OpGuard

Bitwise Alignment for Precise and General Debugging of Production LLM Training

Ziming Zhou, Yinjie Zhao, Hang Zhu, Wenxiao Wang, Zhihao Bai, Yun Zhang,

Shuguang Wang, Haibin Lin, Peng Huang

ByteDance | Seed

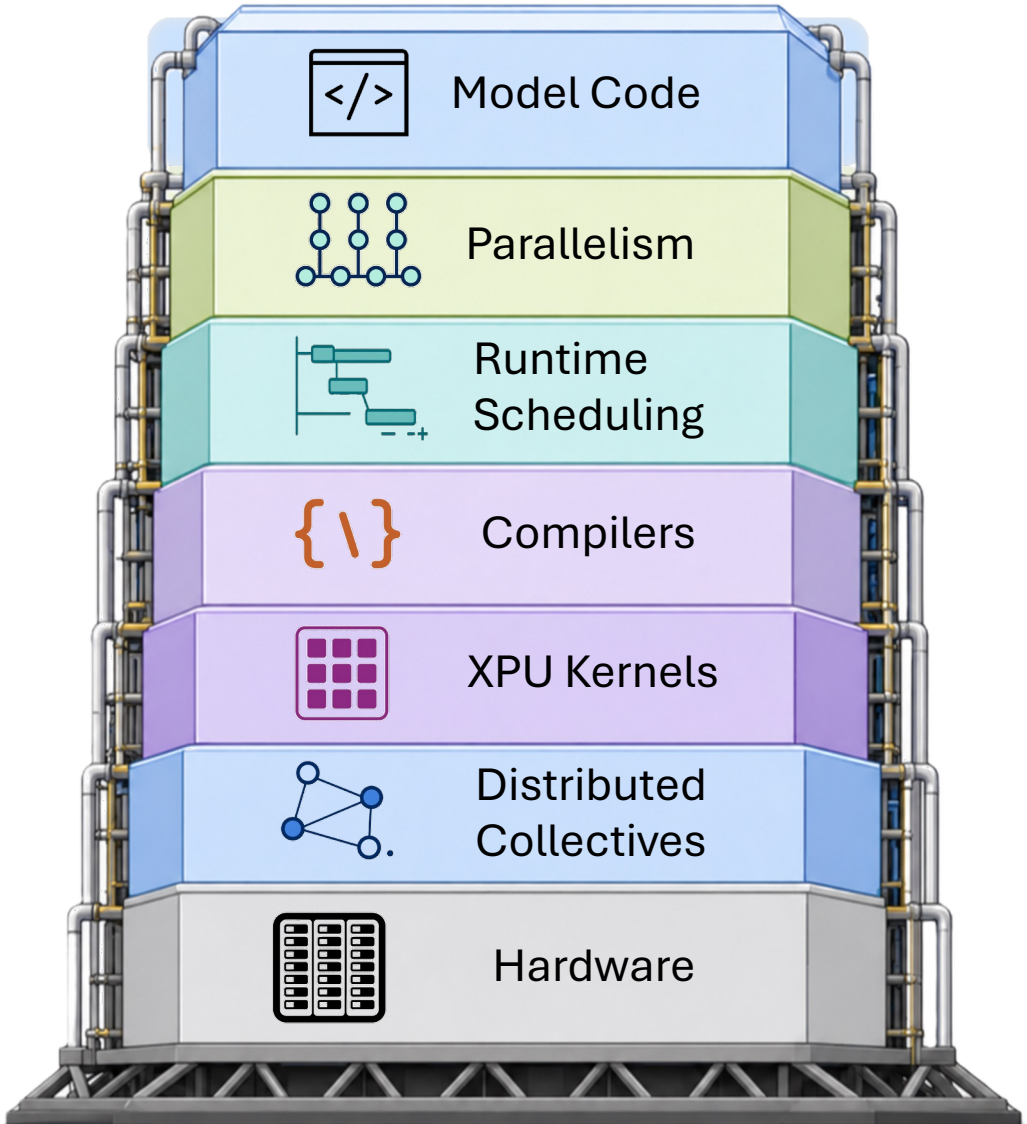
OrderLab

UNIVERSITY OF MICHIGAN

Production LLM training is error-prone

1. **Complex stack with extensive optimizations**
2. **Rapidly evolving**
3. **Running on 1000+ GPUs for long time**

➡ Bugs are **inevitable**!



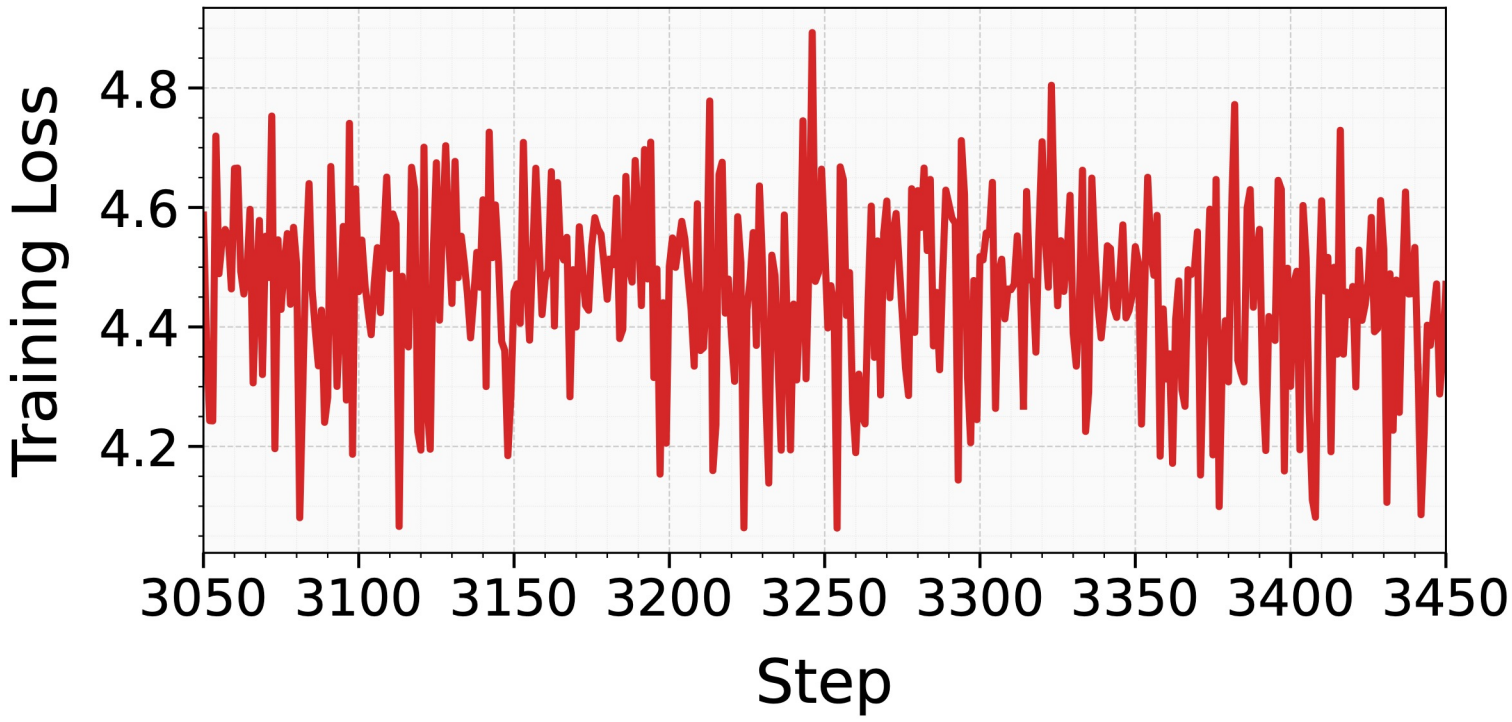
Debugging is extremely difficult

Engineers monitor high-level training signals (e.g. loss curves)

- Where is the bug? (little clue)

  Compilers, model code, optimizer, kernels, etc.?

