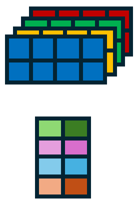
Verifying Semantic Equivalence of Large Models with Equality Saturation

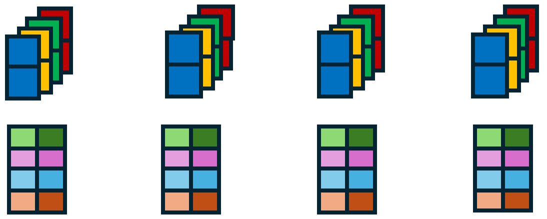
Kahfi S. Zulkifli*, Wenbo Qian*, Shaowei Zhu, Yuan Zhou, Zhen Zhang, Chang Lou

EUROMLSYS 2025

Enabling large models through scaling that are prone to silent errors

Llama 3.1
405 B
deepseek
671 B
Don't fit on one GPU

Model
Data

GPU 0
Single device


GPU 0 GPU 1 GPU 2 GPU 3
Tensor Parallelism (TP)

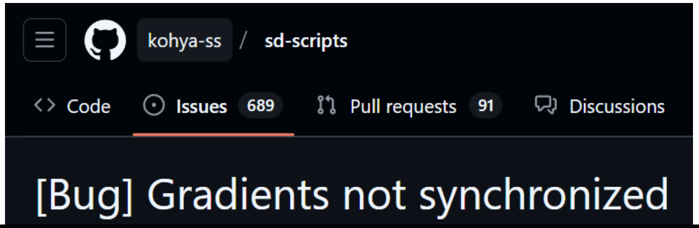
Scaling techniques are complex

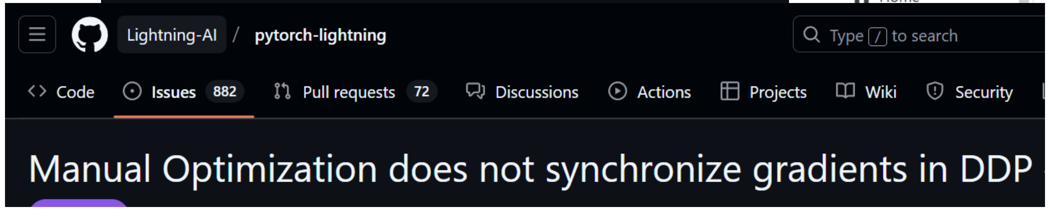
Sharding Communication

Optimizer Schedule

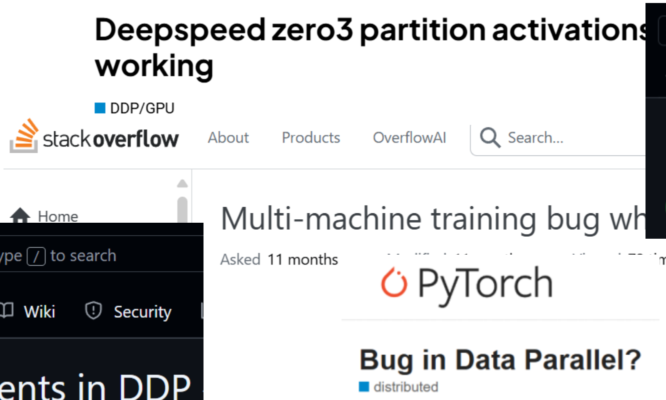
Prone to silent errors

Wrong communication operations

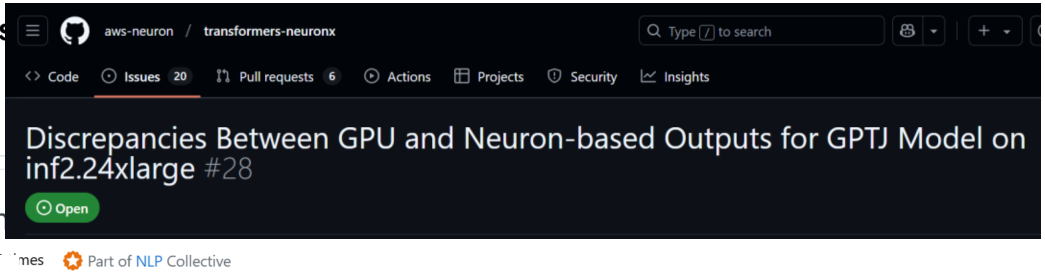




Wrong sharding



Wrong calculation



Behavior not like single-device pipeline, loss value not decreasing or garbage outputs

