UNIVERSITY OF MICHIGAN

OrderLab

TrainCheck

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Training with Confidence:

Catching Silent Errors in Deep Learning Training with Automated Proactive Checks

Yuxuan Jiang, Ziming Zhou, Boyu Xu, Beijie Liu, Runhui Xu, Peng Huang

Healthy Metrics, Broken Training

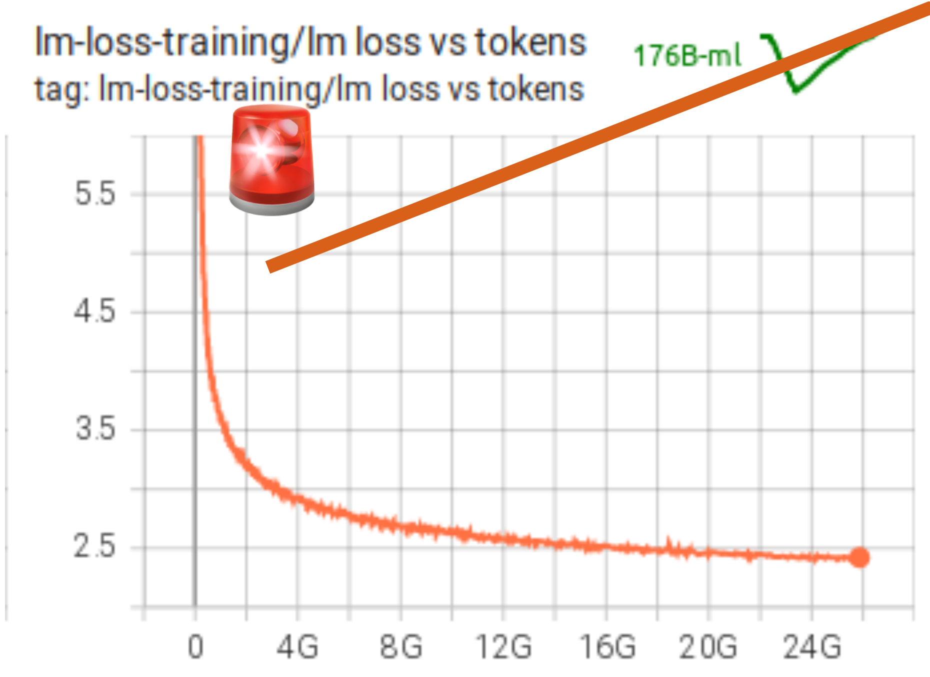
a BigScience initiative

BLOOM

176B params · 59 languages · Open-access

BLOOM (176B) – 384 A100 GPU, 3.5 months

✅ *"Loss curve looks healthy"*



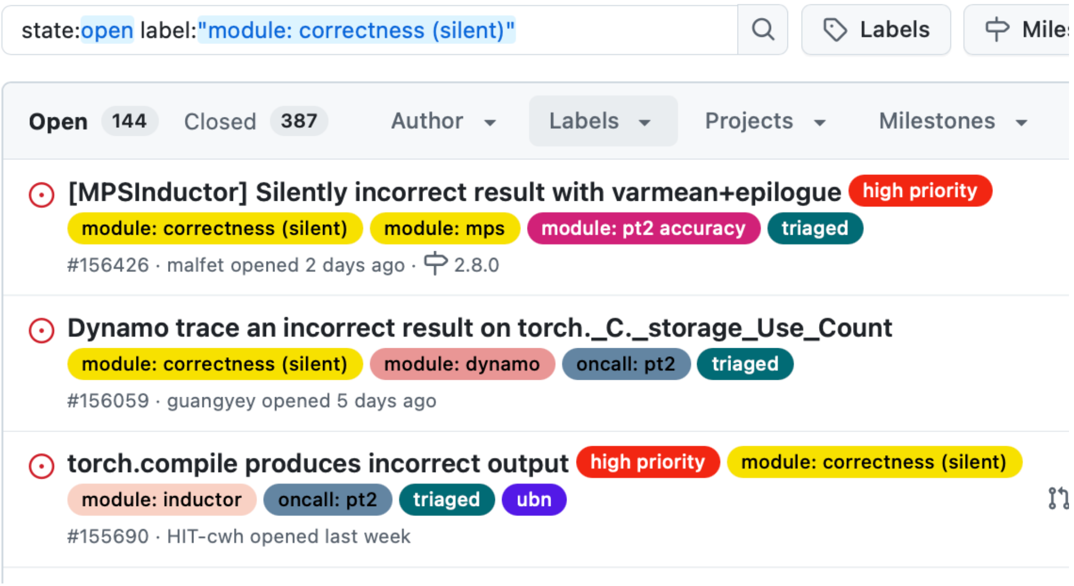
Took 10 days to notice, 4 more to diagnose and fix

🚨 *Weights silently diverged across GPUs*

- **Checkpoints became invalid**

⚠️ *Could've wasted 3.5 months & 384 A100s*

BLOOM Isn't Alone – Silent Training Errors Are Everywhere



Seen in other large-scale projects

- **OPT-175B**: 17 loss explosions, 3+ training method changes
- **BloombergGPT**: weight decay misapplied to all parameters
- **Shanghai AI Lab**: > 60% of GPU time spent on cancelled jobs

How to detect silent training errors early on?