- Even unclear whether it is buggy? (could be noises)



This figure is from a real production training run

Strategy in practice: compare with another run

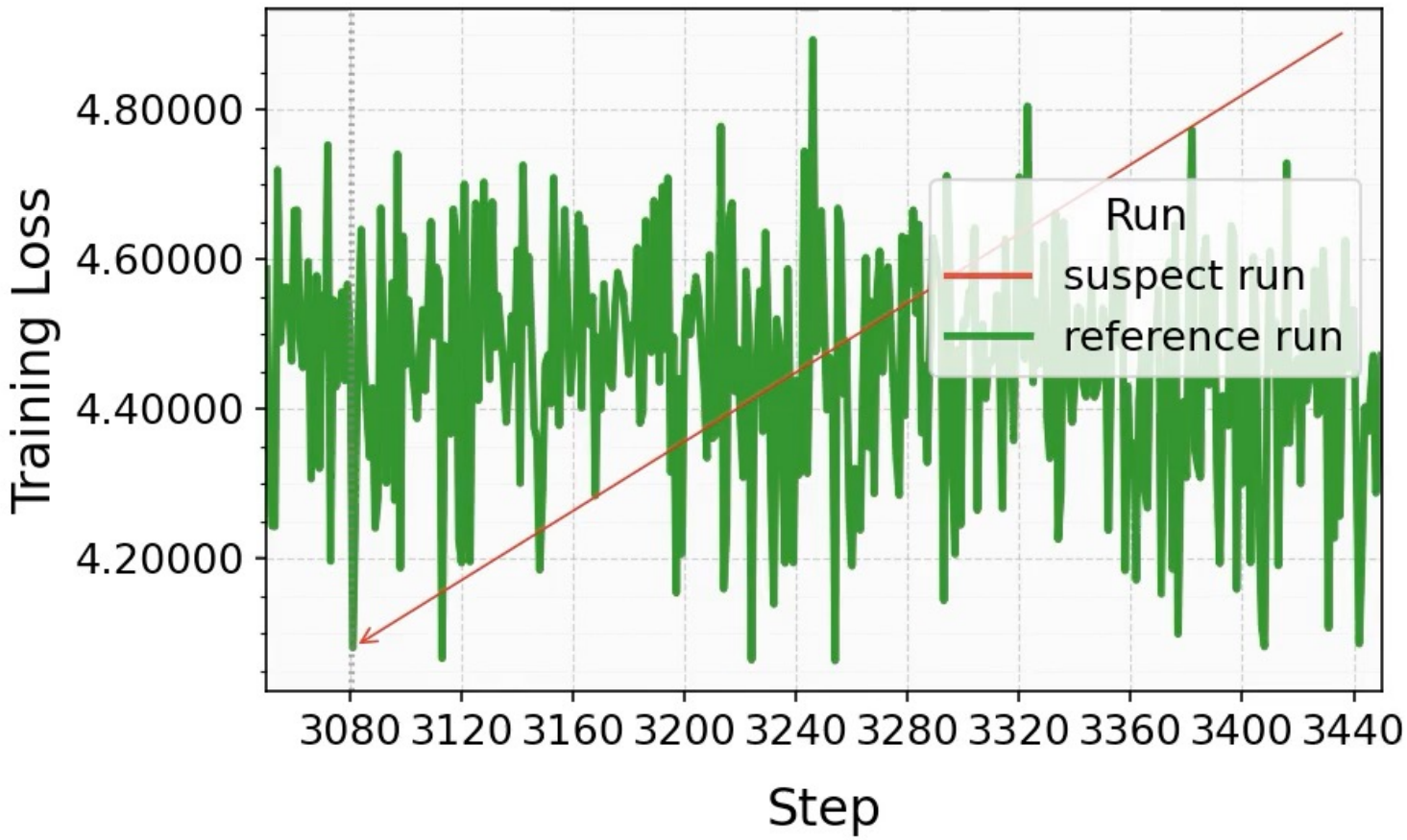
- Well studied in conventional software
  - Delta debugging techniques (e.g. Zeller, ESEC/FSE ’99)
  - Establish a baseline for correctness and narrow scope

- Sources of reference runs
  - Same training job with different configuration
  - Swapping suspected kernels with equivalent kernels
  - An older version of the training frameworks

- Compare the two runs’ training curves

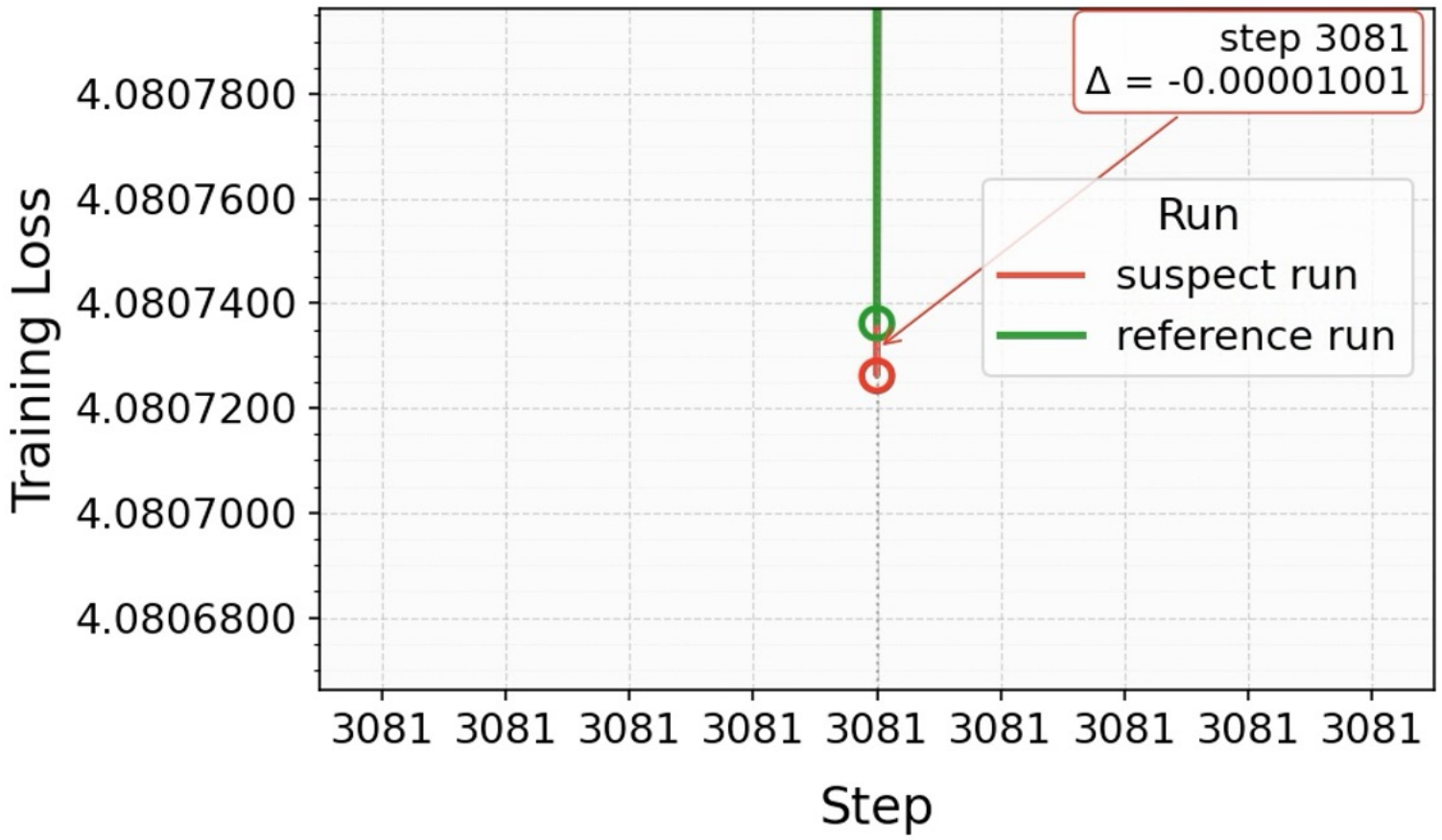
Debugging remains difficult

From an earlier production incident



Debugging remains difficult

From an earlier production incident



If you don't see the red curve, that's exactly because the differences are tiny!