Model quality to drop

A silent error in AWS Transformers Neuron, a machine learning inference library

Goal: Slice key tensor from query-key-value matrix

```
--- attention.py
slice_lim = active_qkv.size[-1]//
  (n_heads_tp + 2 * n_kv_heads_tp)
active_k = hlo.slice_along(active_qkv, -1,
  (n_heads_tp+n_kv_heads_tp)*slice_lim,
  start=0)
```

```
+++ attention.py
slice_lim = active_qkv.size[-1]//
  (n_heads_tp + 2 * n_kv_heads_tp)
active_k = hlo.slice_along(active_qkv, -1,
  (n_heads_tp+n_kv_heads_tp)*slice_lim,
  start=n_heads_tp*slice_lim)
```

Bug causes incorrect model outputs

Fix is simple, but difficult to detect

ML pipeline consists of multiple modules

Optimizer

Sharding

Scheduling

Backend

No explicit error signals

Crash

Hang

*This bug is already fixed

Silent bugs are tricky since they are subtle

Runtime recovery

CheckFreq [FAST ‘21], Varuna [EuroSys ‘22], GEMINI [SOSP ‘23], Oobleck [SOSP ‘23], Bamboo [NSDI ‘23], ReCycle [SOSP ‘24]

- + Fault-tolerant to failures
- − Relies on explicit error signals

Our Position: Expose silent errors before deployment

Testing frameworks

DeepXplore [SOSP ‘17], DeepTest [ICSE ‘18], Eagle [ICSE ‘22], NNSmith [ASPLOS ‘23], MLIRSmith [ASE ‘23], PolyJuice [OOPSLA ‘24]

- + Detects many bugs
- − No guarantee of absence of bugs

Our Position: Guarantee absence of errors in pipelines

Developers approach in debugging is ad-hoc

Examine intermediate tensor values in the entire huge code space manually

attention.py

```
print(...)
slice_lim = active_qkv.size[-1]//
  (n_heads_tp + 2 * n_kv_heads_tp)
print(...)
active_k = hlo.slice_along(active_qkv, -1,
  (n_heads_tp+n_kv_heads_tp)*slice_lim,
  start=0)
print(...)
```

