller dataset like AIME/AMC. We suggest at least 16 trials for small datasets.
    - Report std & error bar

Talk Outline

- **Background on the Numerical Precision**
- Our key findings and suggestions on the reasoning eval.
- Why reasoning model is so sensitive?

IEEE 754 Data Format

- TL;DR: # Exponent bits control the numerical range, # Mantissa bits control the "precision"



Low Precision Inference is the common Practice

- Lower Precision, higher Throughput (almost linear)

Technical Specifications		
	H100 SXM	**H100 NVL**
FP64	34 teraFLOPS	30 teraFLOPS
FP64 Tensor Core	67 teraFLOPS	60 teraFLOPS
FP32	67 teraFLOPS	60 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS	835 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS	3,341 teraFLOPS
INT8 Tensor Core*	3,958 TOPS	3,341 TOPS

https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306

Floating Point Arithmetic is not Associative

- Because rounding errors $(a+b)+c \neq a+(b+c)$
- One Interesting Rounding Error Example:

Pytorch 2.6.0

```
>>>import torch
>>>a = torch.tensor(8.125, dtype=torch.float16)
>>>b = torch.tensor(1032, dtype=torch.float16)
>>>a + 1024 == b
tensor(True)
```

https://zirui-ray-liu.github.io/blog/2024/rounding-error/

Many Existing Efforts

- Algorithm Side:
  - Kahan Summation, Exact Dot Product Accumulator
- Hardware Side: CUDA FMA

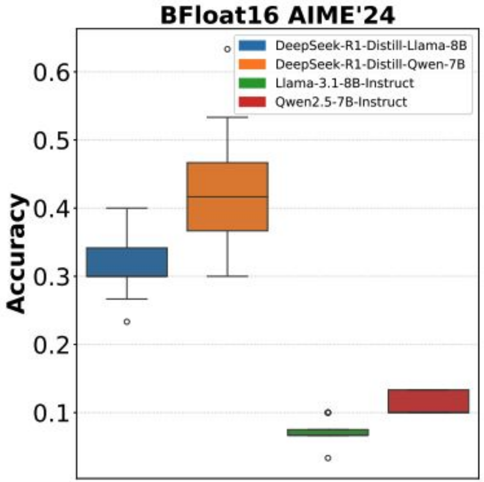
2.3. The Fused Multiply-Add (FMA)

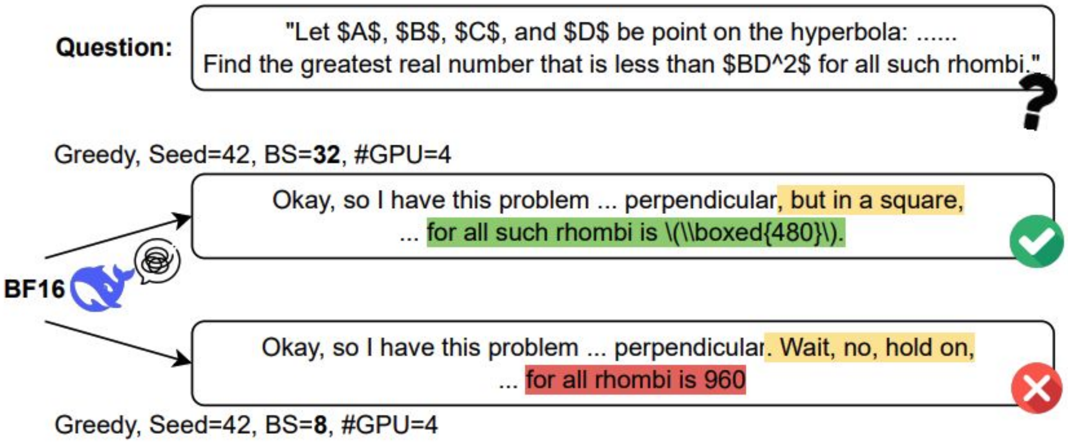
In 2008 the IEEE 754 standard was revised to include the fused multiply-add operation (FMA). The FMA operation computes $\mathrm{rn}(X \times Y + Z)$ with only one rounding step. Without the FMA operation the result would have to be computed as $\mathrm{rn}\left(\mathrm{rn}(X \times Y) + Z\right)$ with two rounding steps, one for multiply and one for add. Because the FMA uses only a single rounding step the result is computed more accurately.

https://docs.nvidia.com/cuda/floating-point/index.html

But Still…

- Huge Extra Variance Caused by Numerical Issues
    - TL; DR: Changing the eval. batch size/GPU count/GPU version change models' results under greedy decoding





Outline

- Background on the Numerical Precision
- **Our key findings and suggestions on the reasoning eval.**
- Why reasoning model is so sensitive?

