



CONFERENCE

—NORTH AMERICA 2026—

Before the Loss Spike: Training Alignment for Precise PyTorch Debugging at Scale

Ziming Zhou
University of Michigan & ByteDance Seed

SPEED THROUGHPUT



Hidden bugs are expensive

Anthropic

A postmortem of three recent issues

16% of requests

Wrong context routing degraded Sonnet 4 responses at peak hours.

Official postmortem, Sep 2025

Meta

POSTED ON JULY 22, 2025 TO DATA INFRASTRUCTURE

How Meta keeps its AI hardware reliable

Weeks to months

Silent gradient corruption can waste training iterations before its effects become visible.

Engineering at Meta, Jul 2025

OpenAI

OPENAI, 2026 engineering

Core dump epidemiology: fixing an 18-year-old bug

Using population-level analysis to debug tricky crashes in our data infrastructure.

12+ crashes / day

A one-instruction race caused repeated production crashes in Rockset data infrastructure.

OpenAI Engineering, Jun 2026

“The ones that fix more of them (bugs) end up training better models.”

Jiayi Weng, OpenAI · WhynotTV Podcast #4 · 1:06:55–1:07:00 (English subtitles)

When can we fully “trust” an optimized implementation?

REFERENCE

PyTorch /
Megatron

vs.

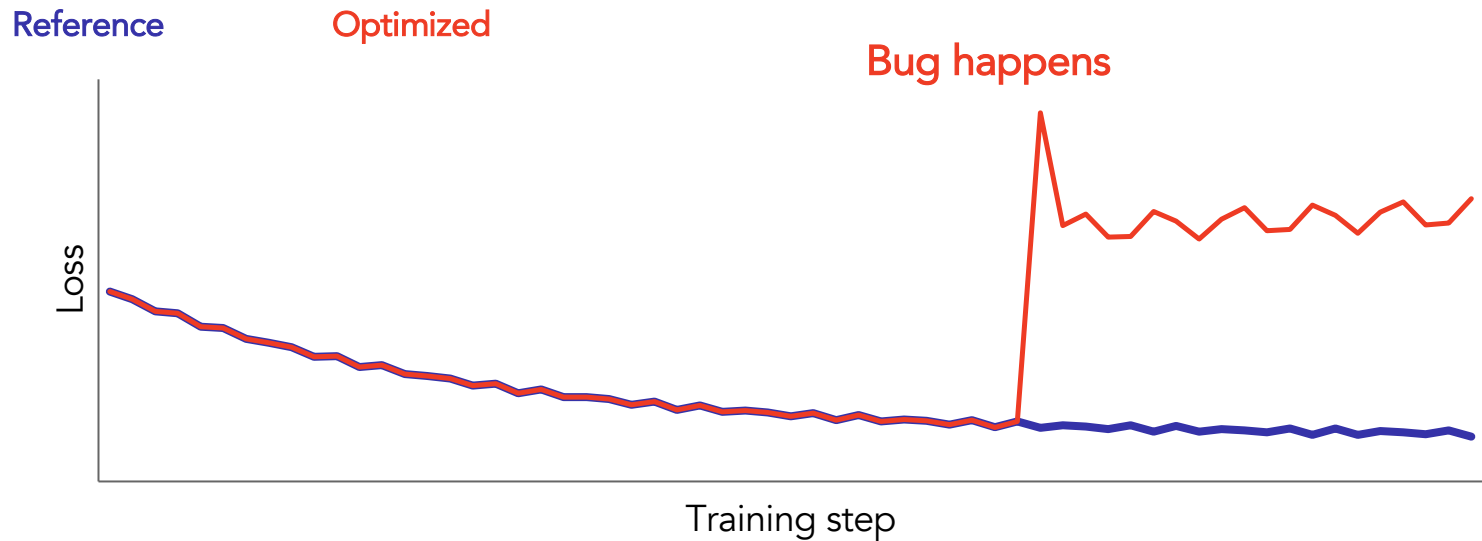
CANDIDATE

Optimized
implementation

Same inputs. Same initial weights. Same training setup.

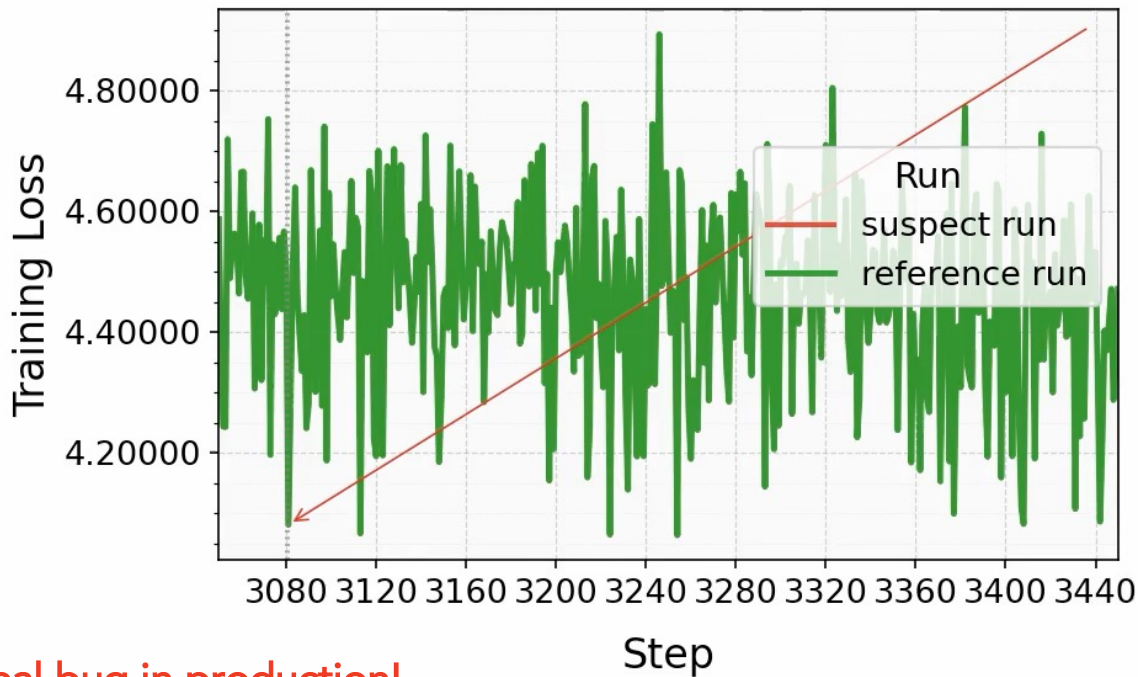
Do their loss curves agree?