Optimizer

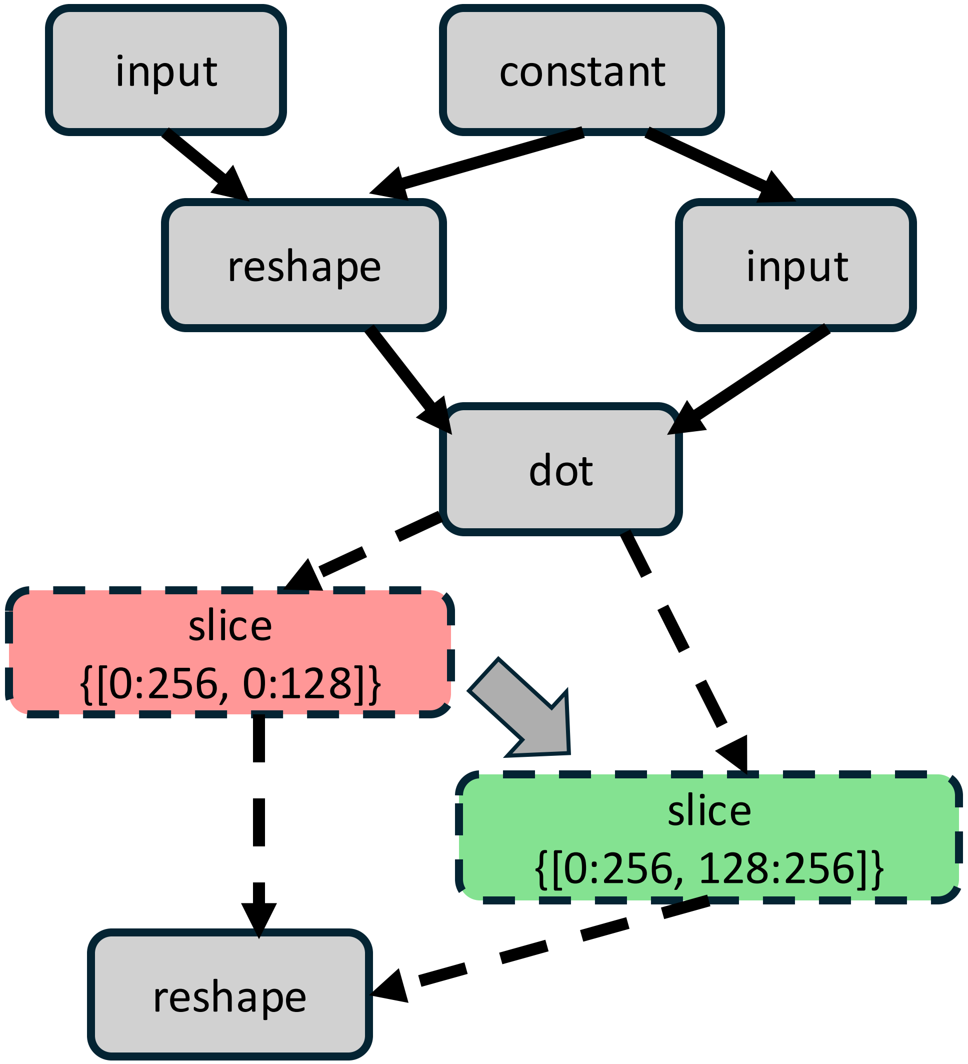
Sharding

Scheduling

Backend

- Numerous amount of phases
- Hard to differentiate correct and wrong tensors due to floating-point round-off errors
- Tedious to manually piece tensors on multiple devices to match single on