Our Contribution: From Problem to Solution

🔍 **Studied 88 real-world silent training errors**

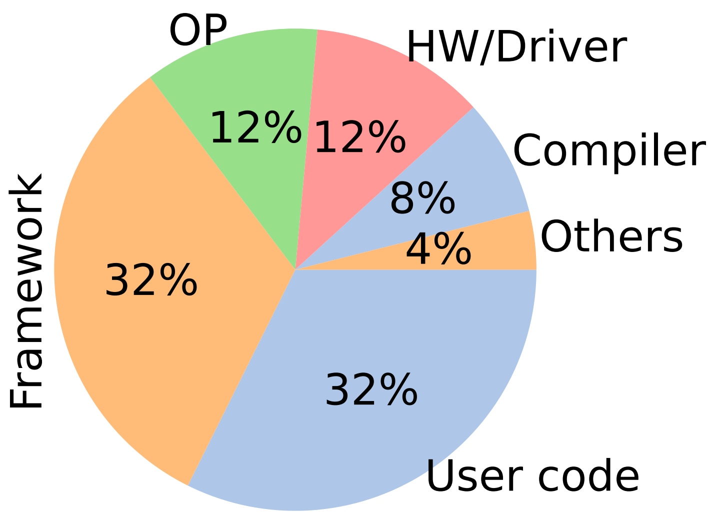
> GitHub issues, StackOverflow posts, and industry reports

🛠️ **TrainCheck: A System to Proactively Catch Silent Training Errors**

TrainCheck

What We Learned from 88 Silent Errors

Root causes are diverse and widespread



Single-component solutions (e.g., compiler testing) might be **inadequate**

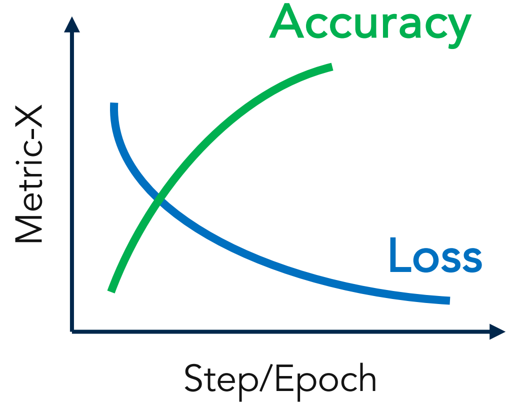
➡️ We need **runtime, end-to-end solutions** to detect issues **early** across **the full training stack.**

What We Learned from 88 Silent Errors

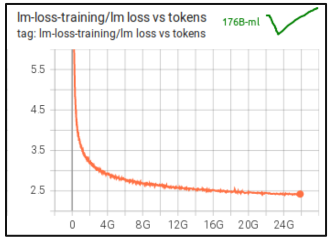
Hard to detect & severe impact

🧠 Eval metrics appear non-deterministic → ⏱️ Delays in detection

Weights & Biases TensorBoard

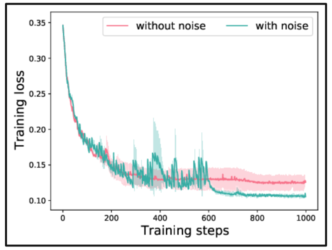


1



- No symptoms, until it's too late

2

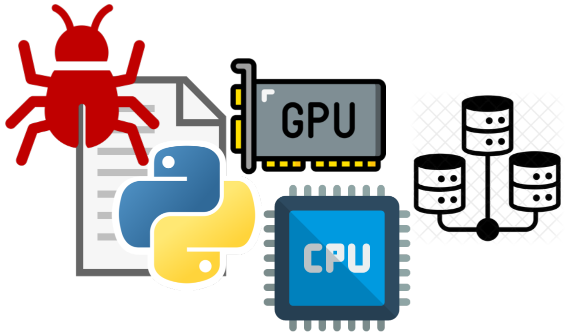


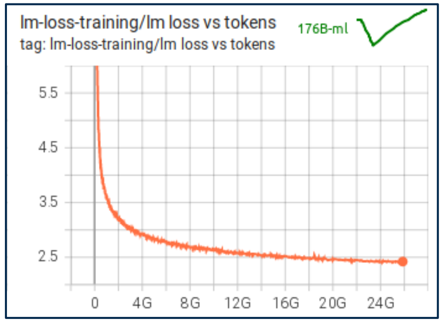
- Noisy signal, unclear if it's a real issue

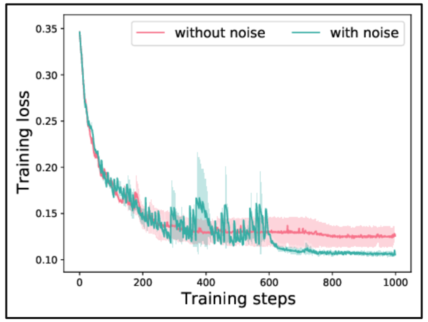
➡️ Early silent error detection should go **beyond eval metrics**

Training Invariants for Early Detection

★ Many silent errors have **precise**, **actionable** root causes







🎯Training Invariants

Concrete, accurate "specs" of the low-level components

✅ **Enable early detection**

➡️ Non-determinism is an artifact of checking at too high levels

Example: Bloom Parameter Divergence

Root cause: gradient clipping is only applied to the first worker within tensor parallel (TP) groups

```
@torch.no_grad()
def get_grads_for_norm(self, for_clipping=False):
    grads = []
    tensor_mp_rank = bwc_tensor_model_parallel_rank(mpu
    for i, group in enumerate(self.bf16_groups):
        for j, lp in enumerate(group):
            if not for_clipping:
                if hasattr(lp, PIPE_REPLICATED) and lp.
                    continue

            if not (tensor_mp_rank == 0 or is_model_par
                continue # YUXUAN: as compared to the

            if not self.fp32_groups_h
                continue

            grads.append(self.fp32_gr

    return grads
```

