Batch-invariance: Our Testing and Observations on Current Community Efforts

Anyu Yang Junbeom In Xiaocong Zhang

December 8, 2025

1 Intro

In short, batch invariance vaguely means the LLMs' inferencing gets for the same input the same results in various batches, which is one building block of overall determinism of LLMs.

We have replicated and tested the software setup of the initial vLLM batch invariance pull request[1] (PR). On our machines, we found all the small models we tested are **NOT** batch invariant. We haven't tested the exact same model (Qwen3-8B) claimed in the PR, so it doesn't directly veto the legitimacy of the PR.

Nonetheless, with preliminary testing, we show that in this PR the variances are still prevalent in terms of models and graphics processing units (GPUs). Further, we strongly suspect the Qwen3-8B to not be batch invariant, at least on our GPUs. Ultimately, we find the ongoing efforts to reach batch invariance by the community unreliable due to current methodology (naive and insufficient testing).

2 The Definition of Batch Invariance

The discussion of batch invariance sits under the general topic of determinism. Since LLMs itself is probabilistic, it's hard to realize the issue of non-determinism. But if we put aside randomness in algorithms, the persisting non-determinism might surprise people sometimes. Then we see the effect of batching and its intrication with the implementation of GPU kernels.

From reading the blog, it seems that the definition of "batch invariance" is, not getting different results due to different positions in a batch or different batch sizes.

In other words, when the batch size changes, each element in the batch can get different results." This is how batch variance is introduced in the original blog.

But it raises some questions:

- If we accept the definition, does batch invariance also imply different batches should have the same results (given the same input)?

[1] We also tested the batch invariance feature in the main brach vLLM. But since our machines don't have the required Compute Capability 9.0, the observed variances carry less significance.

- Is batch invariance the only thing once we exclude randomness in algorithm from non-determinism in LLM? (logical complement)

For the second question: If so, we would try prove or at least, reason about it. If not, we surely want to find a new source of non-determinism.

3 Tracking Community Activity

The initial blog and the accompanying PR did attract attention. Since then, the community has been working on it actively. Most notably, the batch invariance has been adopted in official vLLM release, with dedicated documentation[2].

Now batch invariance is a project[3] under vLLM, with 19 pull requests in total. Some new features include expanding batch invariance to multiple GPUs (Tensor Parallism), supporting DeepGEMM and Blackwell, etc.

The consensus is, this feature helps debugging and the quality of reinforcement learning.

4 GPUs, Environments and Models

We tested with two machines, Ubuntu 22.04.1 with RTX 3060 6G and WSL2 Ubuntu 20.04.6 with RTX 4060 8G. While the community is using datacenter GPUs [4] (H100, H200, etc.) and larger models, testing smaller models on consumer-grade GPUs is also meaningful given the nature of this concept. As we are not providing strong guarantees as in formal verification, it's necessary to see real results on different machines.

The relevant packages' versions are:

- cuda 12.8, 12.9
- cuda-toolkit 12.9
- vllm pr-24583
- torch 2.9.0
- torchvision 0.24.0
- torchaudio 2.9.0
- xformers 0.0.33.post1

See Appendix A about why we used torch 2.9.0 and other compatible packages, instead of what's required in configuration files.

The relevant models we're able to cover:

- "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

[2]https://docs.vllm.ai/en/latest/features/batch_invariance/#future-improvements

[3]https://github.com/orgs/vllm-project/projects/29/views/1

[4]The official vLLM only supports batch invariance feature for Compute Capability (CC) 9.0

- "Qwen/Qwen2.5-1.5B-Instruct"
- "Qwen/Qwen3-0.6B"
- "Qwen/Qwen3-1.7B"
- "Qwen/Qwen3-4B-Instruct-2507-FP8"

5 Experiments and Observations

We are running the provided `deterministic_vllm_inference.py`, which concurrently sends 1000 identical requests to a vLLM server.

To discover variances, we tweak and utilize the script as such:

- We set the maximum output tokens to 400 or 500.
- We write prompts to induce longer answers. See Appendix B.1 for examples.
- Not surprisingly, we change relevant parameters and run the scripts multiple times.

5.1 Results

The result is, **we have found NONE of the models we tested to be batch invariant.**

The number of unique samples we tested for Qwen3 models[5] in one single execution of `deterministic_vllm_inference.py` are recorded in table 1.

Model	prompt 2	prompt 3	prompt 4
Qwen/Qwen3-4B-Instruct-2507-FP8	58	6	632
Qwen/Qwen3-1.7B	2	2	7
Qwen/Qwen3-0.6B	17	3	3

Table 1: Number of unique outputs under different prompts.