Debugging remains difficult

5 days

Manual debugging

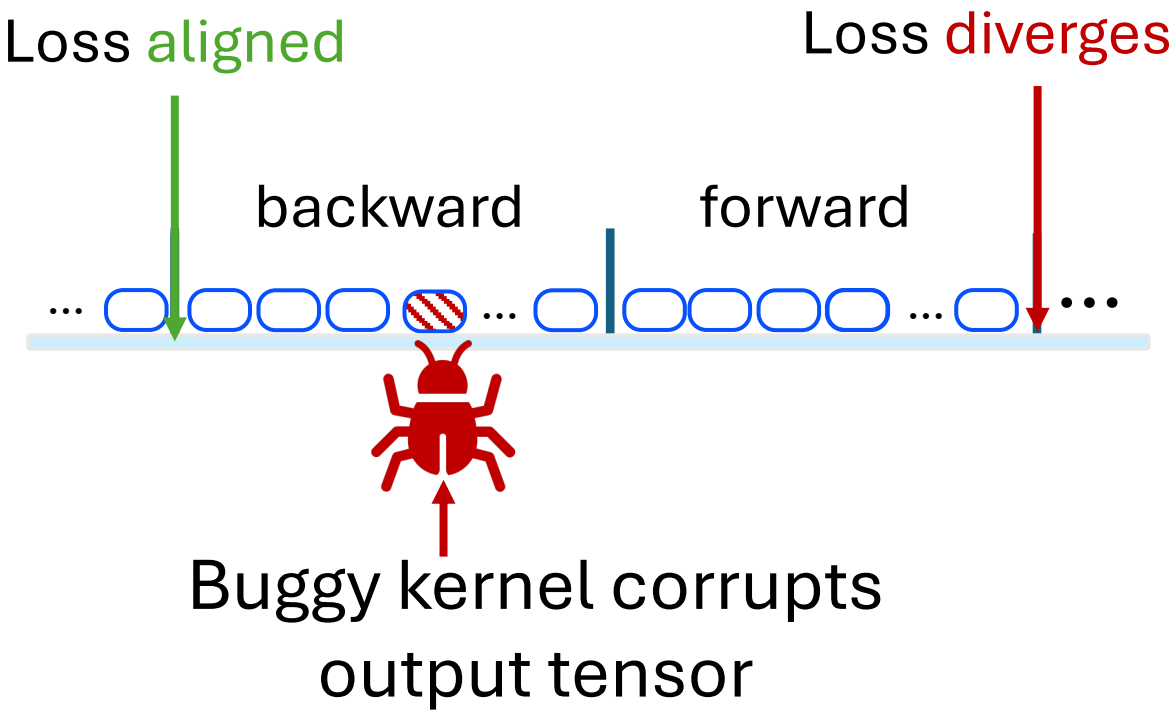
- 50+ reruns
- Root cause not identified

Problems:

- Expensive (many reruns)
- Noisy (large suspect region)
- Delayed symptom (root cause is many operations in the past)

Root cause:

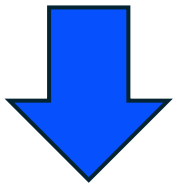
A buggy kernel that causes data race



Key insights

Composite signals make comparisons ambiguous

- Mix effects from many operations
- Subject to numerical noises

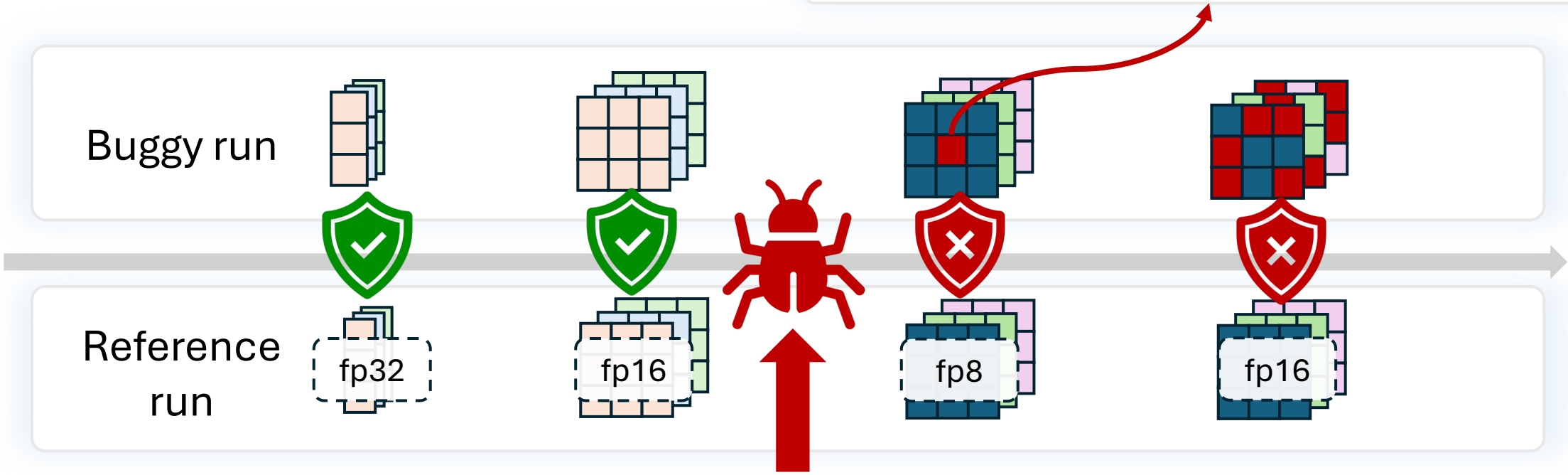


Effective debugging of LLM training requires precise, unambiguous comparisons

Proposed abstraction: bitwise alignment

- Given two training runs, view them as a sequence of “points”
- Compare the tensors at each point
- Must match ***bit by bit***

$$0\ 0010\ 100(fp8) \neq 0\ 0010\ 101\ (fp8)$$



First mismatch point becomes a **clean pivot for debugging**!

Bitwise alignment seems impractical

1 Many sources of nondeterminism

Even for the same training job, running it twice, the results may not match exactly

2 Make the training stack fully deterministic?

- Difficult, if not impossible
- Extensive changes
- Severely slow down training

3 Bit by bit comparisons is too slow at production scale

Contributions

1. Formalize bitwise alignment as a new correctness oracle
2. Design OpGuard to achieve practical bitwise alignment
3. Deploy OpGuard in ByteDance's production training stack

Choose the proper granularity for bitwise alignment

❌ **Each Python operator** → costly / noisy

❌ **Each training step** → not precise

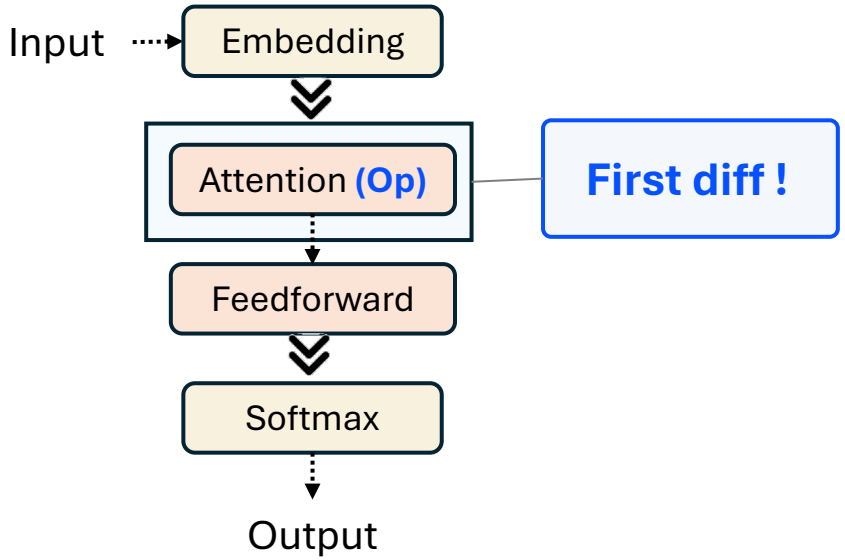
Model-level operation (Op)

Consumes tensors and produces tensors

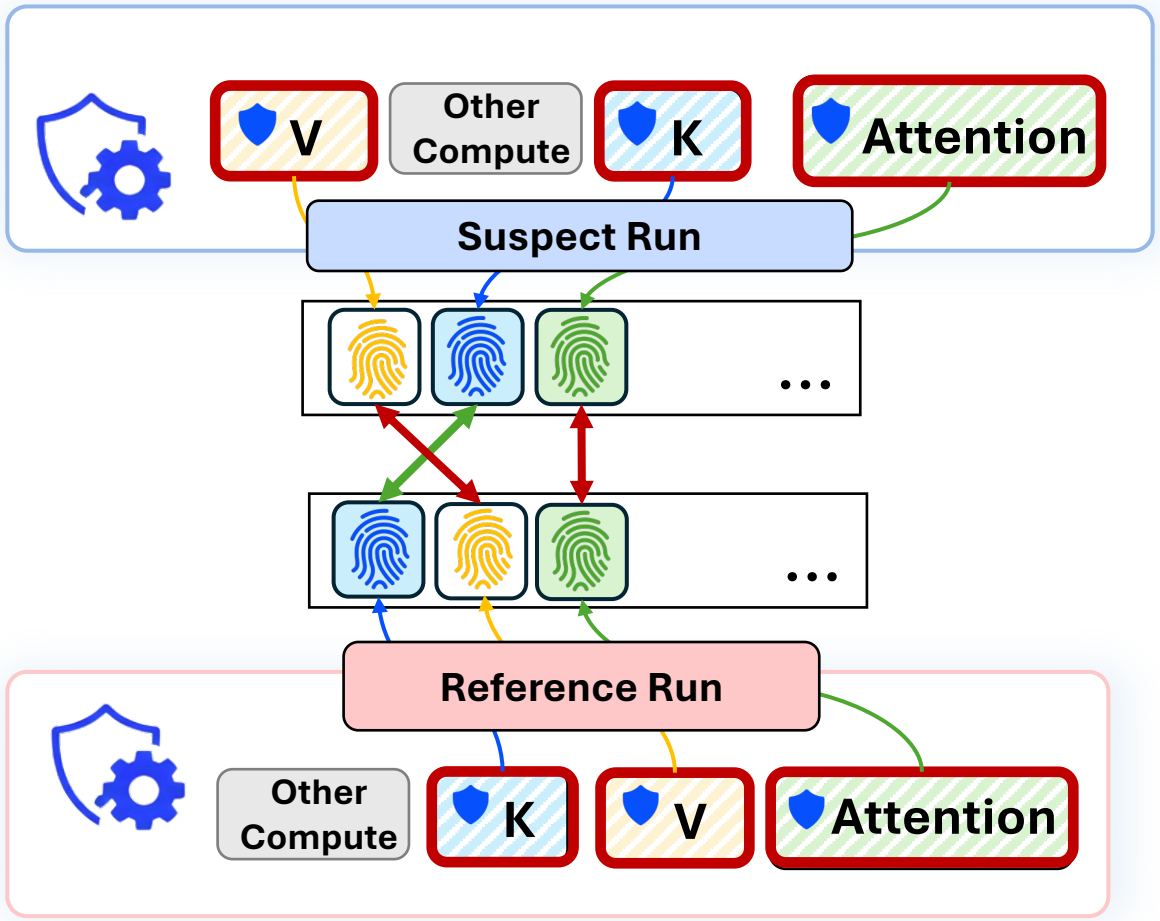
Example: q, k, v → Attention → attention score

