Defeating Nondeterminism in LLM Inference

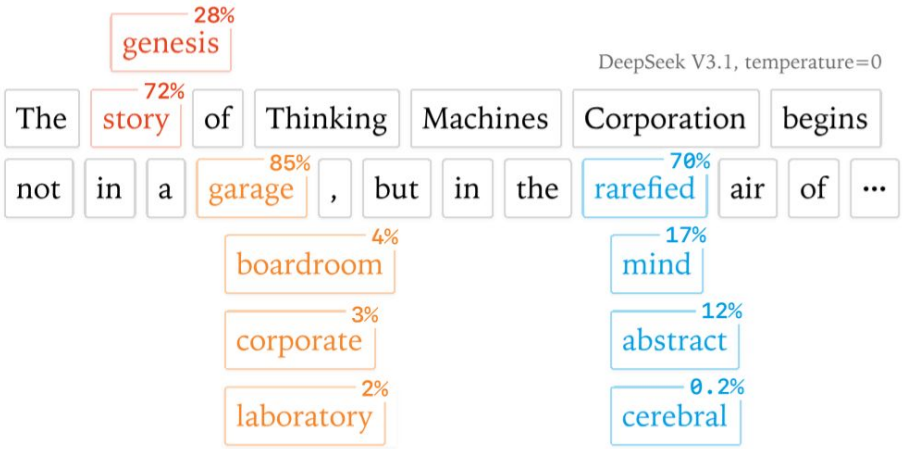
Horace He in collaboration with Thinking Machines

Presented by: Sabrin Nowrin

BACKGROUND

Reproducibility is essential for scientific progress.

LLMs often produce different outputs for the same input



Why is this expected?

- LLMs use sampling → probabilistic token selection
- Different outputs are normal when randomness is enabled
- Even with temperature = 0 (greedy decoding), LLMs are still not deterministic in practice

Nondeterminism

- Same input, code, and environment
- Running the same program more than once
- Produces different outputs
- Breaks reproducibility and reliability

“Concurrency + Floating point” Hypothesis

if the order in which concurrent threads finish is nondeterministic and the accumulation order depends on the order in which concurrent threads finish (such as with an atomic add), the accumulation order will be nondeterministic as well.”

COMMON ASSUMPTION

LLM nondeterminism is caused by

- **Floating-point non-associativity**

$$(a + b) + c \neq a + (b + c)$$

- **Concurrent GPU execution**

Concurrency + floating-point = nondeterminism

Floating-point non-associativity

- Computers have limited memory ➜ only limited digits can be stored.
- Floating-point values store nearby representations, not exact numbers.
- Every calculation step involves rounding to fit limited precision.
- As a result, changing the order of addition can produce slightly different outcomes.

Concurrency & Floating point

```
import random

vals = [1e-10, 1e-5, 1e-2, 1]
vals = vals + [-v for v in vals]

results = []
random.seed(42)
for _ in range(10000):
    random.shuffle(vals)
    results.append(sum(vals))

results = sorted(set(results))
print(f"There are {len(results)} unique results: {results}")
```

Output:
There are 258 unique results: [-2.220446049250313e-16, -2.137705678287655e-16, -1.6480741981475503e-16 , ..., 8.326672684688674e-17]

ATOMIC ADD

GPUs run many cores in parallel

Atomic add = safely add to a shared value (no conflicts)

Guarantees correctness, not order

Order depends on which core finishes first

Atomic adds change order, causing floating-point differences.

Why modern GPU kernels rarely need atomic adds?

- Atomic adds are not necessary for the vast majority of kernels.

- Sufficient batch parallelism: Parallelize across batch dimension, so each core handles one full reduction → no need for atomic adds.

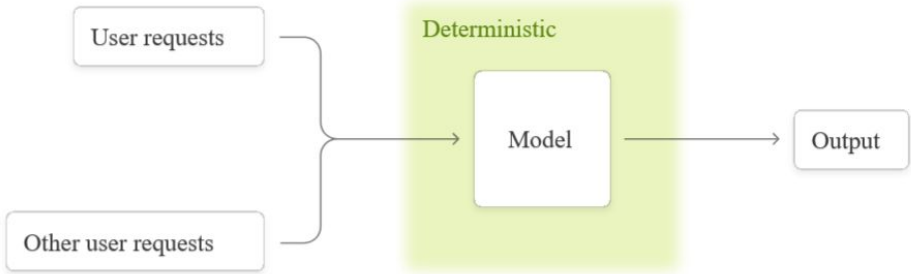
- Deterministic reduction strategies: Use split reductions and combine results in a fixed order (via cleanup or synchronization) to maintain determinism without performance loss.