1. `for_clipping == False` → collect gradients to compute norm (de-duplication needed)

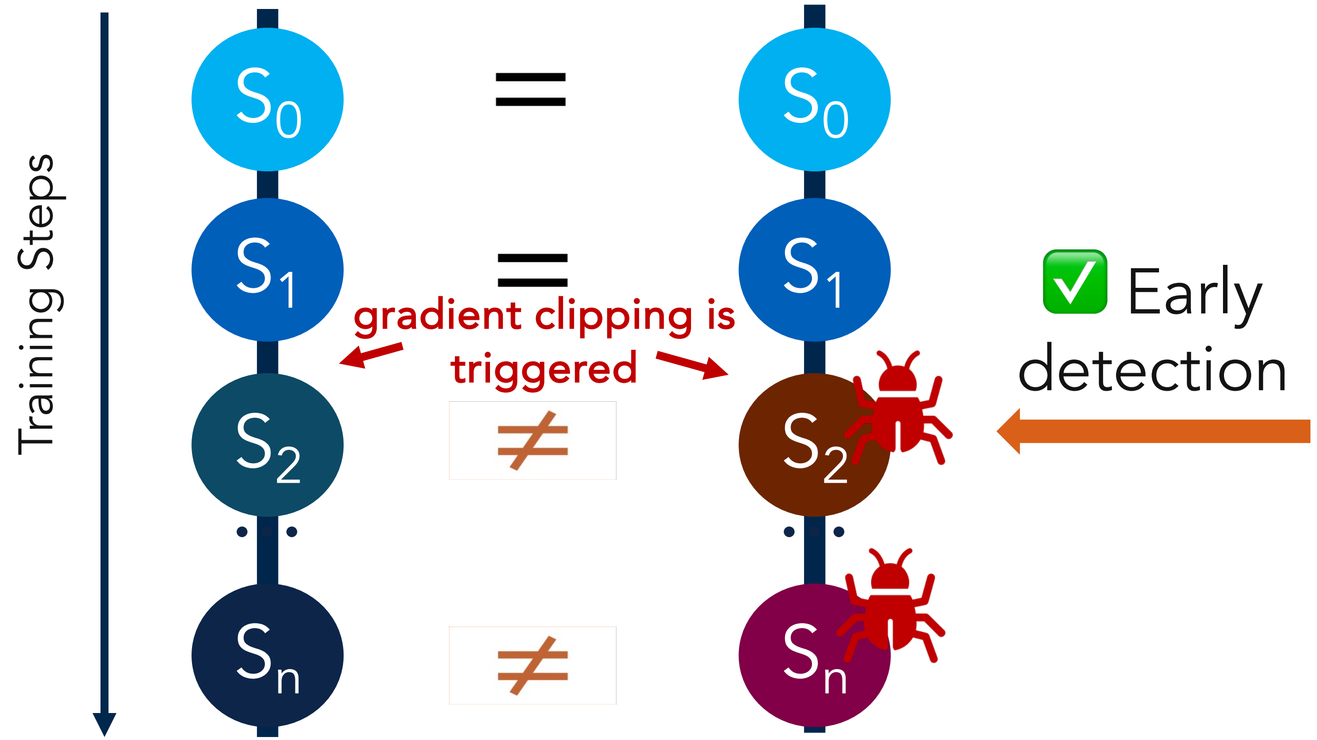
2. `for_clipping == True` → collect gradients to be clipped (all gradients needs to be clipped)

The de-duplication logic is misplaced to for_clipping == True

Example: Bloom Parameter Divergence

Root cause: gradient clipping is only applied to **the first worker** within tensor parallel (TP) groups

TP Worker 0 TP Worker 1



❌ **Error not detected until end of training**

Training Invariant:

1. API Behavior Invariant

`get_grad_for_norm` API contract

2. State Relationship:

Parameters should be equal across workers

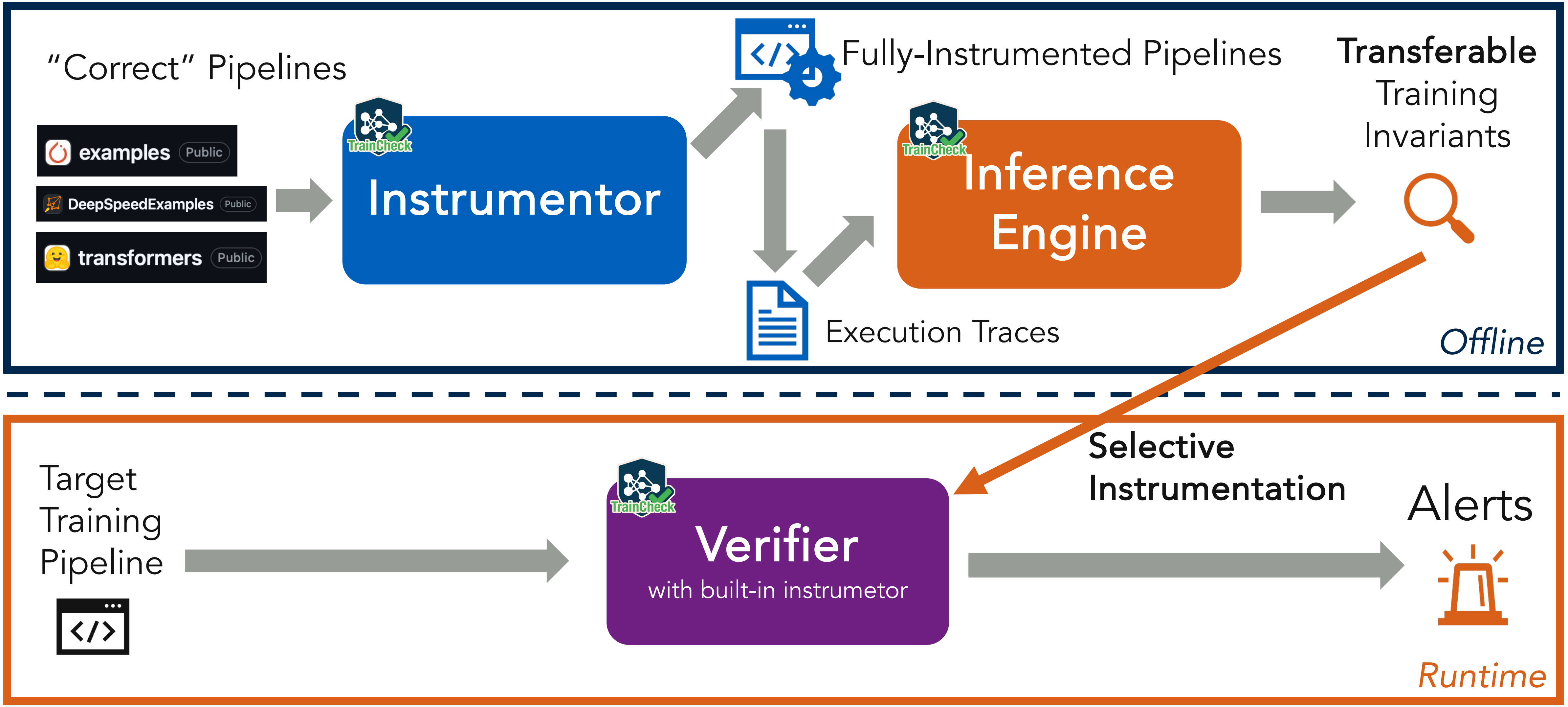
TrainCheck

An end-to-end system that **infers** and **checks** training invariants to prevent silent training errors

Goals:

- Check properties lower than high-level signals
- Automated workflows
- Continuous runtime validation
- Systematically cover diverse root causes

Automated Inference + Proactive Validation



Inference Engine: Key Challenges

1. **Inferring Semantically Relevant Invariants**

2. **Context-sensitive Semantics**
   - DL behaviors depend on subtle runtime contexts
   - Statistical likelihood might not be a good indicator of invariant validity

3. **Limited Development Histories for Inference**
   → Invariants must be **transferable**

4. **A Huge Search Space**
   - Each iteration logs 50 MB of traces (e.g., GPT-2 pretraining)

Invariant Representation

"The **weights** of certain layers should stay **consistent** across **tensor parallelism (TP) ranks**."

(1) Relation

(2) Descriptors – Abstraction over concrete API / variable instances to check

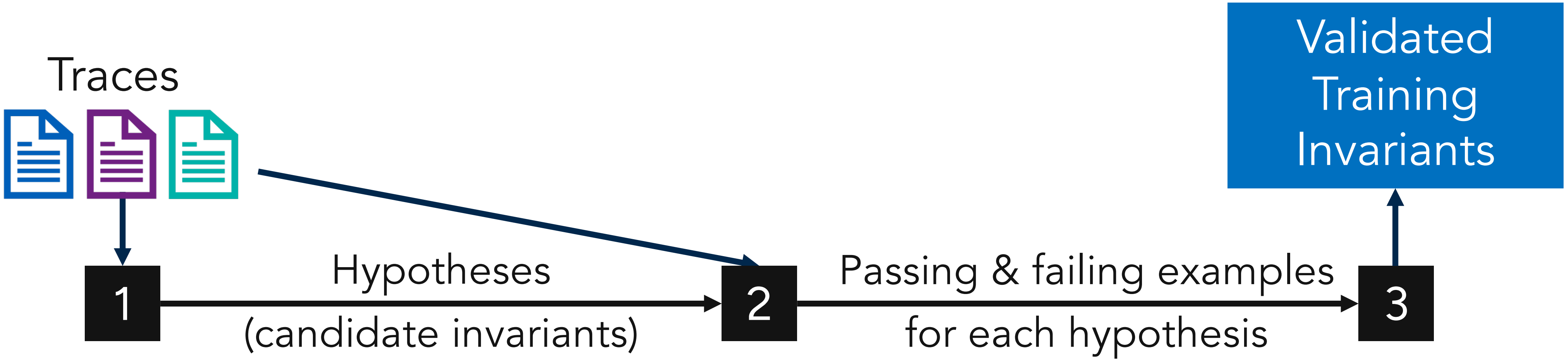
```
Consistent(torch.nn.Parameter.data, torch.nn.Parameter.data)

&& UNEQUAL(meta_vars.TP_RANK)
&& EQUAL(meta_vars.step)
&& EQUAL(name)
&& CONSTANT(attr.tensor_model_parallel, false)
```

!! Only applicable to **LayerNorm** (**<1%** of parameters).