Expose silent errors without explicit signals

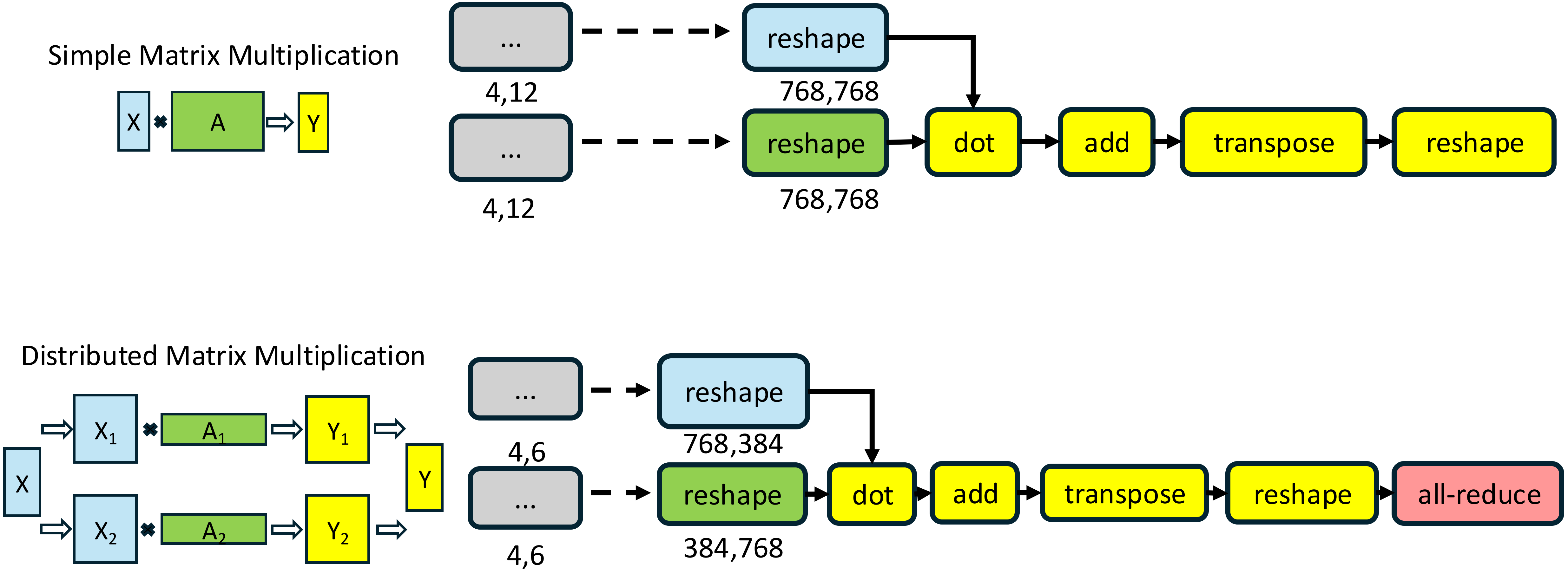


Insight: Silent errors are introduced by semantic changes, reflected in computational graphs

```
--- attention.py
slice_lim = active_qkv.size[-1]//
  (n_heads_tp + 2 * n_kv_heads_tp)
active_k = hlo.slice_along(active_qkv, -1,
  (n_heads_tp+n_kv_heads_tp)*slice_lim,
  start=0)
+++ attention.py
slice_lim = active_qkv.size[-1]//
  (n_heads_tp + 2 * n_kv_heads_tp)
active_k = hlo.slice_along(active_qkv, -1,
  (n_heads_tp+n_kv_heads_tp)*slice_lim,
  start=n_heads_tp*slice_lim)
```

Know correct computation by having a baseline model to compare

Approach: Verify semantic equivalence



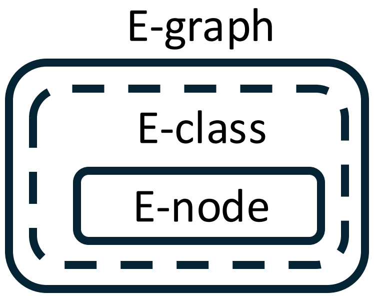
Graph rewriting with equality saturation

Original graph G, transformed graph T, rewrite T so that it becomes equivalent to G

T → T1 → T2 → T3 → T4 → G

T → T1 → T2 → T5 → T6 → ?

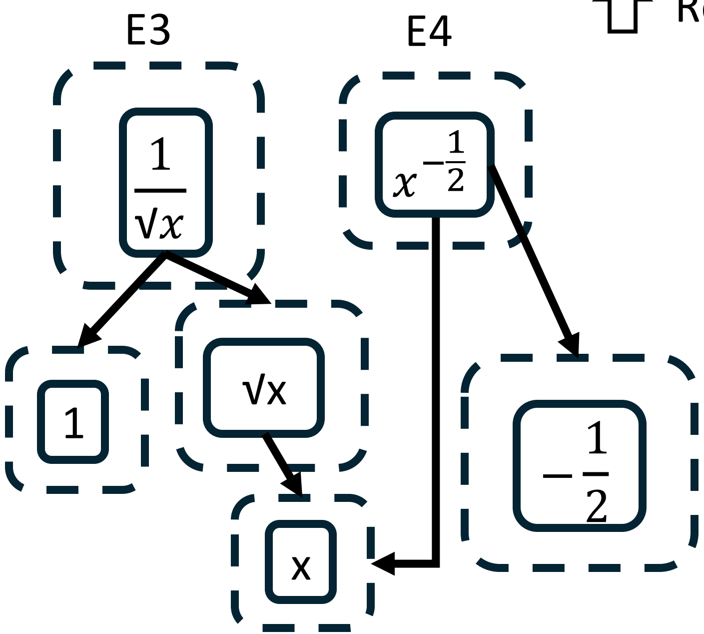
So many ways to rewrite via semantic-preserving transformations in various different orders