The way to interpret is, as long as we see one execution with multiple unique samples, we know the kernels are not batch invariant for this model and GPU.

5.2 Observations

1. First of all, variance is ephemeral if you don't pay attention, but if you look closely, it's not hard to catch either. The results are often the same, but you can easily find 3-5 unique samples even with a tiny model (1.5 B) and a small batch size (180).
2. Larger models, longer output token sequences, larger batch sizes, larger sampling sizes are more likely to find variance. The aforementioned factors have different rationales behind them: longer answers induced by good questions lead to more iterations (before getting EOF token).

[5]We assume Qwen3-1.7B, Qwen3-0.6B and Qwen3-8B to execute the exact same code path in vLLM.

3. The test script concurrently sends 1000 requests by default, which are reasonably bursty. However, there is no guarantee that these requests would be processed in a single batch: The server could have a smaller batch size limit, or continuous batching might play a role.

4. Based on the code, comment and discussion of the PR, it's trying to be general, supporting multiple models and machines. The author claims to pass tests with Qwen3-8B, not mentioning machine requirements. When people say they test with Qwen3-30B, no objection raised by author. When people ask about support for Qwen3 moe, the answer seems to be negative. It's stated the PR doesn't support multi-GPU (tensor parallelism).

6 Discussion

Observation 3 is not an issue in practice, since the implicit expectation is to get only one unique result across all samples. But it might affect reproducibility given different configurations (batch size) of vLLM servers. This is also relevant to the definition of batch invariance.

Referring to observation 2, We think often the claims to support batch invariance are too unserious. People say, "I tested it with model X_i on machine Y_i, it passed the test." But how confident are we to believe it would work for other models (even just of different sizes), or another GPU? Equally disturbing, if people haven't tweaked the script (increase maxTokens, increase sample size, change prompts, repeat, etc.) to thoroughly test , the result is unreliable even with the exact same model and GPU.

Following that, we can confirm with observation 4 that at least for the initial PR, the workflow is author and people claiming to pass tests and get merged, without a viable means to verify. So it's likely that later these code will be found not really invariant. It's not hard to fix it when problems are discovered, but the issue is, in this situation we are never really sure if the code is batch invariant.

7 Possible Further Explorations

7.1 GPU Kernel Code Inspection

Now that we have variances found, we can manually review the kernels and reason about it. It might be vLLM- or PyTorch-related.

7.2 Review Existing Test Cases in vLLM's Batch Invariance Project

The testing for batch invariance is too naive in the initial PR, and we suppose the subsequent ones in the batch invariance project are alike. But a systematic review would help reach this conclusion.

A Inspecting What's Wrong with PyTorch 2.8.0

There are some inconsistency and confusion in pr24583: On one hand, all relevant packages (torch, torchvision, torchaudio, xformer, etc.) are requiring torch 2.8.0 in config files like `pyproject.toml` and `requirements/cuda.txt`, on the other hand, the testing script `deterministic_vllm_inference.py` specifically ask for torch 2.9.0 in comments.

Initially we dismissed the comment requesting torch 2.9.0 and went with the config files, since torch 2.9.0 hadn't been released when they publish pr24583. Then we really encountered a runtime error with torch 2.8.0, and it seems the error is a direct result of code change introduced in this pull request. The error is:

```
ValueError: Q and KV block size must be divisible by BLOCK_M and BLOCK_N.
We got Q_BLOCK_SIZE=16 and KV_BLOCK_SIZE=16.
```

It is pr24583 that fixed `BLOCK_M`, `BLOCK_N`, `Q_BLOCK_SIZE` and `KV_BLOCK_SIZE` to 16. But $16 \% 16 == 0$ should hold! After inspecting torch 2.8.0, we discovered it's a logical bug in validation of kernel options:

```
SPARSE_KV_BLOCK_SIZE % conf.block_n != 0
or SPARSE_Q_BLOCK_SIZE % conf.block_m != 0
```

Instead of `conf.block_m` and `conf.block_n`, it should have used

```
cur_kernel_options["SPARSE_KV_BLOCK_SIZE"] % cur_kernel_options["BLOCK_N"]
!= 0
or cur_kernel_options["SPARSE_Q_BLOCK_SIZE"] % cur_kernel_options["BLOCK_M"]
!= 0
```

which gets real configuration if available.