Why?

- Far fewer boundaries to compare
- Each Op is a meaningful computation
- The first differing Op localizes the divergence



OpGuard workflow



1 Determinism control

2 Preflight: (~ 5 iters) discover stable Op

3 Guarded replay: fingerprint at Op boundary

4 Offline alignment

OpGuard aligns diverse runtime executions

Non-determinism is everywhere

Sources of non-determinism

- GPU scheduling
- Communication order
- Kernel choices
- Hardware timing
- Randomness
- …

Enforcing full determinism

- Too expensive
- May hide bugs

  (e.g., race condition under a specific schedule)

OpGuard

Control: numerically relevant choices

Left visible: true residual divergence

Example: communication order matters

Different reduction orders → different bits

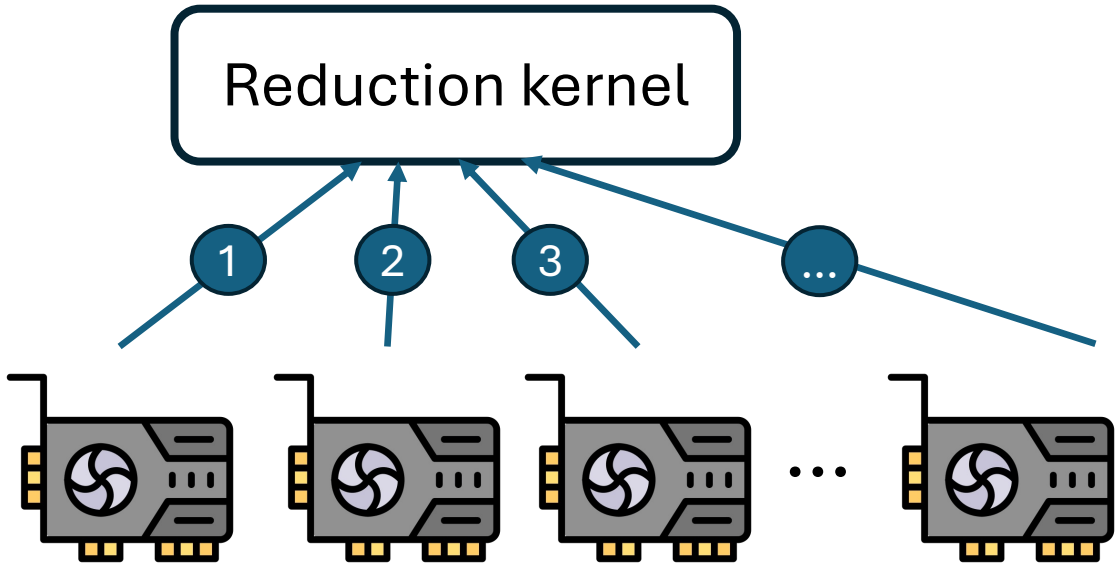
(1e10 + -1e10) + 1 = 1
1e10 + (-1e10 + 1) = 0

Controlled:

- Cross-GPU reduction order
- Collective algorithm / protocol

Not forced:

- Local GPU scheduling
- Hardware timing

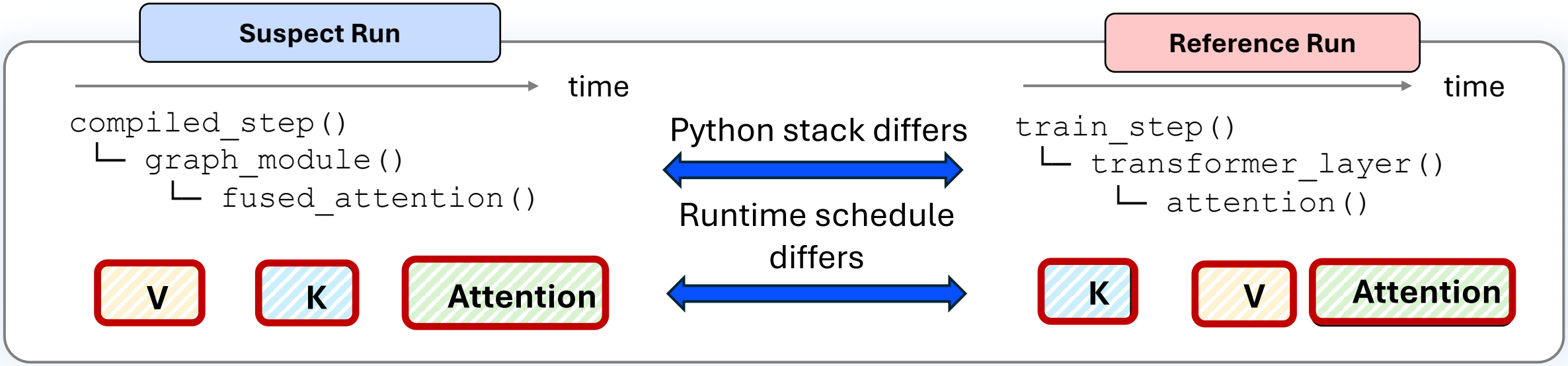


Control major nondeterminism; treat the rest as bug signals

(Refer to our paper for more detail)

Align Ops without freezing the training stacks

Same computation, different traces

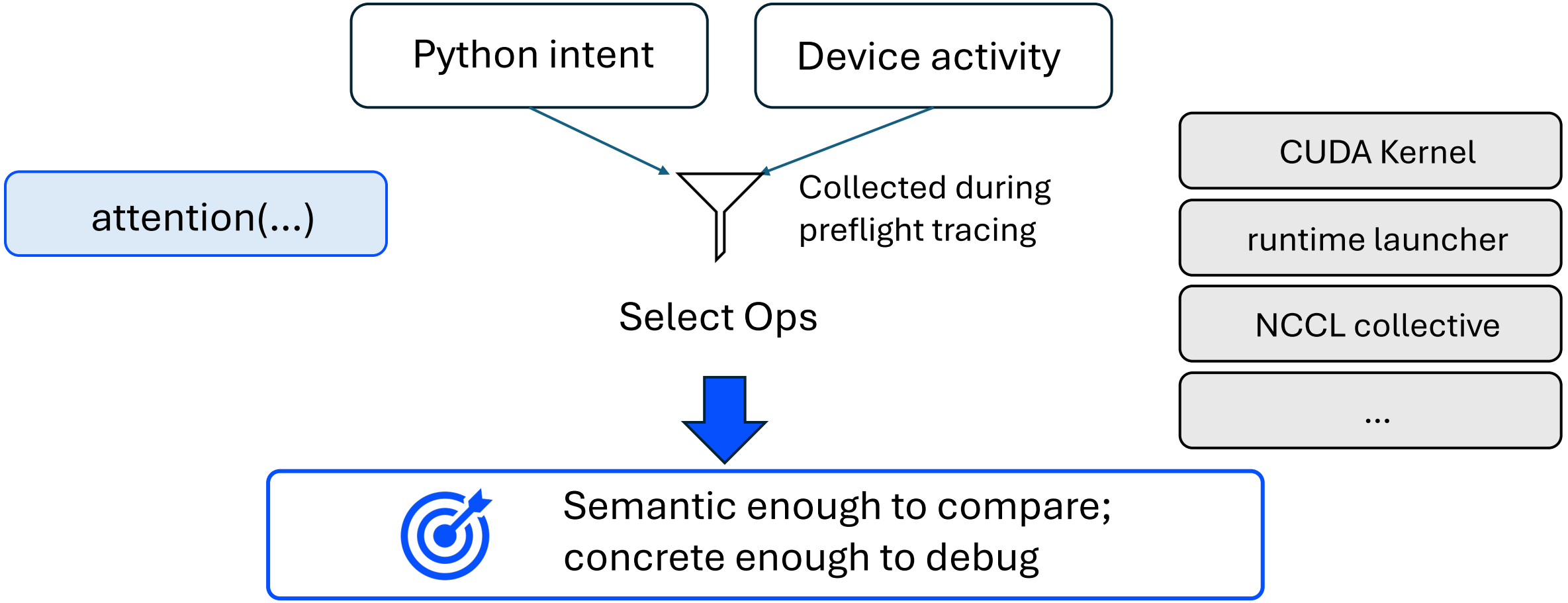


OpGuard achieves robust alignment by:

- **Step 1: find stable Op boundaries**
- **Step 2: match Ops across runtime schedules**

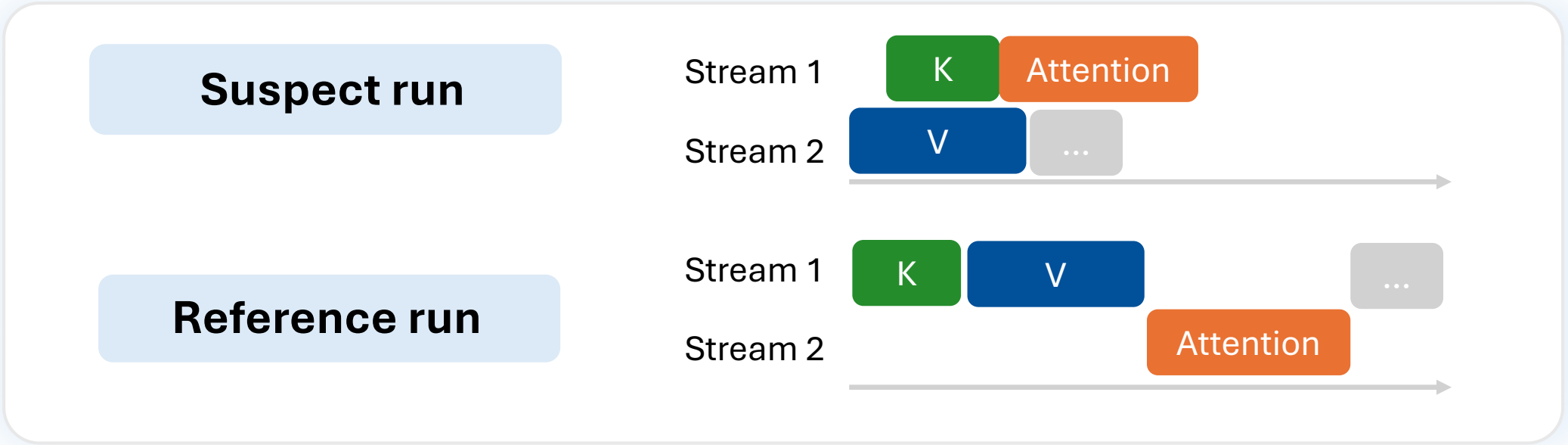
Find stable Op boundaries

One model op may trigger many low-level events:



(Refer to our paper for more detail)

Match Ops across runtime schedules



- Position-based matching is brittle

  Optimizations reorder / overlap computation

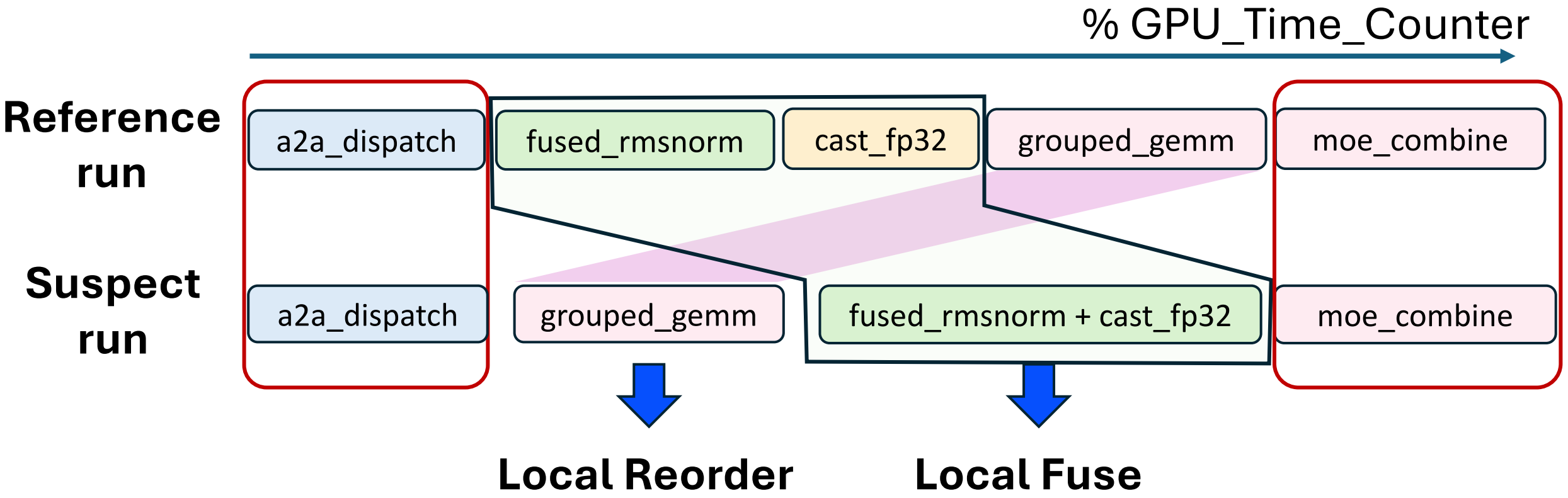
- Exhaustive matching is too expensive

  Hundreds of semantic ops per forward pass

Schedule-tolerant alignment

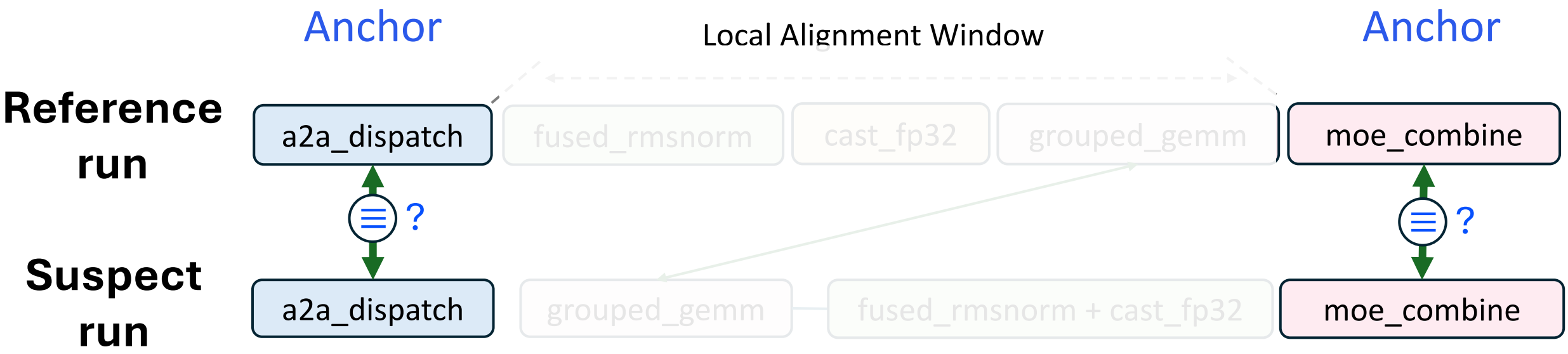
Match Ops across runtime schedules

Ops may reorder or fuse within a region—but not across major boundaries



Match Ops across runtime schedules

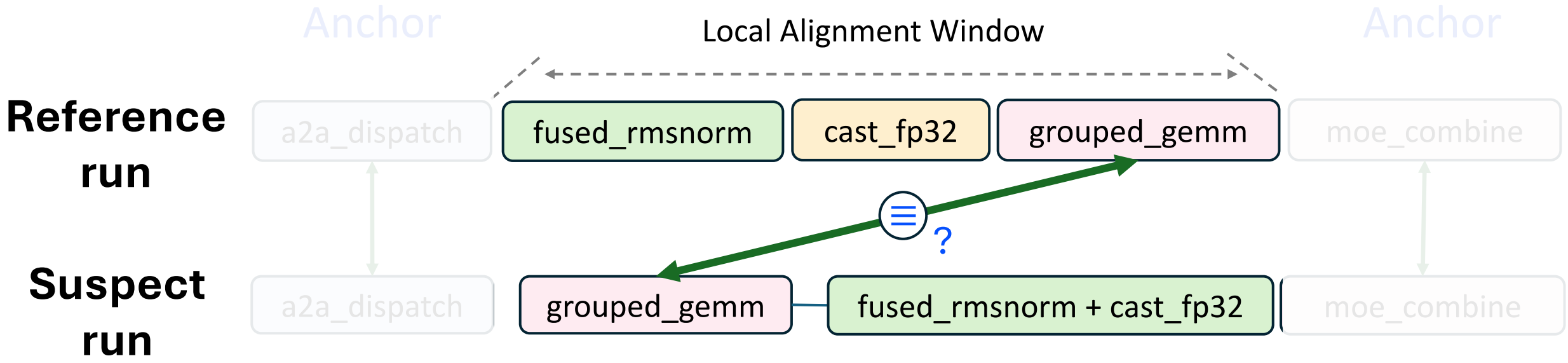
Discover anchor ops



Anchor recovers global structure of execution

Match Ops across runtime schedules

Align inside anchor window



Dynamic Programming tolerates schedule changes from real optimizations

OpGuard is deployed in ByteDance's production training environment and actively used by 15+ teams

Evaluation

1. Does bitwise alignment localize diverse real bugs?
2. How general is the primitive across stacks / reference choices?
3. What are the lessons from deployment?
4. How much overhead does OpGuard introduce?