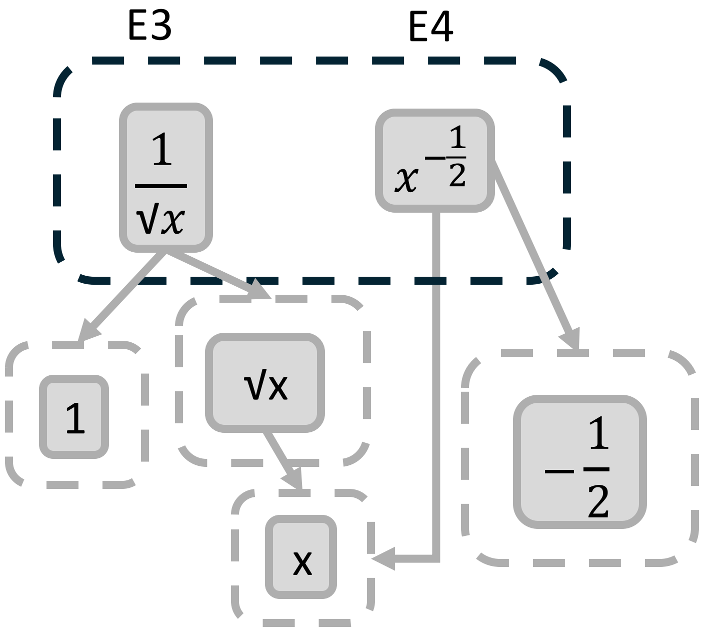
Expression 1: G
Expression 2: T

⇒ E-graph E: G, T ⇒ T is equivalent to G?