What if nondeterminism is actually happening outside GPU?

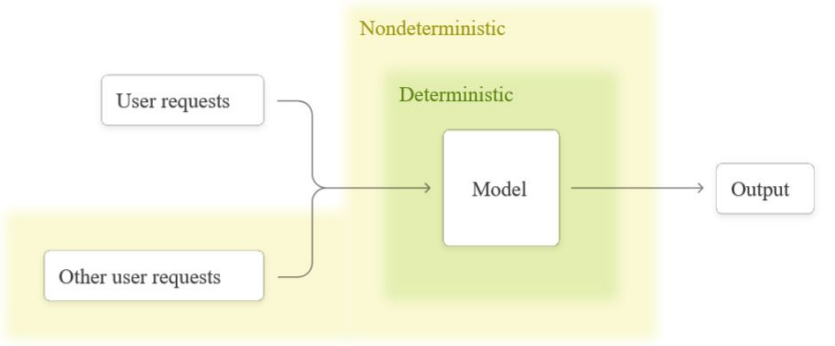
Determinism in the LLM Forward Pass

- No atomic adds in forward pass
  - LLM forward computation avoids shared updates

- Run-to-run deterministic behavior
  - Same input produces identical outputs across runs



System-Level Sources of Nondeterminism



- Deterministic forward pass ≠ deterministic system
- Dependence on parallel requests
  - Batch size (other user requests) influences computation
- Lack of batch invariance
  - Same input can produce different outputs under different batch sizes

Deterministic System, Nondeterministic Experience

- ✓ System may be deterministic internally
- ✓ Other users are not visible inputs
- ✓ Server load introduces unpredictable batch sizes
- ✓ Lack of batch invariance makes output depend on batch size
- ✓ Batch dependence + varying server load → nondeterministic behavior

Root Cause

- ✔ Primary cause of nondeterminism for all LLM inference
  - Server load varies unpredictably, which changes batch size and leads to different outputs

- ✔ Not GPU-specific
  - the same issue occurs on CPUs and TPUs as well

How to make kernels batch-invariant

- To make transformer implementation batch-invariant, all kernels must be batch-invariant
- Pointwise operations are already safe
- Only reduction operations need attention

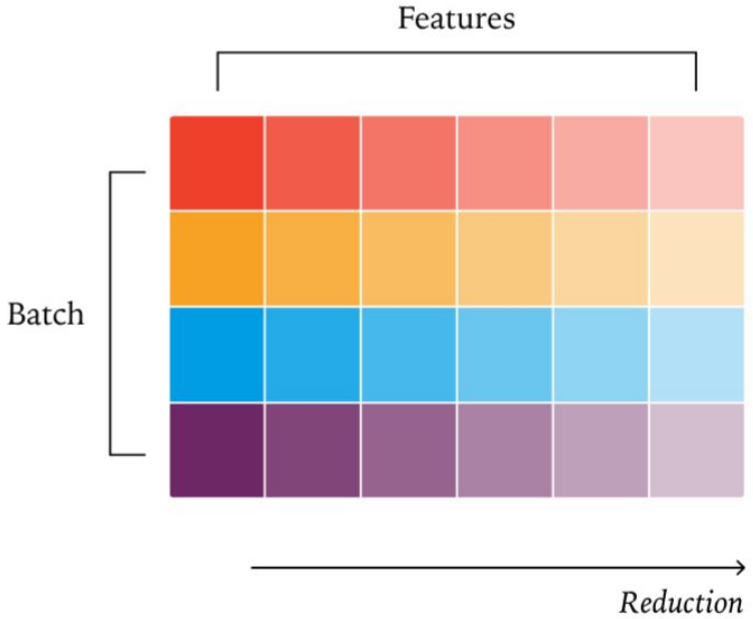
Key operations

RMSNorm | Matrix Multiplication | Attention

RMSNorm

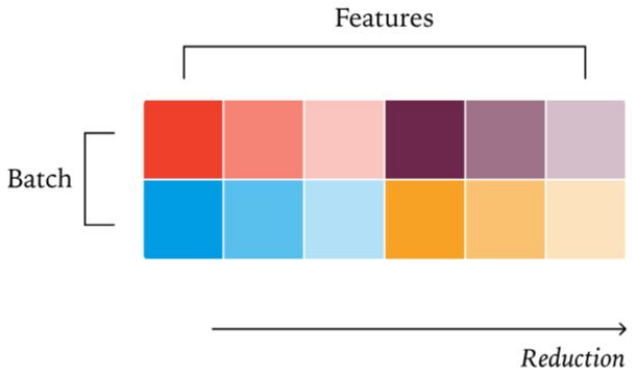
Batch-invariant RMSNorm

- Give each core one data sample (row)
- Each core computes independently (no communication)
- This is data-parallel strategy
- Reduction order must stay the same regardless of batch size (batch invariance)