What we expect: an obvious loss spike



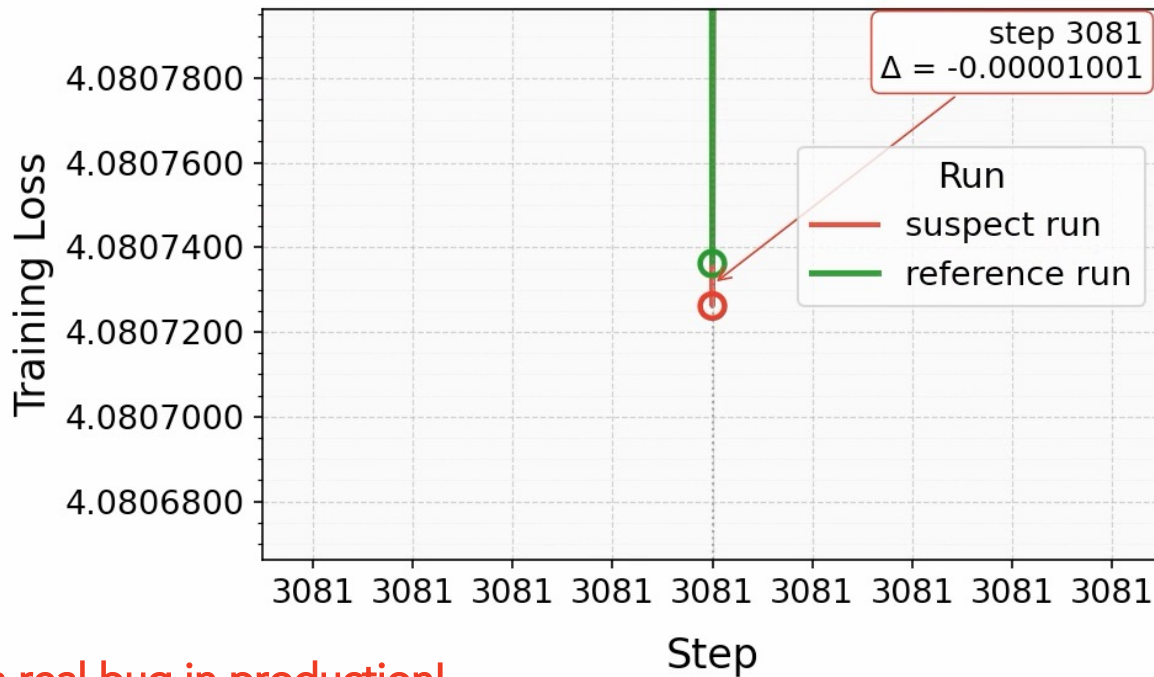
Illustration, not measured training data

In reality: the loss can look super healthy



From a real bug in production!

In reality: the loss can look super healthy



From a real bug in production!

Debugging → Guessing



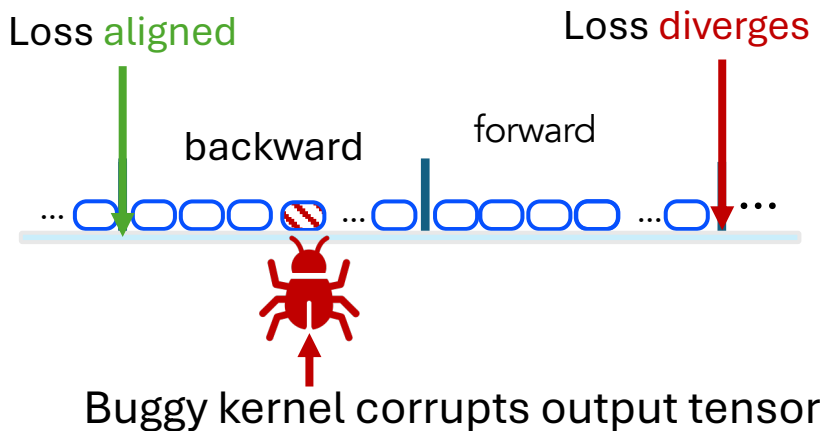
5 days

Manual debugging

- 50+ reruns
- Root cause not identified

Root cause:

A buggy kernel that causes a data race



This buggy kernel has been in PyTorch for more than 7 years!

Debugging → Guessing (Cont'd)

Seeing a loss difference is only the beginning

Noise Normal variation can hide a small error.

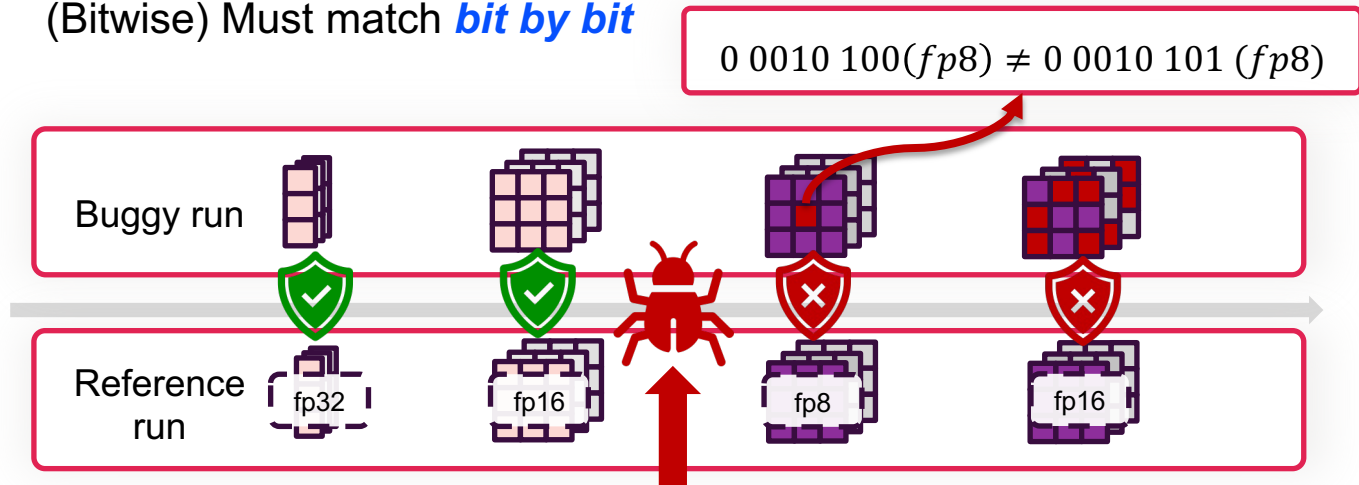
Propagation The visible symptom may appear long after the cause.