⇧ Rewrite Rules

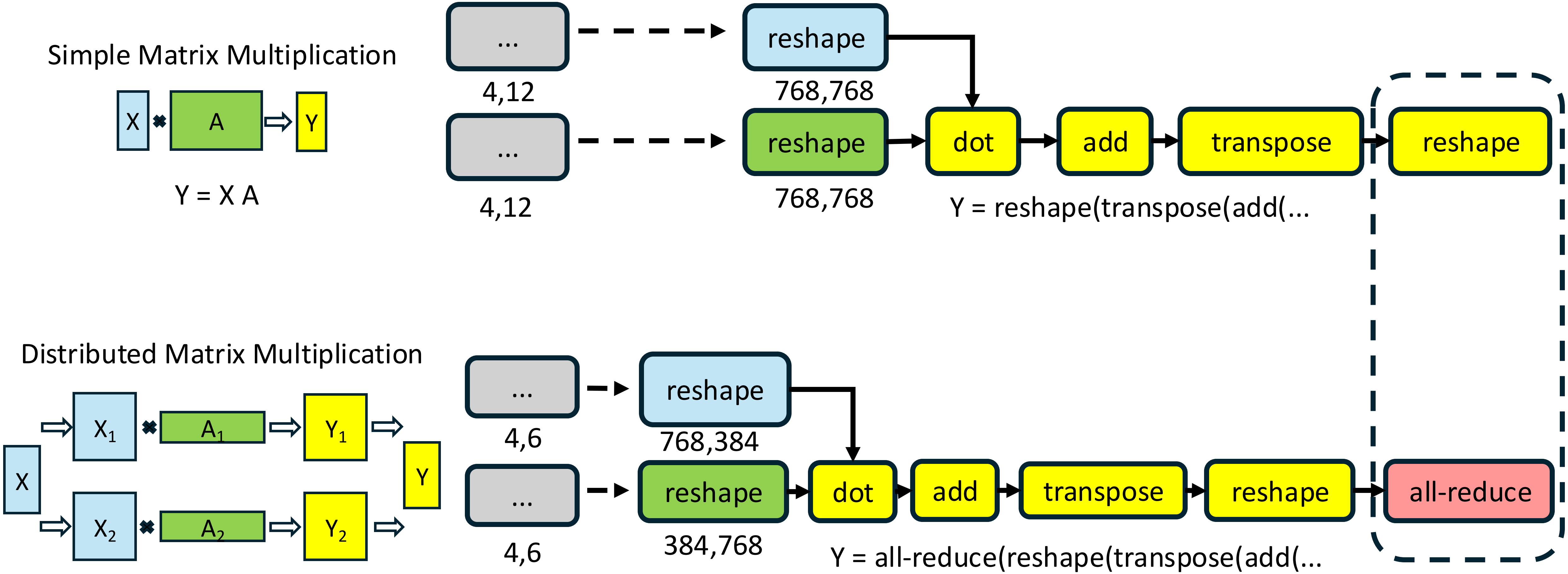


$$\frac{1}{\sqrt{x}} \rightarrow x^{-\frac{1}{2}}$$



Equality saturation in computation graphs

Outputs of original graph = Outputs of transformed graph ?



$Y = X\,A,\ X = [X_1, X_2],\ A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$

Rule generality and practicality

Generic rule
dot(x, y) → ...

- Matches too many e-nodes

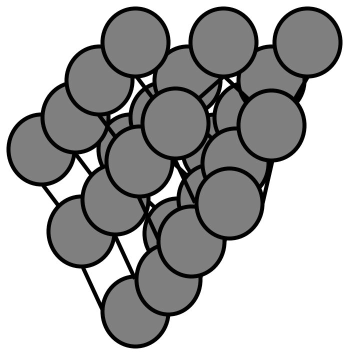
Specific rule
all-reduce(reshape(transpose(x))) → transpose(transpose(reshape(x)))

- Covers too few cases

Solution
- Layout and distribution analysis of tensors with Datalog-style reasoning
    - Compute relations between single device and distributed tensor and propagated through operator

- Rewrite rule generation
    - Using predefined templates, reason about different layout transformations between single-device and distributed tensor

Graph scaling in large models

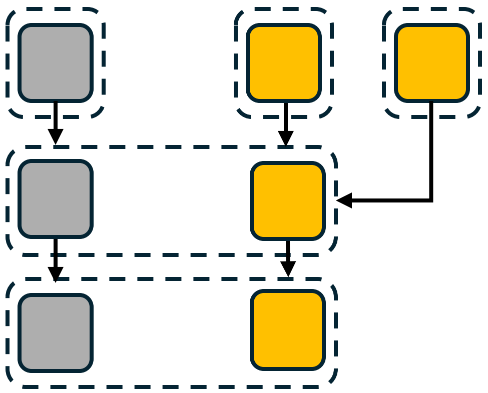


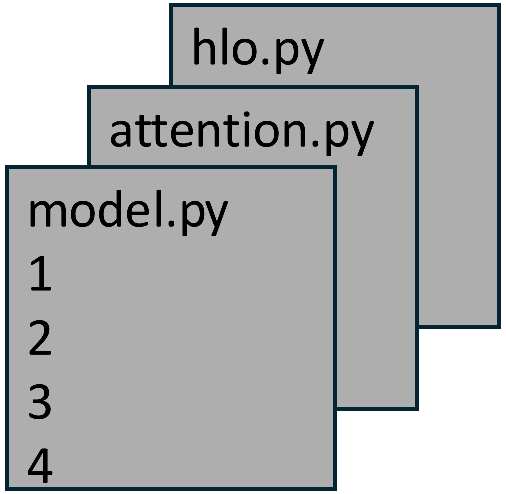
3 hours

E-graph larger than computational graph and grow at an exponential rate compared to the growth of computational graph

Solution
- Graph partitioning with heuristics
    - Divide at layer boundaries and predefined list of operators (e.g. softmax)