OpGuard changes debugging workflow

Before

- Manually compare two runs
- Guess which subsystem to check

With OpGuard

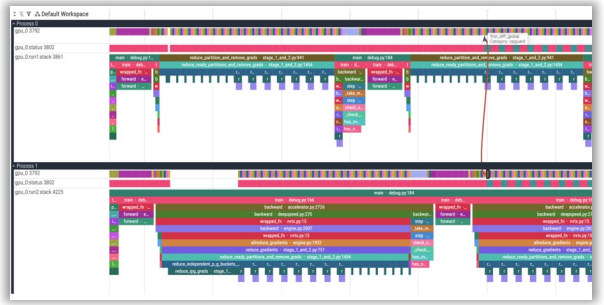
Suspect run

Reference Run

First mismatch

A reliable debugging pivot

Inspect with visualizer

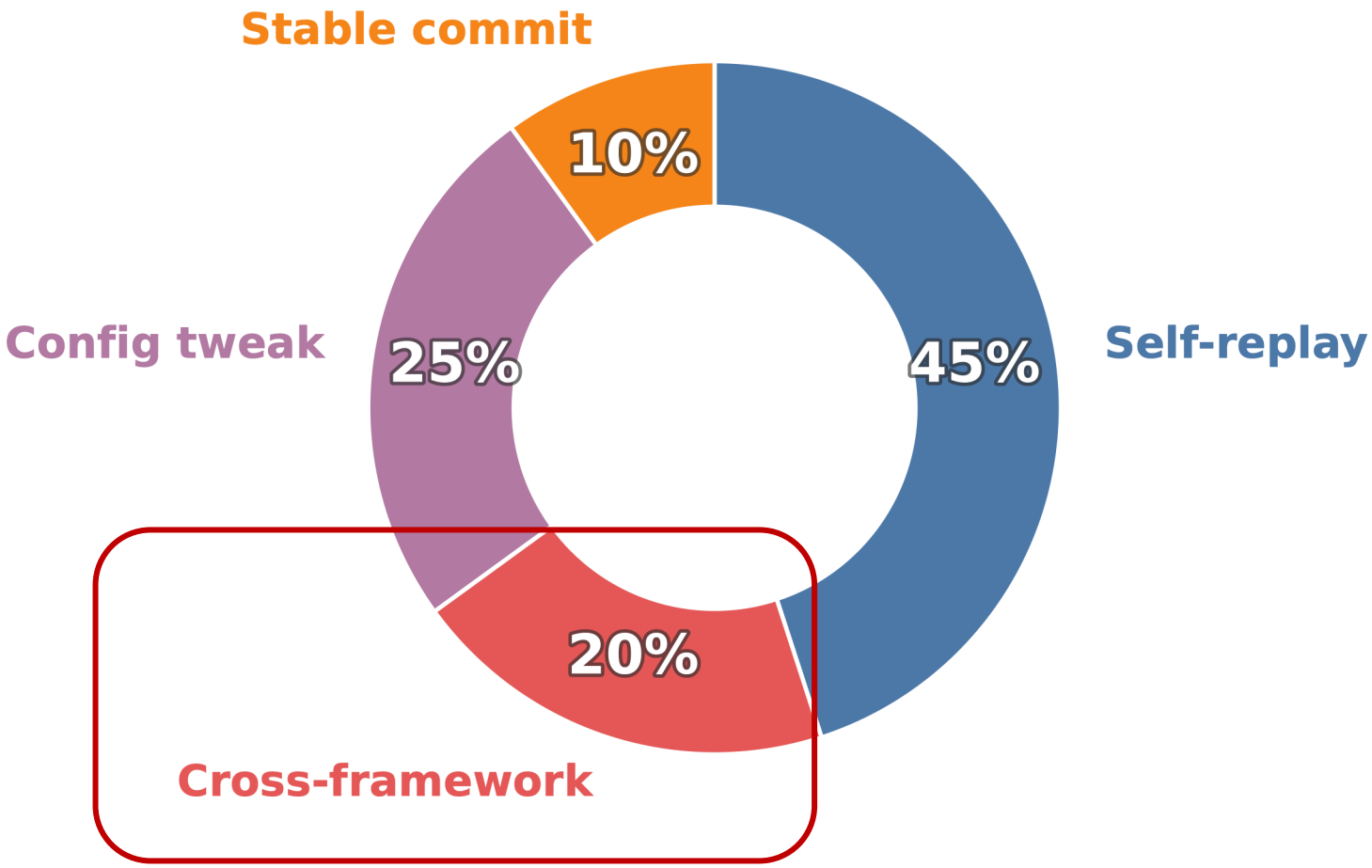


Bitwise alignment enables broader comparisons

Compare across stacks, configs, commits, …

As long as they implement the same computation

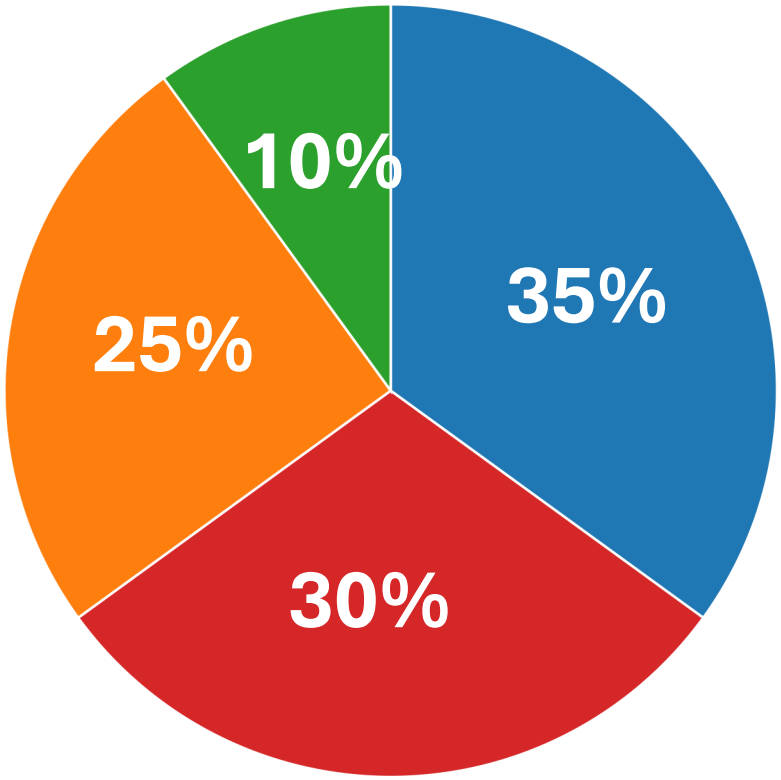
Where are reference runs selected?