Scale Which step, layer, operator, or rank should we inspect?

Where did the two runs first diverge?

Training Alignment

- View each training run as a sequence of “points”
- Compare the tensors at each point
- (Bitwise) Must match *bit by bit*



The first mismatch becomes a **clean pivot for debugging!**

Bitwise reproducibility is gaining traction

Google

Exact replay from any checkpoint

[PaLM · 2022 · §5](#)

ByteDance

Bitwise checks for failure diagnosis

[ByteRobust · SOSP 2025 · §4.3](#)

Microsoft AI

Determinism as an infrastructure requirement

[MAI-Thinking-1 · 2026 · §2.8.2](#)

NVIDIA

Deterministic pretraining and checkpoint resume

[Megatron Core · NVIDIA Technical Blog · 2026](#)

Thinking Machines

Bitwise agreement between sampler and trainer

[Defeating Nondeterminism in LLM Inference · 2025](#)

Reported guarantees apply within each system's supported configuration.

Impact: OpGuard changes ByteDance's debugging workflow



Before

- Manually compare two runs
- Guess which subsystem to check



With OpGuard

Suspect run



Reference run

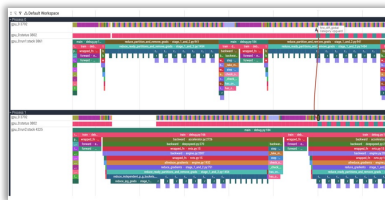


First mismatch

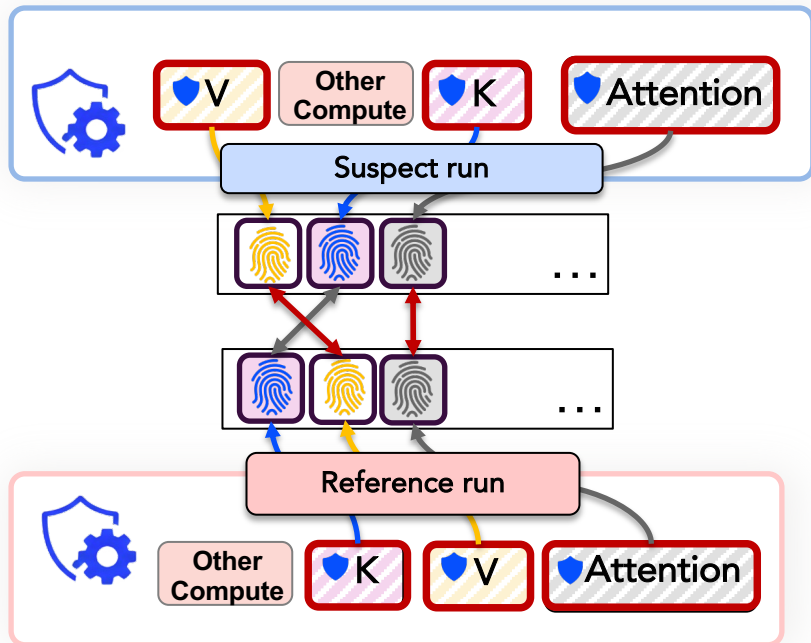


A reliable debugging pivot

Inspect in the visualizer



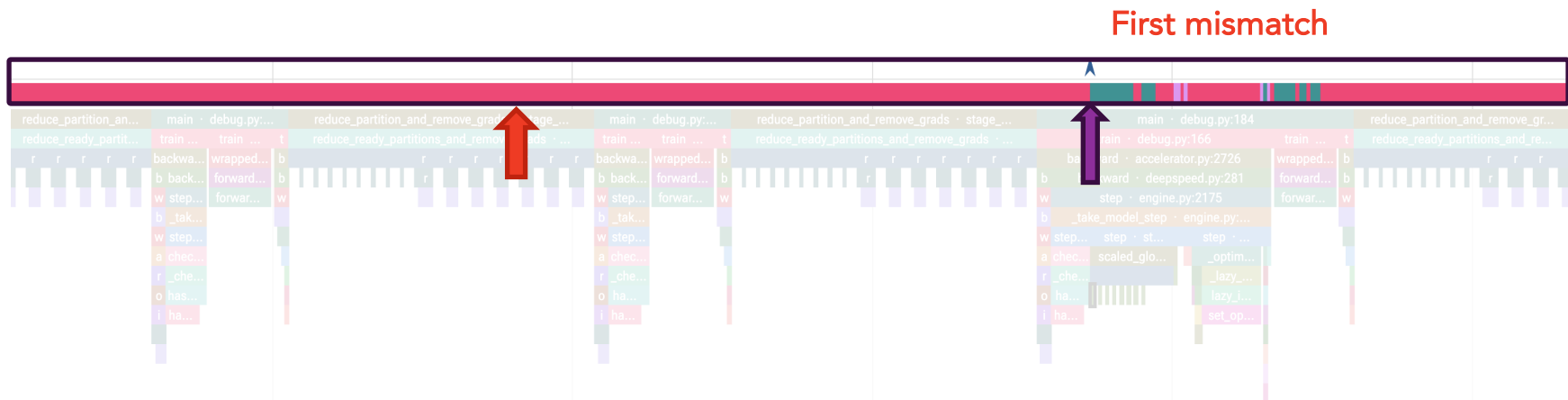
How OpGuard works



- 1 **Determinism control** 
- 2 **Preflight (~5 iterations)**
discover stable ops
- 3 **Guarded replay:**
fingerprint at op boundaries
- 4 **Offline alignment**