LLM Decoding Strategy

- (Deterministic) Greedy decoding: choose token with highest prob.
- (Random)Top-p/nucleus sampling: Set a threshold p and sample from the top tokens with cumulative probability < p
- (Random) Top-k sampling: Set a threshold k and only sample from the top-k tokens

Commonly Adopted Evaluation Strategy

- For greedy decoding, report single run Acc.
- For random sampling based method, report Avg. Acc/Pass@1

Our Experimental Setting

- We change the # of GPUs, GPU version, and eval. batch size
  - Changing these factors will impact the micro-level token scheduler, underlying kernel selection & implementation, thus changing the floating point arithmetic orders
- System: vLLM 0.8.2

Our Challenge to Existing Evaluation Strategy

- For greedy decoding, report single run Acc.
  - We show that hardware and sys config (GPU version/TP size/Eval. BS), can significantly change the acc. (often ~2%, for AIME, 9%)
- For random sampling based method, report Avg. Acc/Pass@1
  - We show that hardware and sys config (GPU version/TP size/Eval. BS), can introduce extra 0.3-2% variance
  - And it is harmful when researcher doesn't report error bars!

Our Challenge to Existing Evaluation Strategy

- For greedy decoding, report single run Acc.
  - GPU version/Counts/Eval. BS, can significantly change the acc. (often ~2%, for AIME, 9%)
  - Suggest: If you still keep token-level reproducibility, use FP32

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

Our Challenge to Existing Evaluation Strategy

- For random sampling based method, report Avg. Acc/Pass@1
  - GPU version/counts/Eval. BS), can introduce extra 0.3-2% variance
  - Suggest: run it with more trials! Report both Acc & Error bars

	MATH500 (n=4)			AIME'24 (n=16)			AIME'24 (n=64)		
	BF16	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**
DeepSeek-R1-Distill-Qwen-7B	0.3158	0.1463	0.1021	1.7151	0.8273	1.1785	0.3749	0.5391	0.7377
DeepSeek-R1-Distill-Llama-8B	0.3602	0.3371	0.1211	1.5124	1.8792	0.8606	0.8774	0.8539	0.5034
Qwen2.5-7B-Instruct	0.4663	0.1686	0.0274	0.7056	0.2523	0	0.1784	0.1382	0
Llama-3.1-8B-Instruct	0.6020	0.1725	0.3293	0.5992	0.2282	0.7759	0.4216	0.2898	0.1296

Our Challenge to Existing Evaluation Strategy

- Why a Extra 0.3-2% std. is concerning to me:
  - On small datasets like AIME/AMC, 16 runs are with ~2% variance purely from hardware/sys; 4–8 runs are definitely more random
  - 95% conf. Interval requires 2 sigma acc.
    - If std. from hardware is 2%, you need at least 4% acc. ↑

	MATH500 (n=4)			AIME'24 (n=16)			AIME'24 (n=64)		
	BF16	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**
DeepSeek-R1-Distill-Qwen-7B	0.3158	0.1463	0.1021	1.7151	0.8273	1.1785	0.3749	0.5391	0.7377
DeepSeek-R1-Distill-Llama-8B	0.3602	0.3371	0.1211	1.5124	1.8792	0.8606	0.8774	0.8539	0.5034
Qwen2.5-7B-Instruct	0.4663	0.1686	0.0274	0.7056	0.2523	0	0.1784	0.1382	0
Llama-3.1-8B-Instruct	0.6020	0.1725	0.3293	0.5992	0.2282	0.7759	0.4216	0.2898	0.1296

Summary

- For greedy decoding, report single run Acc.
  - Hardware and sys config (GPU version/Counts/Eval. BS), can significantly change the acc. (often ~2%, for AIME, 9%)
- For random sampling based method, report Avg. Acc/Pass@1
  - Hardware and sys config (GPU version/TP size/Eval. BS), can introduce extra 0.3-2% variance
  - And it is harmful when researcher doesn't report error bars!