(3) ★ Precondition (Context)

Invariant Inference Workflow



Proactive Hypothesis Generation

- Matches of relation observed → Hypothesis

Full Hypothesis Validation

- Full scan of hypotheses on traces

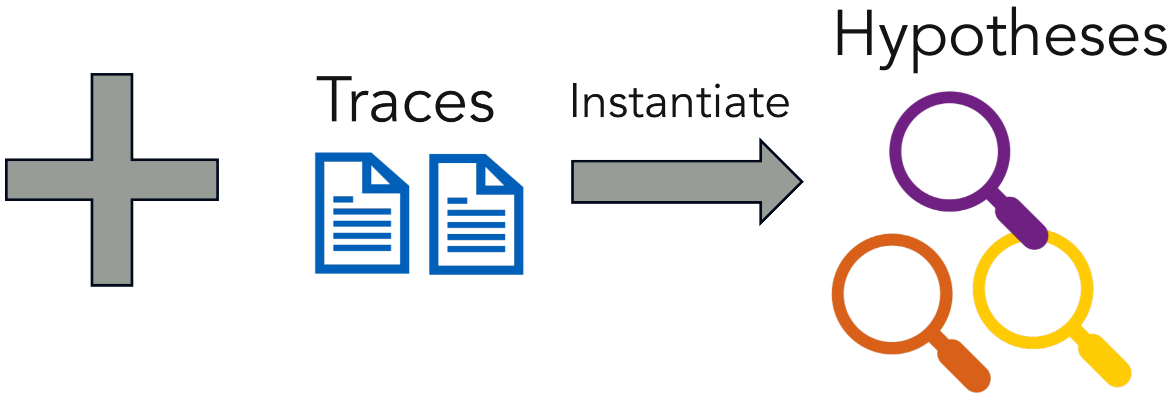
Precondition Deduction

- Determine applicable contexts

Inferring DL-tailored Invariants via Relations

Instantiate invariants using domain-specific templates for DL systems

Relation	Description
`Consistent``(Va,Vb)`	Va and Vb should have the same values, while the values may change
`EventContain``(Ea, Eb)`	Eb must happen in the duration of Ea
`APISequence``(Ia, Ib, ...)`	Ia, Ib, . . . must all occur and in the specified order
`APIArg``(Ia, is_distinct)`	Ensures argument consistency or distinction in all calls to Ia
`APIOutput``(Ia, bound_type)`	The output of Ia must meet certain attribute constraints

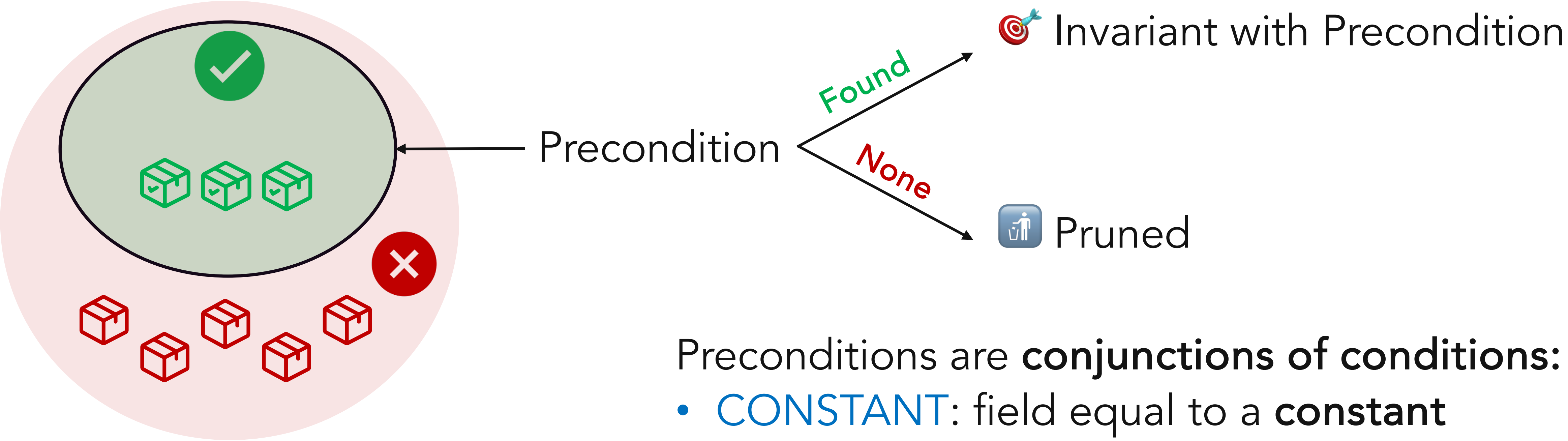


→ Narrows the search space

→ Keeps inference relevant to training semantics

★ Precondition

For every hypothesis, infer a precondition based on passing/failing examples:



Preconditions are **conjunctions of conditions:**

- CONSTANT: field equal to a **constant**
- EQUAL: field has **the same value**
- UNEQUAL: field has **different** values
- EXIST: field **exists**

Why Precondition

- **Transferability** across different training setups
- Validity of DL invariants is not tied to statistical likelihood
  → Help **preserve rare but meaningful invariants**
  → **Prune superficial ones** that happen to hold frequently

✅ **Consistency Invariant (Bloom)**
- Critical for correctness
- 1:38 Passing to Failing Ratio
- Accepted due to valid precondition

❌ `Consistent(torch.Tensor.is_cuda, torch.Tensor.requires_grad)`
- Superficial & irrelevant
- Holds 99% of the time
- Rejected due to missing precondition

Effort-free, Low-overhead Instrumentation

- Dynamic Instrumentation Via **Monkey-Patching** (API) & **Proxy** (Variable)

```
torch.matmul = wrapper(torch.matmul)      model = Proxy(model)
```