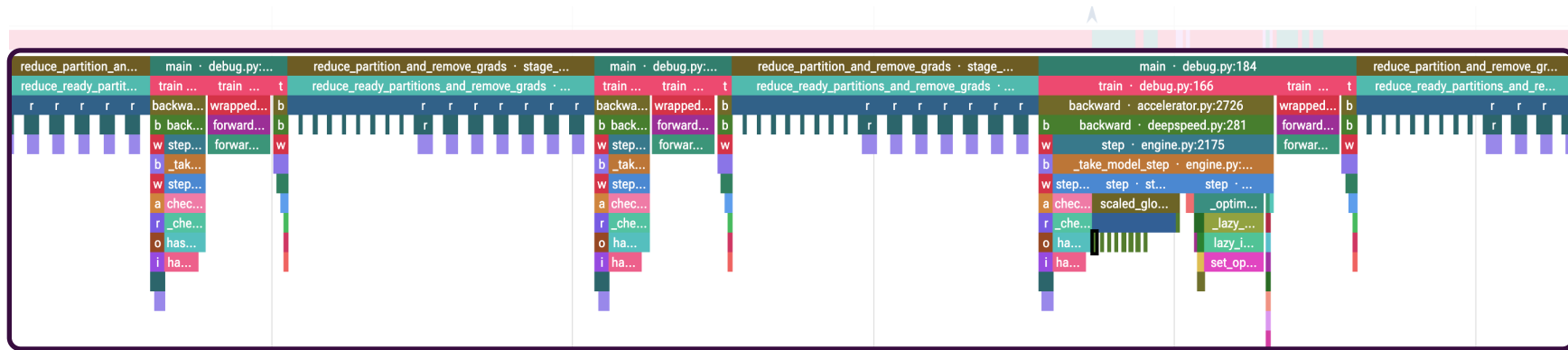
Visualizer: the first divergent operator

Operator sequence overview



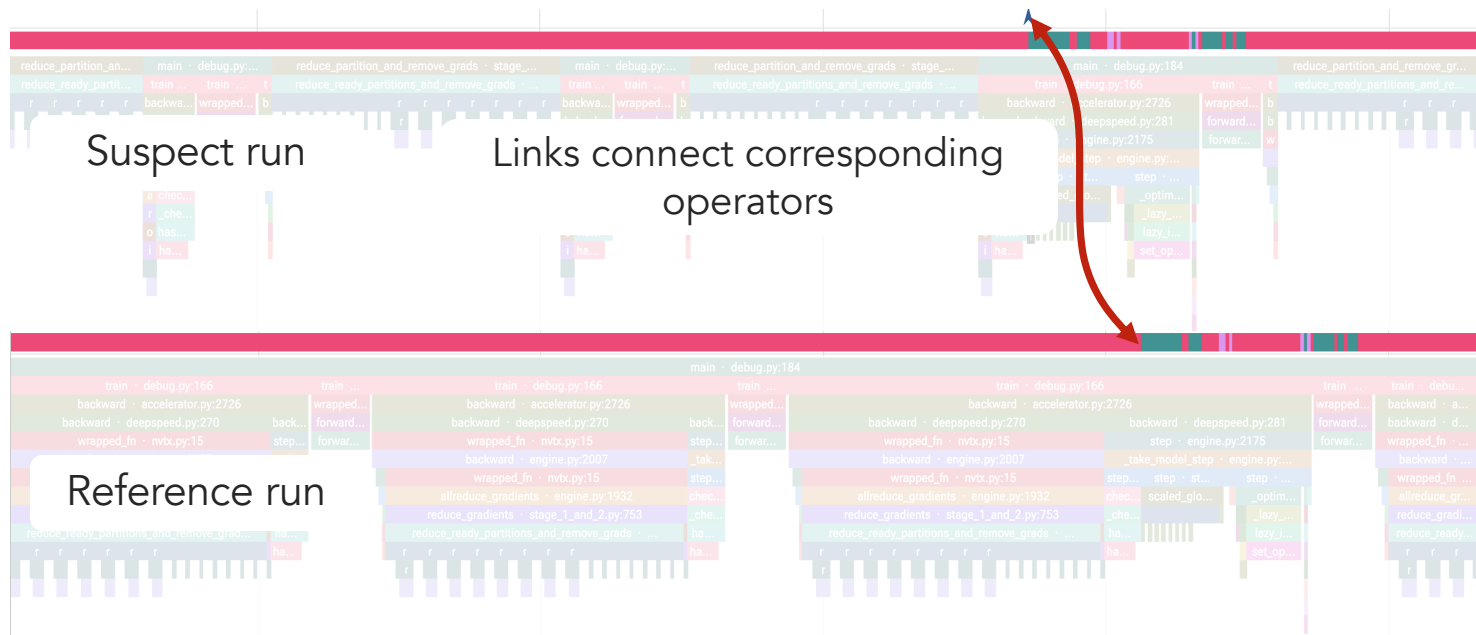
Red ops are bitwise equal; the first green segment marks the divergence point