We think the bug passed functionality test because most people are using high end GPUs and common configurations. But we used a low end gaming GPU and some fixed configuration in Thinking Machines' scripts. There are great variability in model meta-parameters and all kinds of block sizes, so we might expect many similar edge cases exist. And since these edge cases are not covered, many logical loopholes remain in these frameworks.

B Experiment Artifacts

B.1 Example Prompts

Here are prompts we used to get long answers.

- prompt 1: "Tell me the rise and fall of Rome in great detail."
- prompt 2: "You are a careful mathematician. Read the problem, reason step by step, and then give ONLY the final numeric answer on the last line, prefixed by 'Answer:'. Avoid any extra commentary. Problem: Let $f(x) = 3x^2 - 5x + 7$. Compute the exact value of the sum $S = \sum_{i=1}^{500} f(i)$, and express your answer as an integer. /no_think"
- prompt 3: "Solve a multi-step math problem. First derive intermediate quantities explicitly, then compute the final answer. Avoid filler phrases. Begin directly with

the first computation step. Problem: A train travels 135 km at 72 km/h, then continues 215 km at 95 km/h. What is its total travel time in minutes? /no_think"

- prompt 4: "Write a complete Python function that parses a nested JSON structure representing a filesystem, computes total file sizes by directory, and returns the top 5 largest directories. Do not explain. Begin directly with the function definition. /no_think":

C Batch Invariants Matmul Kernel

We implemented Triton Kernel for the proof of concept. Our kernel's reduction order is independent of the batch dimension. The main idea is to parallelize the reduction dimension K and to keep the accumulation order fixed regardless of the number of lows M. Each kernel instance computes a fixed-size output.

C.1 GPU Kernel Code

```
@triton.jit
def bi_mm_kernel(
    A_ptr, B_ptr, C_ptr,
    M, N, K,
    stride_am, stride_ak,
    stride_bk, stride_bn,
    stride_cm, stride_cn,
    BLOCK_M: tl.constexpr, BLOCK_N: tl.constexpr, BLOCK_K: tl.constexpr,
):
    # 1. Current Kernel Instance inquiry
    pid_m = tl.program_id(axis=0)
    pid_n = tl.program_id(axis=1)


    # 2. Offsets
    """
    m : row
    N : column
    K : How many chunks will be processed at a time (I mean K).
    """
    offs_m = pid_m * BLOCK_M + tl.arange(0, BLOCK_M)
    offs_n = pid_n * BLOCK_N + tl.arange(0, BLOCK_N)
    offs_k = tl.arange(0, BLOCK_K)

    # 3. Calculate the A,B location pointer from the K block (MEMORY LOC)
    a_ptrs = A_ptr + offs_m[:, None] * stride_am + offs_k[None, :] * stride_ak
    b_ptrs = B_ptr + offs_k[:, None] * stride_bk + offs_n[None, :] * stride_bn

    # 4. FP32 Accum
```

```
    acc = tl.zeros((BLOCK_M, BLOCK_N), dtype=tl.float32)

    # 5. K axis -> BLOCK_K as unit LOOP
    for k in range(0, K, BLOCK_K):
        # ONLY VALID LOCATION (Last block can go over the K)
        k_mask = (k + offs_k) < K

        # Data Loading from the pointers
        a = tl.load(a_ptrs, mask=k_mask[None, :], other=0.0)
        b = tl.load(b_ptrs, mask=k_mask[:, None], other=0.0)

        # Accumulate
        """
            MATMUL
            a*b to (Block_M x Block_N) add to acc
        """
        acc += tl.dot(a, b)

        # POINTER MOVE
        """
            A -> Column
            B -> Row
        """
        a_ptrs += BLOCK_K * stride_ak
        b_ptrs += BLOCK_K * stride_bk

    # 6. Calculation SAVE

    # MEMORY LOCATION
    c_ptrs = C_ptr + offs_m[:, None] * stride_cm + offs_n[None, :] * stride_cn
    # Validation, Make sure it's before M and N
    mask_m = offs_m < M
    mask_n = offs_n < N
    # C = M x N
    tl.store(c_ptrs, acc, mask=mask_m[:, None] & mask_n[None, :])
```

C.2 Perofrmance

We benchmarked implementations using standard CUDA code.

- **Average Slowdown** : $1.05\times$
- **Small Batch (Less than 16)** : $1.013\times$
- **Mid batch (16 to 265)** : $1.095\times$
- **Large Batch (Biggern than 265)** : $1.051\times$

The Triton kernel maintains complete determinism, with only a few slowdowns compared to the standard PyTorch code.