- Low-overhead Checking Stage via **Selective Instrumentation**

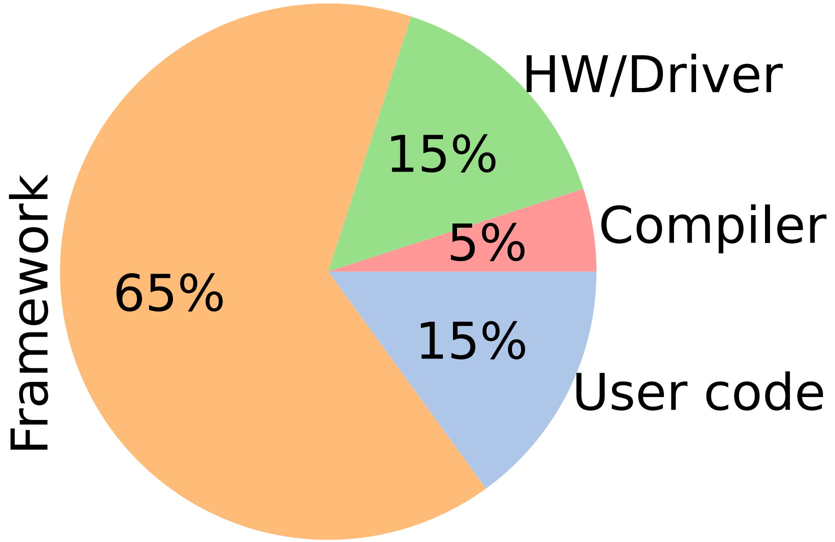
Invariants

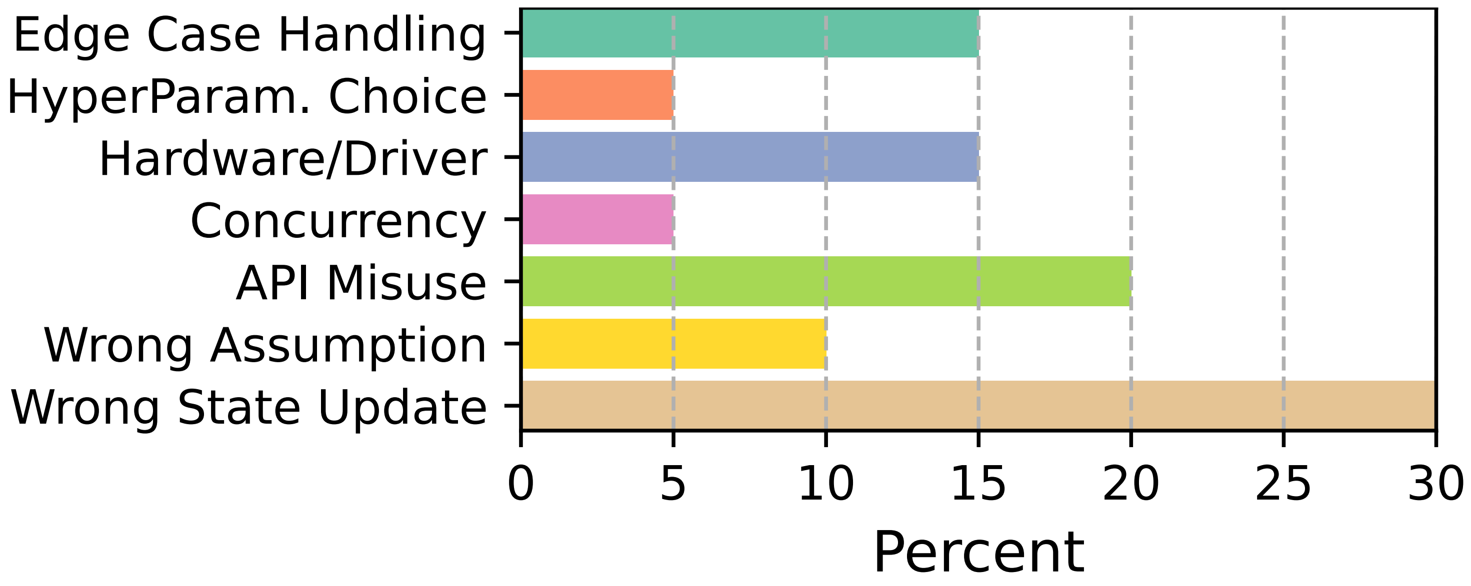
Instrument Options
- APIs of interest
- Trace dumping granularity

E.g. Bloom-176B parameter consistency invariant only needs a parameter dump per iteration.

Detection & Diagnosis Benchmark

- We collect and reproduce 20 real-world silent training errors
  - 6 in the empirical study, 14 newly collected





Quick & Actionable Detection

✅ **TrainCheck detects 18 out of 20 real-world silent training errors within 1 iteration**