Visualizer: each operator's call stack

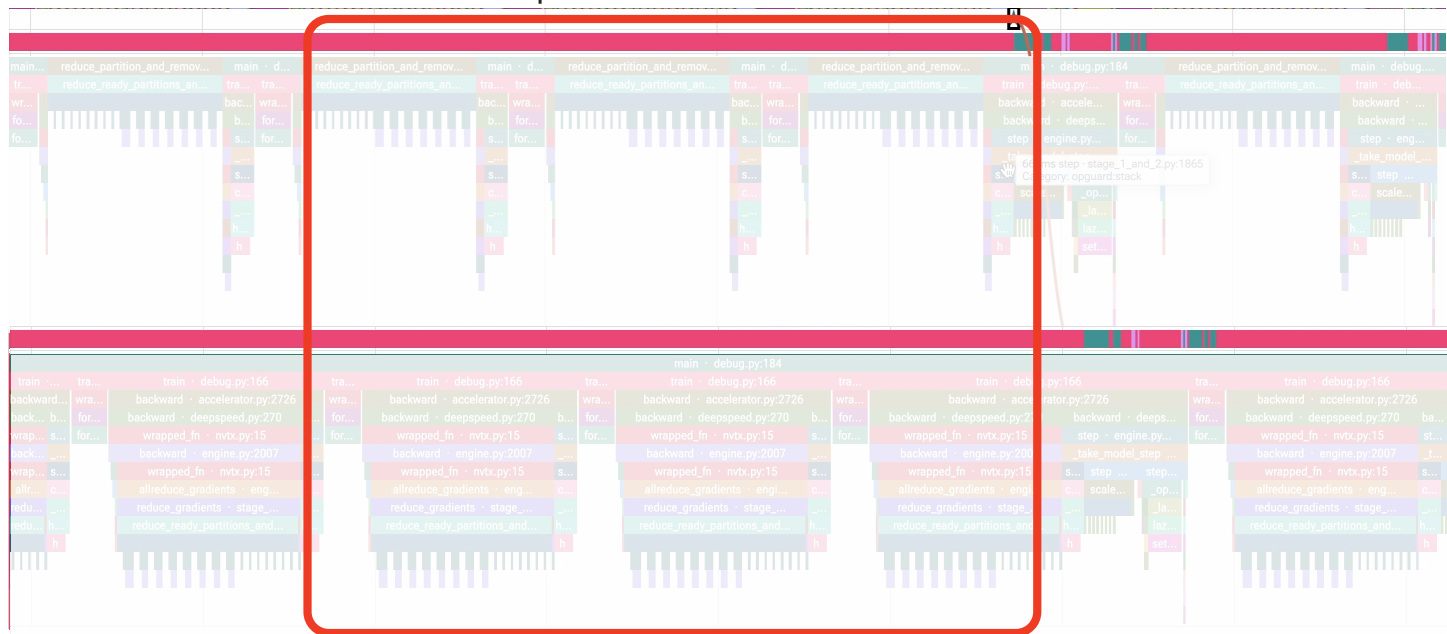


Every operator includes a captured call stack

Visualizer: corresponding operators across runs

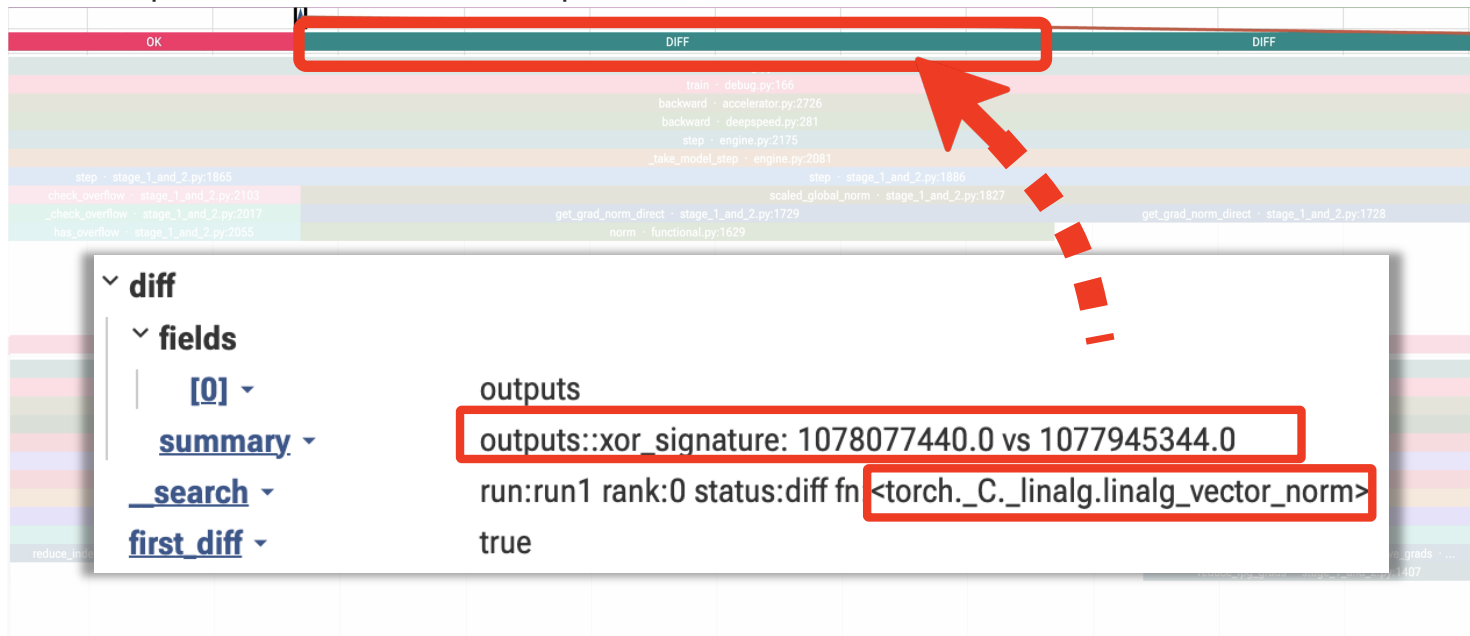


Visualizer: a closer look at the divergence



Visualizer: the differing fingerprints

Output fingerprints differ at this operator

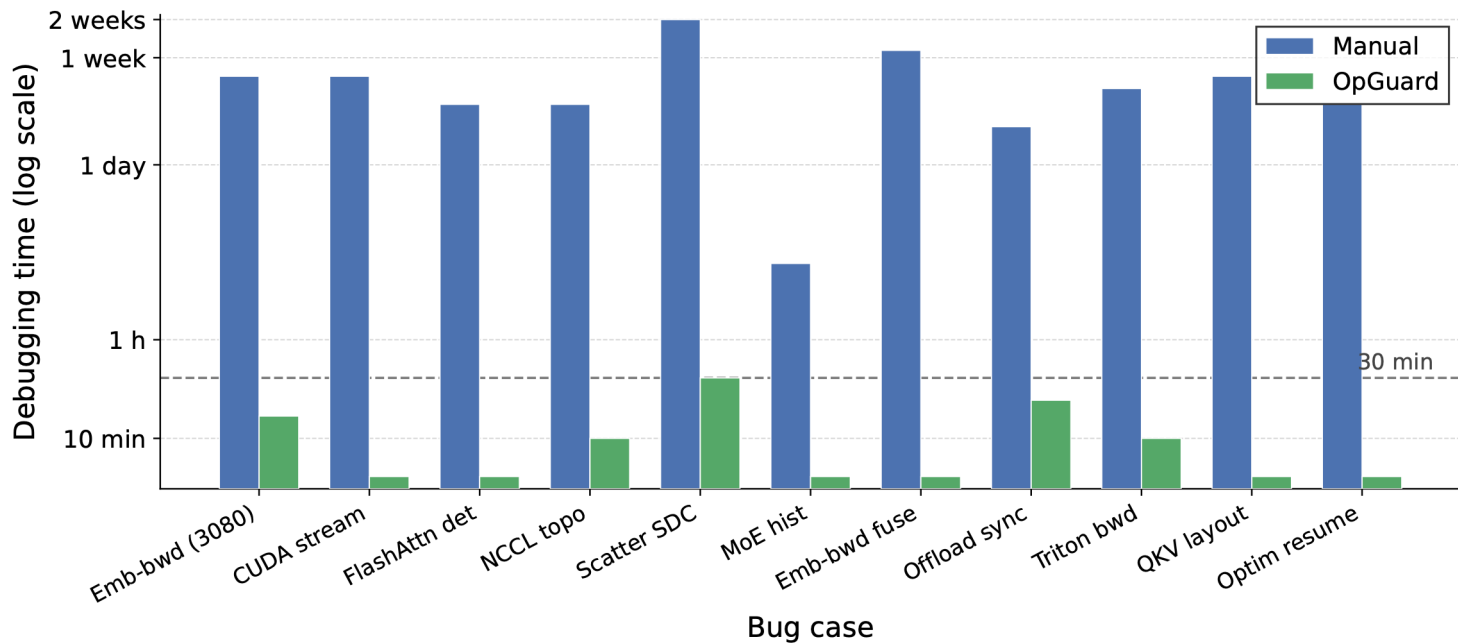


Visualizer: the exact source location

The call stack points to the source line

```
1625 | else:
1626 |     # NB. p should be Union[str, number], not Optional!
1627 |     _p = 2.0 if p is None else p
1628 |     if out is None:
1629 |         return opguard_wrapped__linalg_vector_norm(1626)(torch.linalg.vector_norm,
input, _p, _dim, keepdim, dtype=dtype) <<< TARGET LINE
1630 |     else:
1631 |         return opguard_wrapped__linalg_vector_norm(1628)(torch.linalg.vector_norm, input,
_p, _dim, keepdim, dtype=dtype, out=out)
1632 |
1633 |     ndim = input.dim()
1634 |
```

OpGuard reduces triage time from **days** to **minutes**



Agentic Era: OpGuard as a tool call

FIRST DIFF

vector_norm

DeepSpeed · gradient storage

```
buf = self.ipg_buffer[self.ipg_index]
grad_view = buf.narrow(...)
grad_view.copy_(grad_reduc.view(-1))
grad_reduc.data = grad_view.data
```

AGENT FINDING

**A reusable
communication
buffer.**

Can another stream
still be using it?