When Batch Size Breaks Invariance

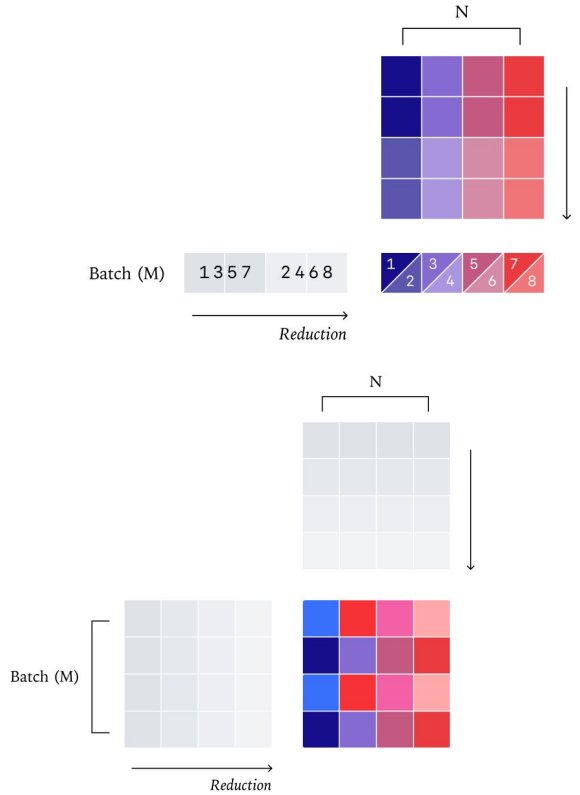
- Small batch → less parallelism, but fast anyway
- Solution: ignore optimization for small batches
- Alternative: use one consistent reduction strategy for all sizes
- Trade-off: slightly lower peak performance, but maintains batch invariance



Matrix Multiplication

Batch-Invariant Matrix Multiplication

- Matmul = pointwise multiply + reduction
- GPUs compute matmul in tiles across cores
- [1024, K] X [K, 1024] with [128×128] size → only 64 tiles
- Small M, N → use Split-K matmul
- Small shapes may also require different tensor core instructions
- Split-K changes how sums are computed (differe order)

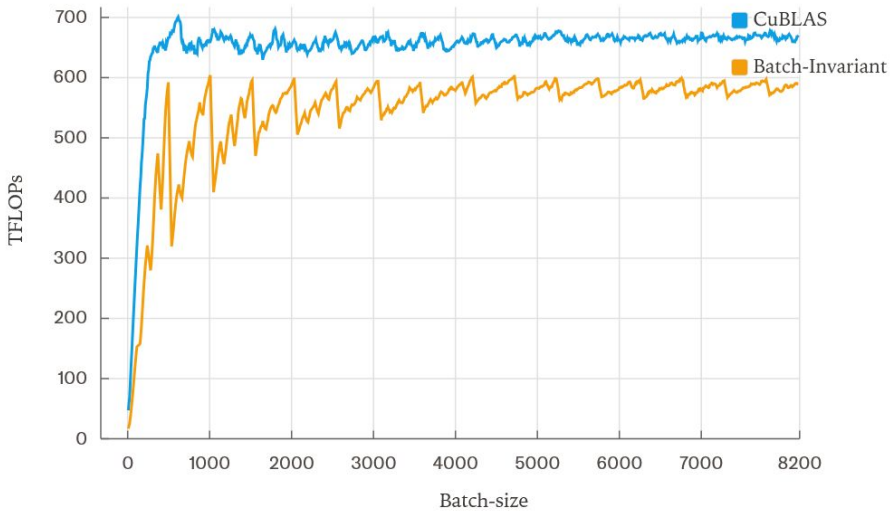


Ensuring Batch Invariance in Matmul

- ✔ Use one fixed matmul kernel for all input shapes
- ✔ Same kernel → same reduction order → batch invariance
- ✔ This may reduce performance slightly
- ✔ But in LLMs, N/model dimension is usually large
- ✔ Large N gives enough parallel work, so Split-K is often unnecessary