✅ **TrainCheck provides actionable diagnosis clues**

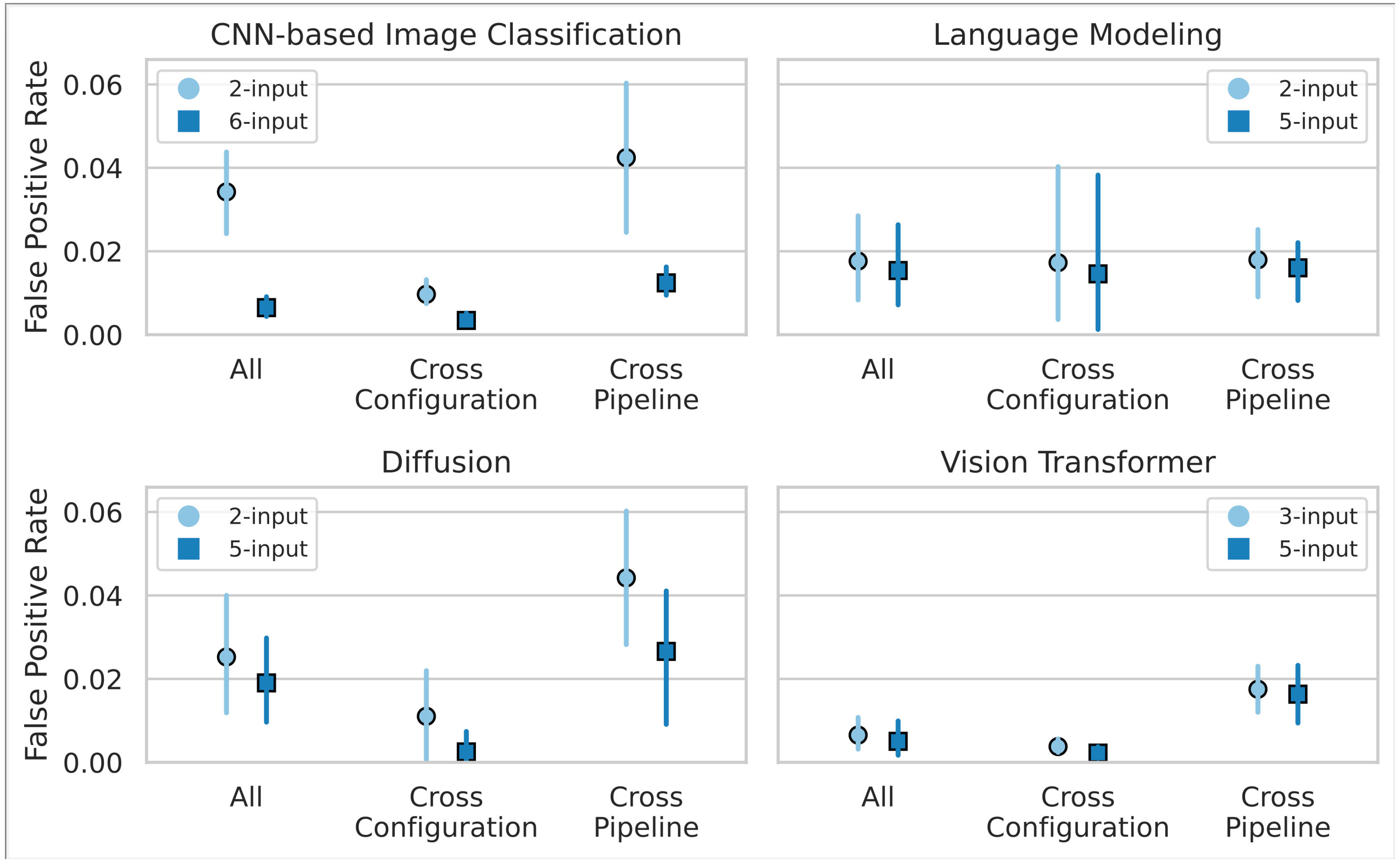
- **Pinpoints** the exact root cause in **10** cases, close to the root cause in **8** more

⚠️ **Baselines (stats monitoring, PyTea + NeuRI)**

- Detect **3/20** cases total
- Pinpoints **only 1** root cause

Another **6 new bugs** exposed in DeepSpeed and Transformers

False Positive Rate <2%



- 63 representative pipelines, diffs in **scale**, **complexity**, and **frameworks** used

👉 TrainCheck consistently shows **< 2% FP rate** with **5** representative input pipelines

A Small Set of Inputs to Detect Many Errors

- **Invariants used for all 18 cases are inferred from example pipelines.**

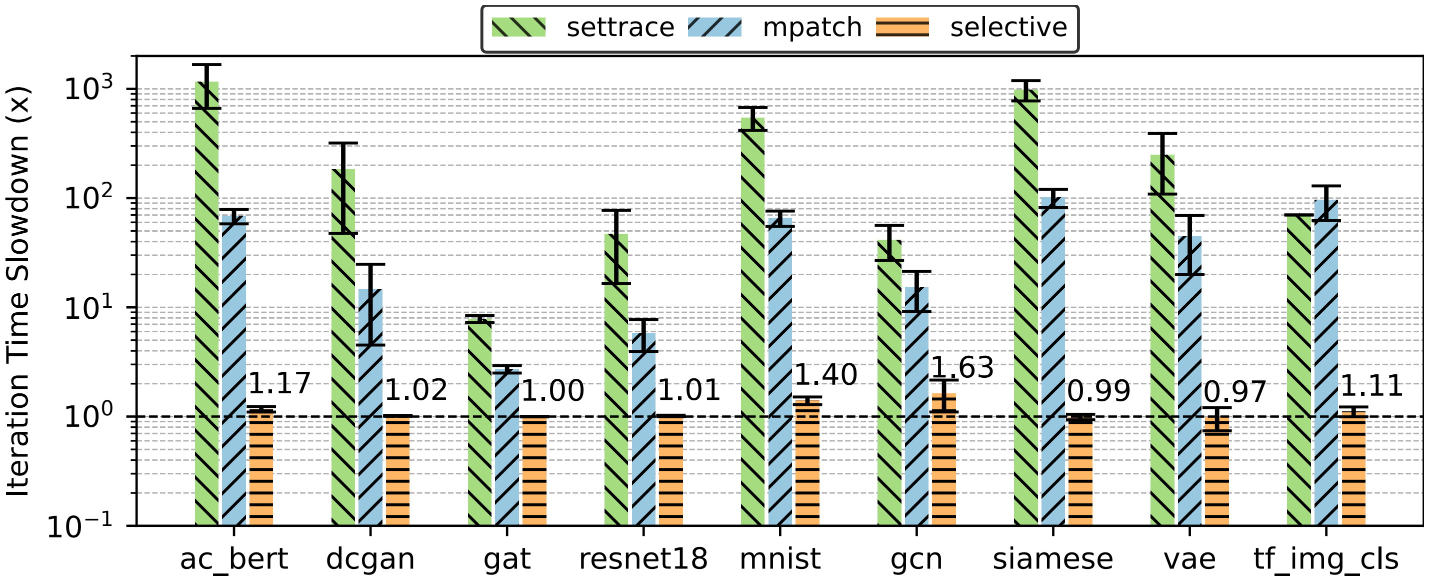
  PyTorch case study:
  - *GCN* covers **77%** of silent issues
  - *GCN + Autocast + DDP* covers **100%**