Root cause revealed using Groq

Two streams race to access the same memory.

Reduction
stream

Still reading buffer A

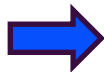
Default
stream

At the same time

Overwrites buffer A

Unexpected deployment lesson: silent data corruption (SDC)

Designed for debugging
software bugs



Found **hardware
corruption** in production

These machines had passed standard health checks and stress tests



suspect vs. reference
run



Op-level comparison



First mismatch
(same input, different output)



**Detected
21+**
distinct SDC machines

OpGuard supports **major** production-level training stacks

-  CUDA graphs
-  torch.compile
-  Different parallel configs (PP/CP/DP/TP/EP)
- ...



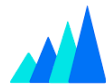
Megatron LM



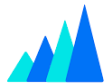
DeepSpeed



PyTorch
Compiler



veScale



veRL



PyTorch FSDP



GPT-NeoX



Transformers

OpGuard is now open source!!!

- What is coming:
 - Alignment with numerical tolerance
 - A more seamless agentic debugging experience
 - Broader alignment support across parallelism modes



Like OpGuard? Give us a ★





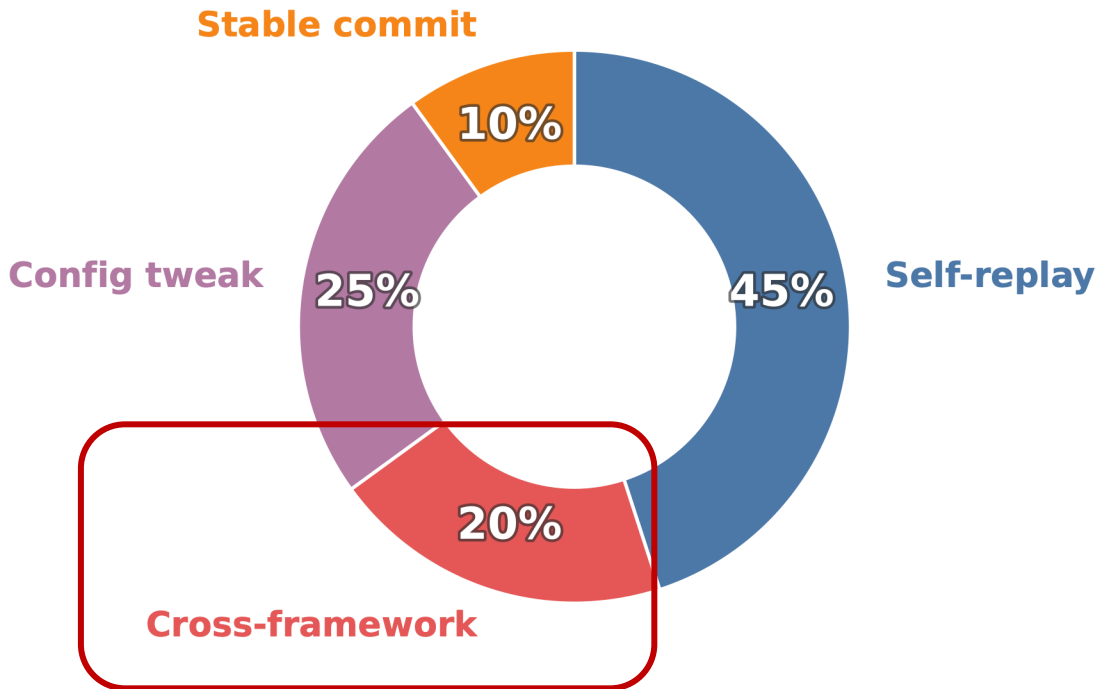
Backup Slides

Alignment enables broader comparisons

Compare across
stacks, configs,
commits, ...

As long as they
implement the
same computation

How are reference runs selected?