Suggestions

- If use greedy decoding for token-level reproducibility, run it in FP32

- For random sampling based method,
  - Run it with more trials, especially for smaller dataset like AIME/AMC. We suggest at least 16 trials
  - Report std & error bars

Outline

- Background on the Numerical Precision
- Our key findings and suggestions on the reasoning eval.
- **Why reasoning model is so sensitive?**

Reasoning is more sensitive

- In all our experiments, we find
  - Reasoning model is much more sensitive to numerical errors
  - It makes sense since they generate more tokens,

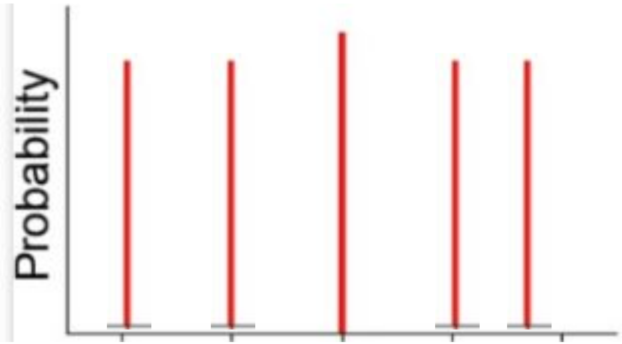
	AIME'24			MATH500		
	BF16	**FP16**	**FP32**	**BF16**	**FP16**	**FP32**
DeepSeek-R1-Distill-Qwen-7B	9.15%	5.74%	0	1.04%	1.12%	0.12%
DeepSeek-R1-Distill-Llama-8B	4.60%	6.00%	5.8e-17	1.59%	0.73%	0.23%
Qwen2.5-7B-Instruct	1.71%	1.45e-17	1.45e-17	0.83%	0.36%	1.16e-16
Llama-3.1-8B-Instruct	1.92%	1.30%	0	0.94%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

Analysis: Why Reasoning is more sensitive?

- Motivating Example: Token Distribution that is extremely numerical Robust – One Hot
  - Small Rounding Error cannot change much in this case, no matter in Top-P sampling/Greedy decoding

Analysis: Why Reasoning is more sensitive?

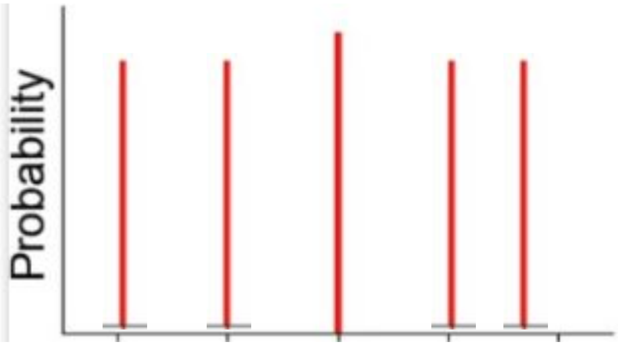
- Motivating Example: For Reasoning Model, this is often the case:
  - Many token's prob. is very close to each other
    - In greedy decoding, small rounding error change the rank.
    - Similarly, top-K/top-P sampling may exclude/include extra tailing tokens



Analysis: Why Reasoning is more sensitive?

- One Concrete example in Greedy decoding

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%
...	...	...	...



My hypothesis: This is not a bug, but a feature

- During RL/SFT, we have many high-entropy minority tokens
- This makes sense because in this way, we can increase the answer diversity and get better rewards in RL

Research Questions Covered in this Talk

- Numerical stability is a long standing problem, why reasoning model is much more sensitive to them compared to others?
    - High Entropy of some minority tokens
- What factors impact the final results and introduce extra variance?
    - GPU version, counts, batch size
- What are the potential solutions to this problem?
    - Token-level reproducibility, use FP32
    - For Random sampling, Run it with more trials & report std!