- **One invariant, many pipelines**
  - 23% of inferred invariants in FP evaluation transfer across **different training tasks**
  - **Conditional** invariants **transfer better** than **unconditional** ones

Invariants can be inferred once and reused across pipelines

Runtime overhead

- Measure per-iteration time slowdown before/after instrumentation.



- Typical checking stage (selective with 100 invariants deployed) is < 11%

Conclusions

Silent training errors are **prevalent**, **costly**, and **hard to detect**

TrainCheck: **automated validation** of training tasks using **inferred invariants**

TrainCheck

https://github.com/OrderLab/TrainCheck

Precondition deduction to ensure **precision** and **transferability**

Actively Maintained!

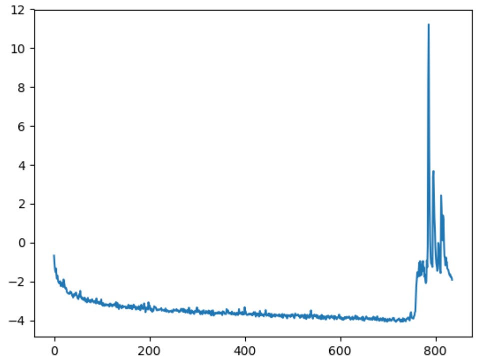
Key results:

- Caught **18/20** real-world silent issues, identified **6 new bugs**
- **≤ 2%** false positive rate, overhead ≤ 11% in realistic settings

Backup Slides

What Silent Issues Does TrainCheck Target?

- TrainCheck targets objective correctness violations.



Optimization-Sensitive Choices

Model Architecture
Optimizer
Learning Rate
...

TrainCheck focuses here

Correctness Violations

Incorrect API usage
Buggy Library Implementation
Faulty Hardware
...

Case Study – AC-2665 Stagnant Training

- Root Cause: FSDP flattened parameters, corrupting the optimizer state
- Applying invariants from the PyTorch GCN example resulted in **100 violations (52 true alarms).**

- **True Positives (52):**
  - 33 → `torch.optim.adamw.adamw` were never invoked
  - 17 → `optimizer.step` did not perform any update
  - 1 → `optimizer.zero_grad` did not zero out gradients

  →⚡ **Optimizers were not properly initialized with model parameters!**

- **False positives** (48) were quickly dismissed
  - 26 → missing ReLU invocations (but T5 does not use ReLU)
  - 7 → specific numerical values in GCN training (e.g., `dropout_rate==0.5`)
  - **Structured inspection allows quick identification of TP/FP**

Example: Bloom Parameter Divergence

Trace snippet for torch.nn.Parameter

1 {"name": "layernorm.weight", "type": "torch.nn.Parameter", "meta_vars": {"TP_RANK": 0, ...}, "attr": {"data": 411977, "is_cuda": true, "tensor_model_parallel": false, ...}}

2 {"name": "layernorm.weight", "type": "torch.nn.Parameter", "meta_vars": {"TP_RANK": 1, ...}, "attr": {"data": 411977, "is_cuda": true, "tensor_model_parallel": false, ...}}

3 {"name": "dense_h_to_4h.bias", "type": "torch.nn.Parameter", "meta_vars": {"TP_RANK": 1, ...}, "attr": {"data": 650462, "is_cuda": true, "tensor_model_parallel": true, ...}}

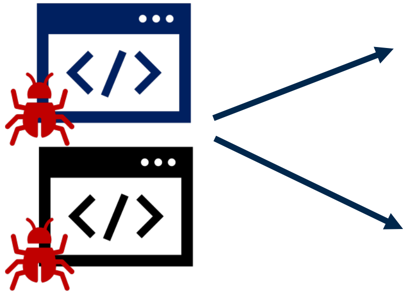
1. Generate hypothesis `Consistent(torch.nn.Parameter.data, torch.nn.Parameter.data)`
2. Validate hypothesis Passing samples: (1,2) Failing samples: (1, 3), (2,3)
3. Deduce precondition `UNEQUAL(meta_vars.TP_RANK) && EQUAL(meta_vars.step) CONSTANT(attr.tensor_model_parallel, false) && EQUAL(name)`

Baselines and methodology

- High-level signal
  - (1) Spike, (2) Trend (3) Anomaly Detection

- Existing research artifact
  - PyTea [ICSE'22] + NeuRI [ESEC/FSE'23]: Automatically inferring and checking shaping constraints for APIs.

Pipelines with Silent Issues



Train stats (high-level signals)

Trace (TrainCheck, and PyTea + NeuRI)

🤔 *How to get these invariants?*

⚠️ **Manual specification/debugging doesn't scale**

- Infrastructure is complex, and evolution is fast-paced
- Encoding intuitions into accurate checks is hard

➡️ **Automated** inference of **precise, context-aware** invariants

Rough Invariant for Catching the Bloom Parameter Divergence Error

(1) Entities to be checked **(2) Relationship**

The weights of certain layers should stay consistent

across tensor parallelism (TP) ranks

(3) Meta Variables