OpGuard diagnoses **diverse** bugs

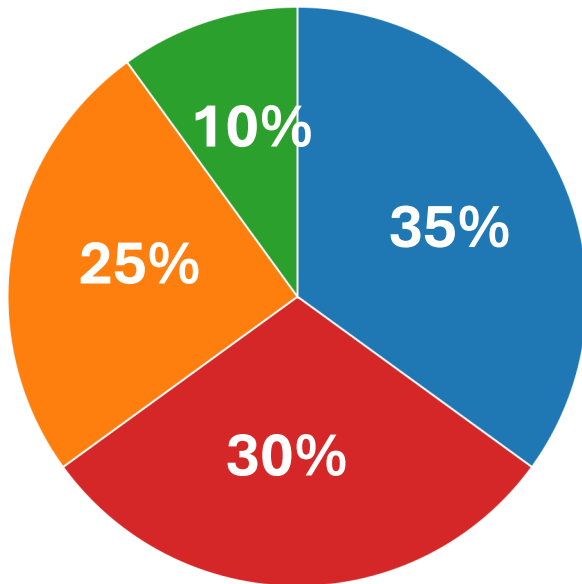
OpGuard successfully diagnoses **20 production cases** with diverse root causes

 Wrong model implementation / logic

 Race conditions

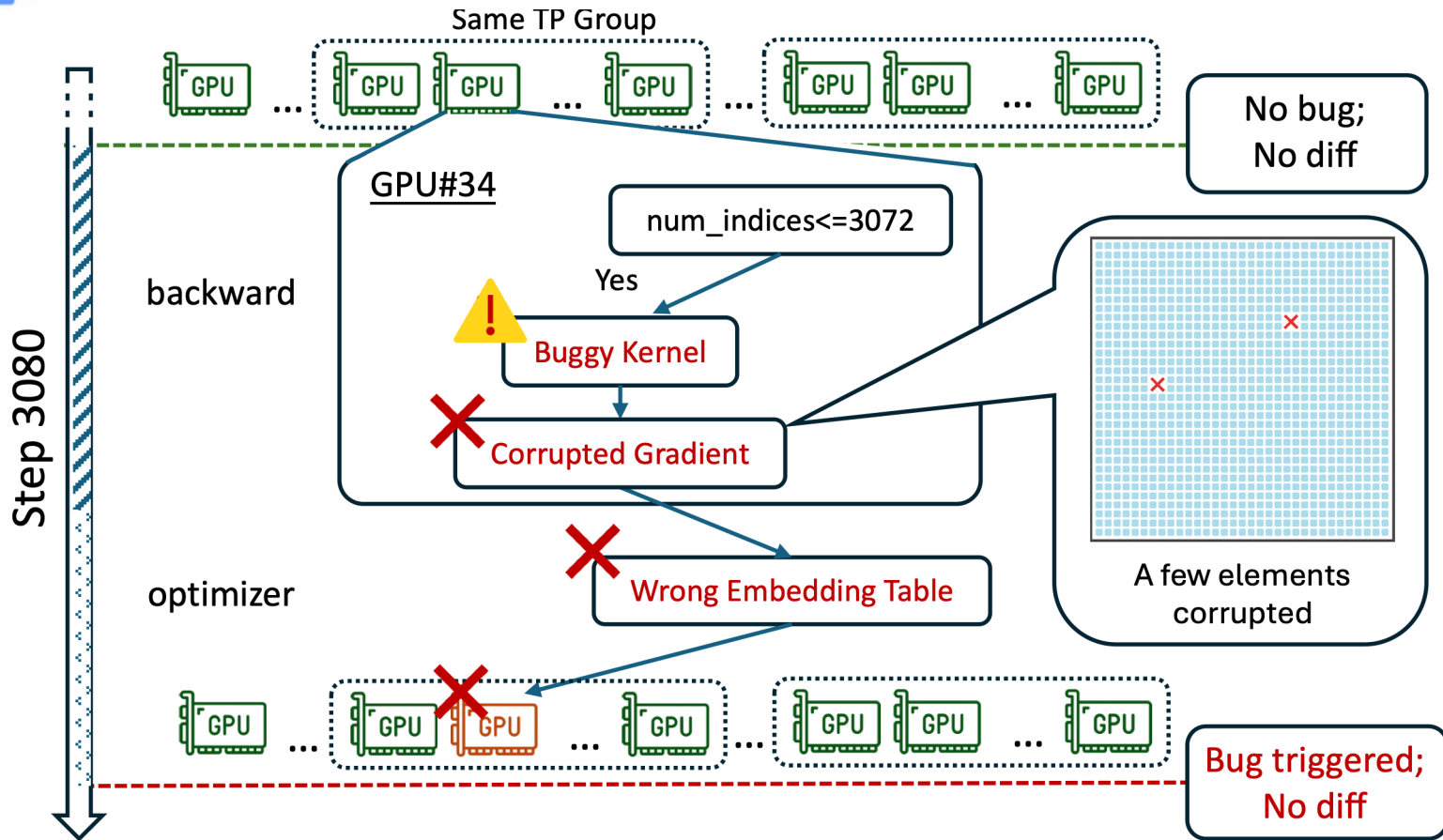
 Unexpected numerical drift

 Hardware faults

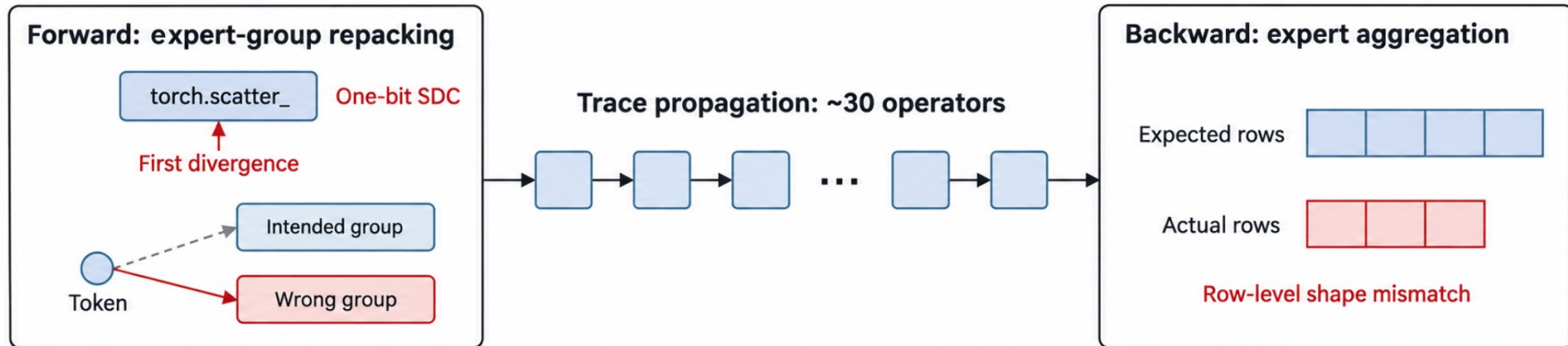


OpGuard also diagnoses 8 longstanding GitHub issues

Case Study 1 (Race Conditions)



Case Study 2 (Hardware Faults)



Beyond the first divergence, OpGuard traces how a forward-pass corruption becomes a backward-pass shape mismatch.