Batch-Invariant Matrix Multiplication

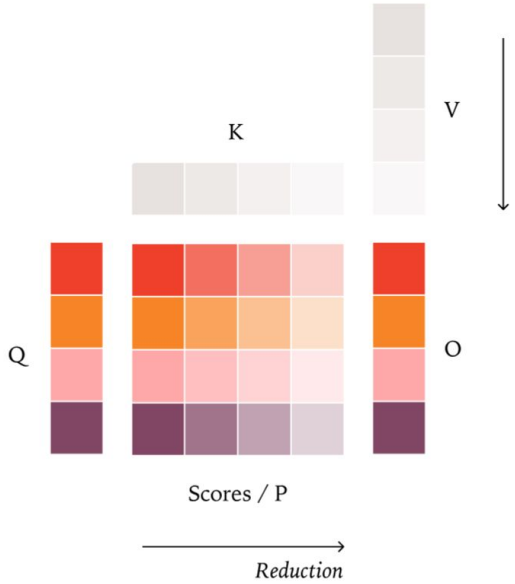
- X-axis (Batch-size): inputs
- Y-axis (TFLOPs): throughput
- Despite obtaining batch invariance, we only lose about 20% performance compared to cuBLAS.



Attention

Attention Adds Complexity

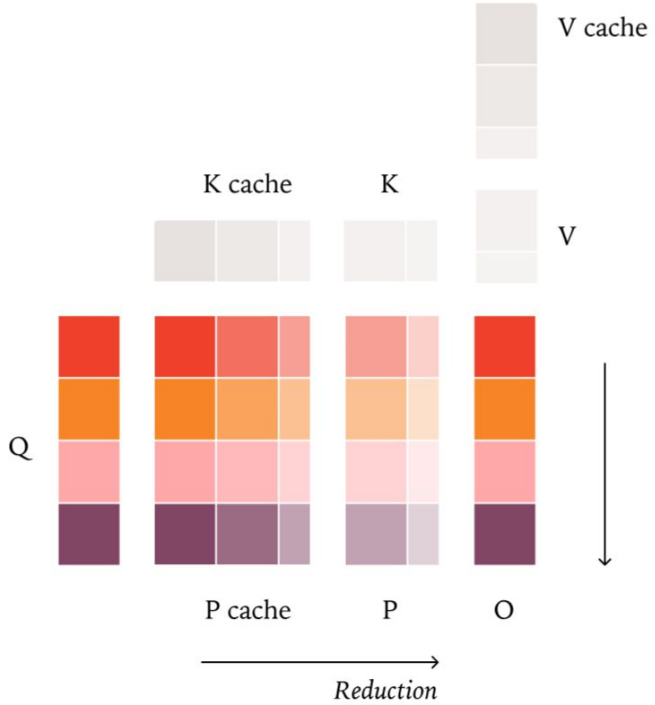
- ✔ Attention has two matmuls ➔ more complexity
- ✔ Reduces over feature + sequence dimensions
- ✔ Needs to handles how sequences get processed
- ✔ Same token must have same reduction order
  ➔ Required for batch invariance



Where it Breaks

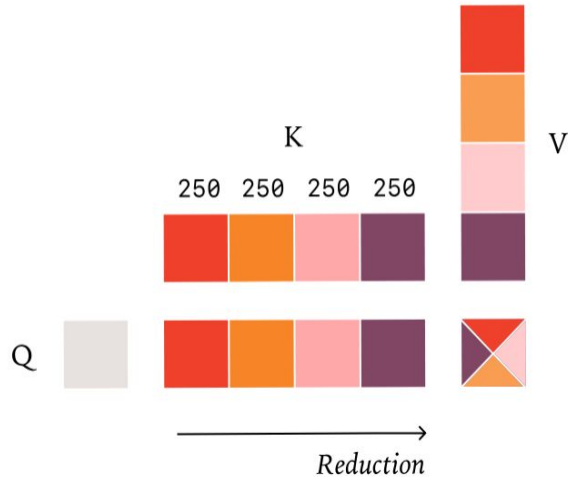
- Sequence may be processed:
  - all at once (prefill)
  - in parts (decoding)

- KV cache + current tokens handled separately
  - ➔ Different reduction order
  - ➔ Breaks batch invariance