OpGuard diagnoses diverse bugs

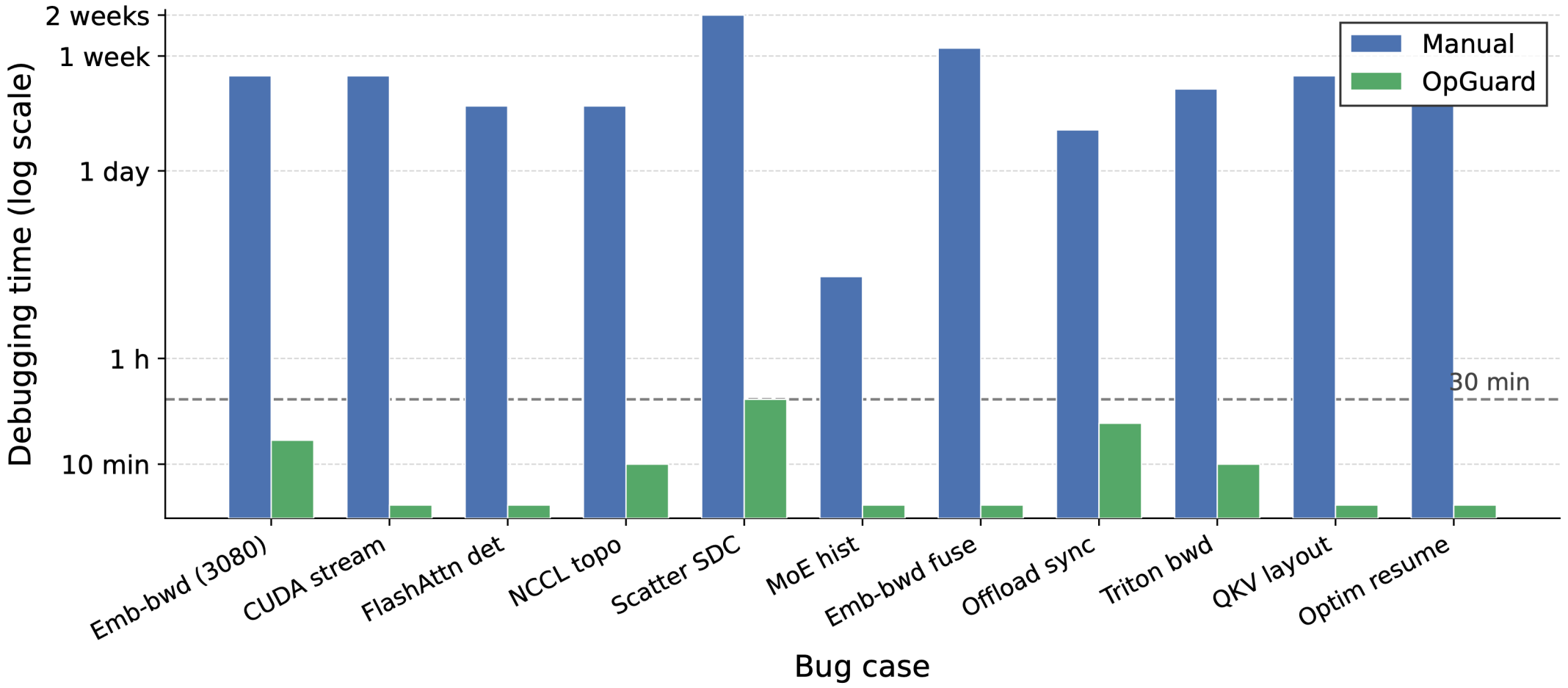
OpGuard successfully diagnoses **20 production cases** with diverse root cause

- **Wrong model implementation / logic**
- **Race conditions**
- **Unexpected numerical drift**
- **Hardware faults**



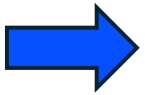
OpGuard also diagnoses 8 longstanding Github issues

OpGuard reduces triage time from days to minutes



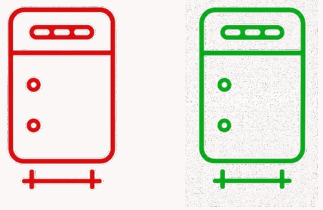
Unexpected deployment lesson: silent data corruption (SDC)

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

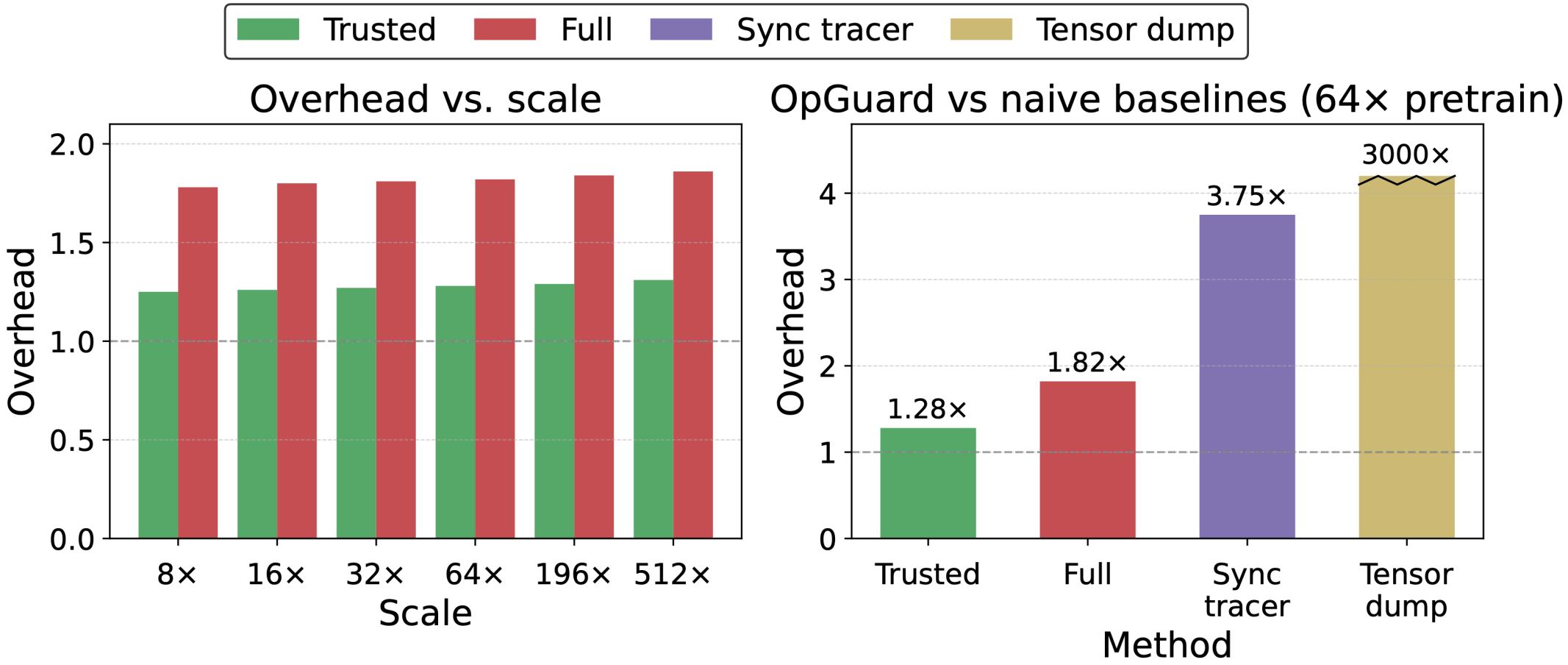
Internal: text & VLM pre-training, post-training framework

Opensource: Megatron-LM, DeepSpeed, GPT-Neox, Verl, etc.

Megatron LM
DeepSpeed
PyTorch Compiler
veScale
veRL
PyTorch FSDP
GPT-Neox
Transformers

Overhead

Trusted mode: skip bitwise signatures for trusted Ops



The major overhead comes from bitwise signature calculation

Conclusion

Bitwise alignment makes LLM training debugging precise, actionable, and practical