False positives

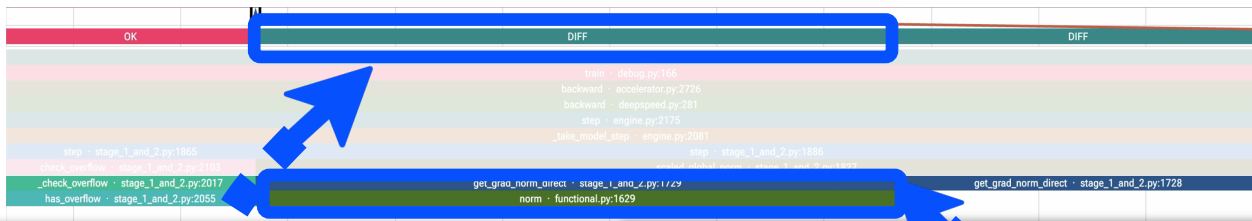


False positives: 2-10 during initial adaptation

- With the visualizer, 2-3 min to confirm noise
- Once marked, they never appeared again



Inspect concrete diff fields and source code in the visualizer



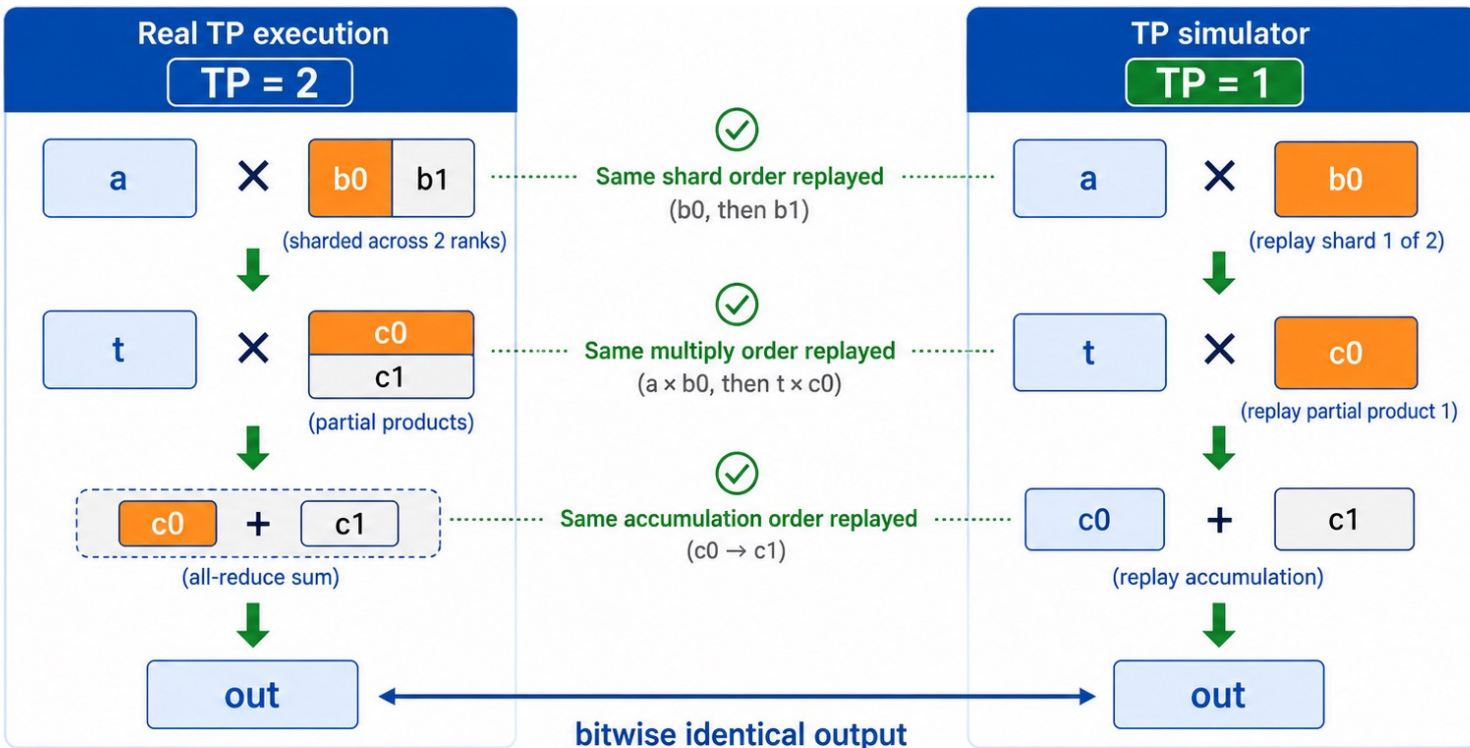
diff
fields
[0]
summary
search
first diff

outputs
outputs::xor signature: 1078077440.0 vs 1077945344.0
run:run1 rank:0 status:diff fn torch._C._linalg.linalg_vector_norm
true

```
1625 | else:  
1626 |     # NB. p should be Union[str, number], not Optional!  
1627 |     _p = 2.0 if p is None else  
1628 |     if out is None:  
>>> 1629 |         return opguard_wrapped__linalg_vector_norm(1626)(torch.linalg.vector_norm,  
input, _p, _dim, keepdim, dtype=dtype) <<< TARGET LINE  
1630 |     else:  
1631 |         return opguard_wrapped__linalg_vector_norm(1628)(torch.linalg.vector_norm, input,  
_p, _dim, keepdim, dtype=dtype, out=out)  
1632 |  
1633 |     ndim = input.dim()  
1634 |
```

Bitwise alignment across parallelism configurations

Example: TP simulator (bitwise alignment across tensor parallelism settings)



Aligned kernel coverage across workloads