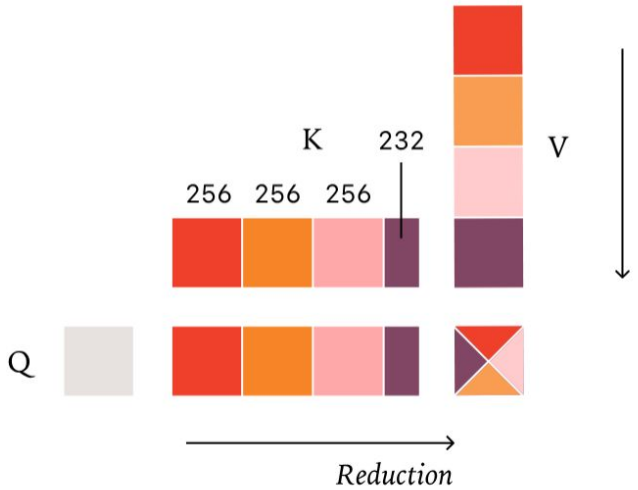
Fixing KV Handling

- ✓ Update KV cache before attention
- ✓ Keep keys/values in one consistent layout
- ✓ Avoid separate reductions for cache vs new tokens
  → Ensures consistent computation
- ✓ Attention often uses split-reduction (Split-KV)
- ✓ Dynamic strategies change with batch size
  → Not batch-invariant



Correct Strategy

- Use fixed split-size (not fixed number of splits)
- Same chunk size → same reduction order
- Number of splits may vary
- Maintains batch invariance



Experiments

Experiment Setup

Model: Qwen/Qwen3-235B-A22B-Instruct-2507

Prompt: “Tell me about Richard Feynman”

1000 runs, temperature = 0

Generate 1000 tokens each

Results (Before Fix)

80 unique outputs from same prompt

Most common output: **78 times**

Outputs identical until **token 102**

Divergence starts at **token 103**

Results (After Fix)

Using batch-invariant kernels

1000 / 1000 outputs identical

Matches theoretical expectation

Performance

- ✔ Built on vLLM with FlexAttention backend
- ✔ Used torch.Library to replace key operators
- ✔ Integrated batch-invariant kernels

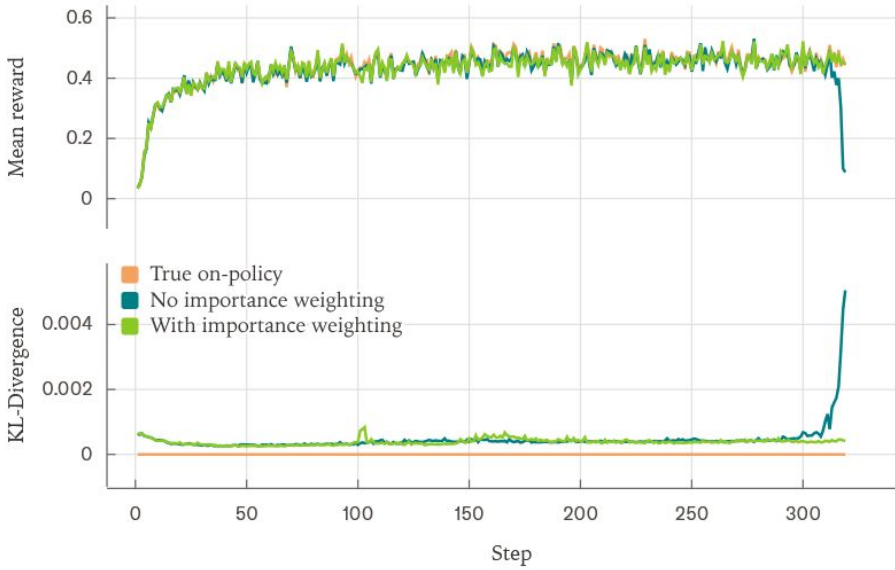
Configuration	Time (seconds)
vLLM default	26
Unoptimized Deterministic vLLM	55
+ Improved Attention Kernel	42

Conclusion

- Floating-point rounding causes differences in computations.
- These differences become visible because batch-dependent processing changes how computations are performed.
- With fixed, batch-invariant computation, LLMs can become fully deterministic and reliable

Experiments

- Nondeterministic inference makes on-policy RL behave like off-policy RL, hurting training
- Without correction, training becomes unstable, but importance weighting helps stabilize it
- With deterministic inference, we get true on-policy RL (KL = 0) and stable training



Reference

- He, Horace and Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference", Thinking Machines Lab: Connectionism, Sep 2025.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” Advances in neural information processing systems, vol. 30, 2017
- S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, “An empirical study of the non-determinism of chatgpt in code generation,” arXiv preprint arXiv:2308.02828, 2023.

Questions?