Precise primitive

Ambiguous composite signals

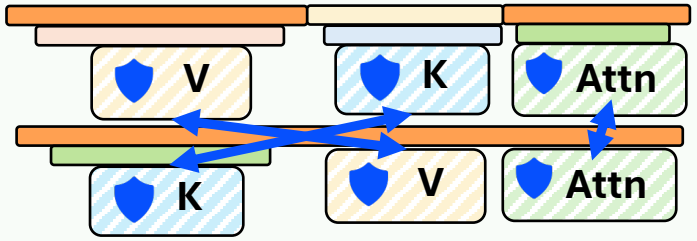
→ Op-level bitwise comparison

Clear pivot

First mismatch becomes the debugging pivot

Determinism control & robust alignment

Align Ops without freezing the training stack



Demo

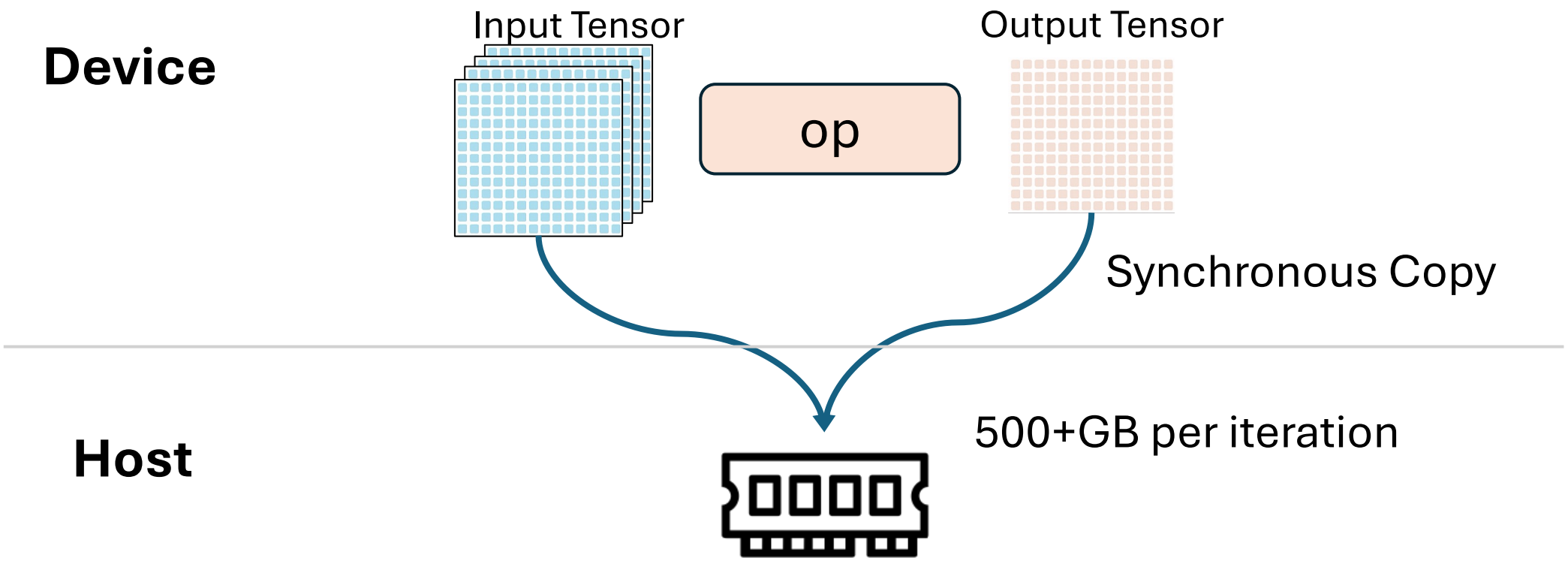
Deployed at **ByteDance**: **15+** teams · **20+** incidents

Diverse root causes, debugging time: **days** → **minutes**

Backup Slides

Bit-by-bit comparison with practical overhead

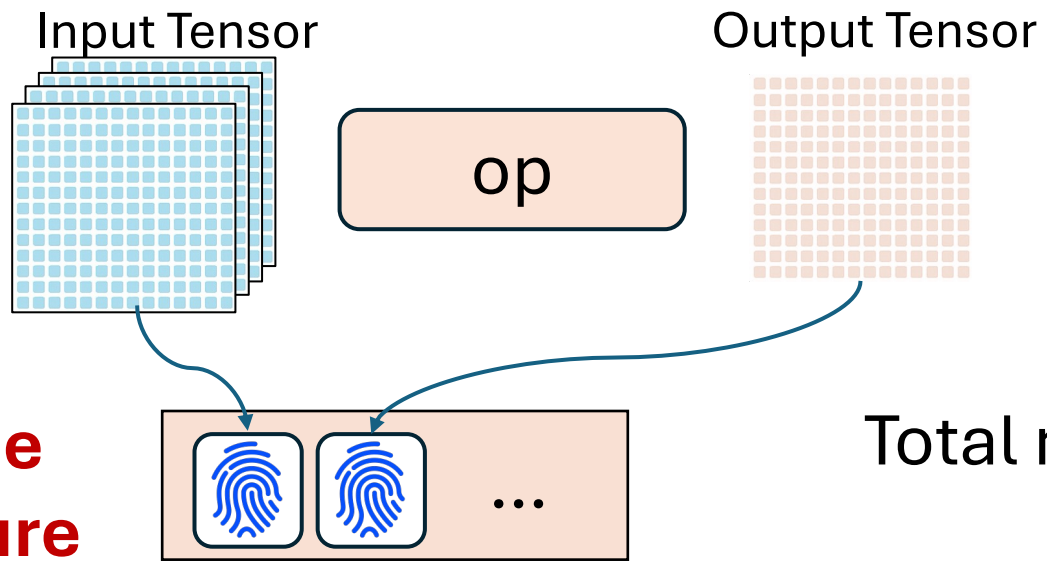
Capture full tensor is too slow



Replay run slow down 3000x

Bit-by-bit comparison with practical overhead

Solution: bitwise signature



Device

Bitwise Signature

Total memory overhead: <1MB

Host

Asynchronous flush to host at the end of each step

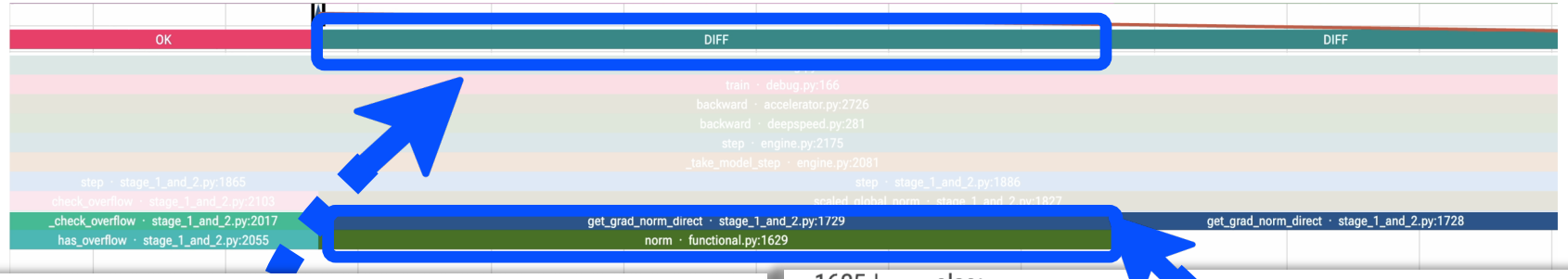
Replay run slow down at most 1.8x

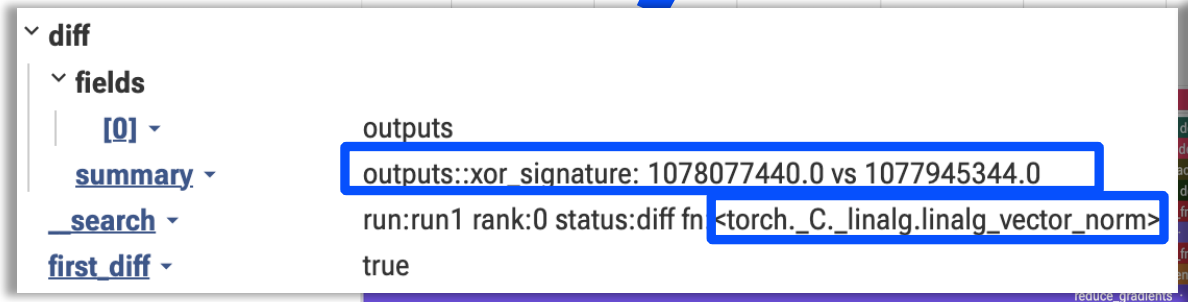
False positives

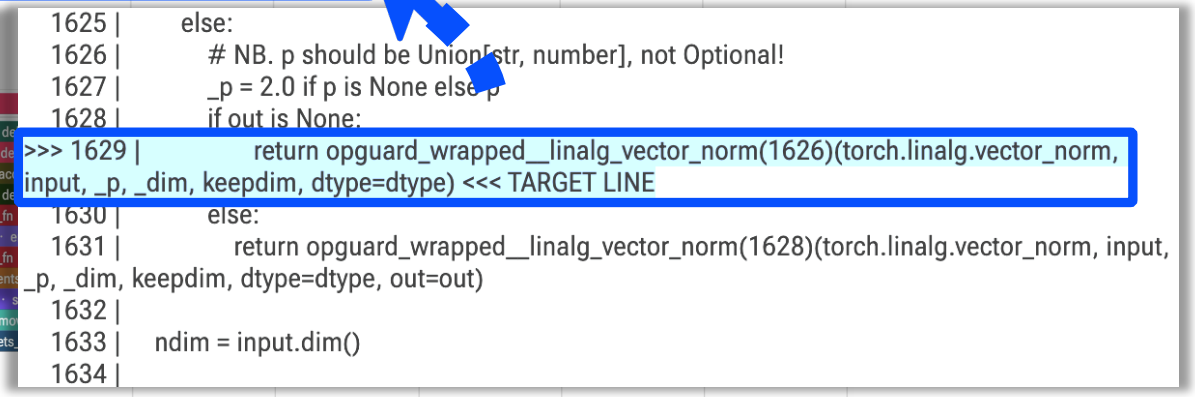
False positives: 2~10 during first adaptation

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

Inspect Concrete diff fields and source code via visualizer







Bitwise alignment under different parallelism

Example: TP-simulator (achieve bitwise alignment under different tensor parallelism)