Lack of debugging support





Manually going through the whole code space is tedious

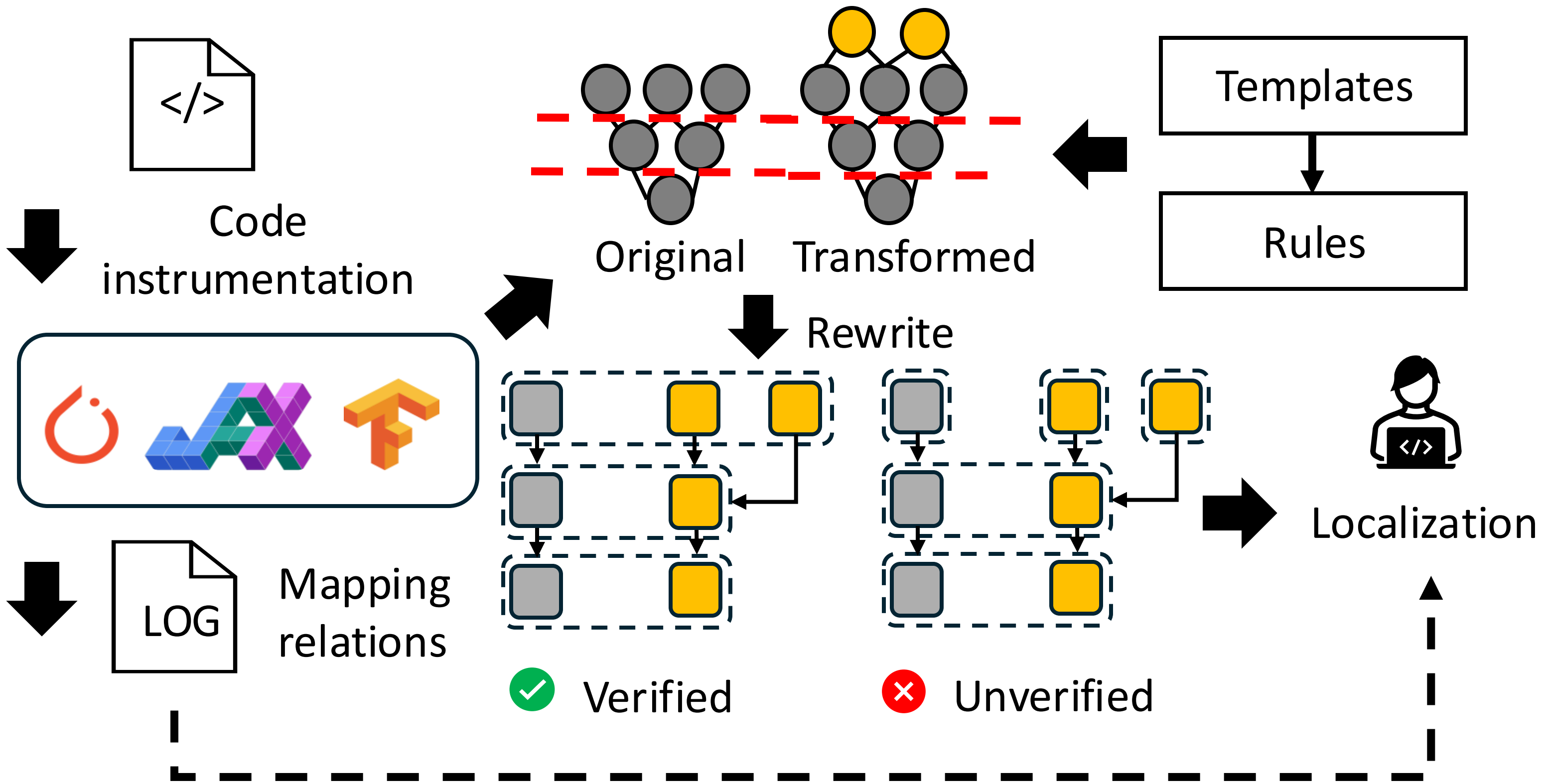
Solution

- Bug localization
  - Create nodes with metadata referring to the source code file and line number
  - Gives out list of unverified nodes with the metadata

Our system and workflow

AERIFY

- A framework that automatically verifies semantic equivalence of large models with equality saturation



Preliminary results and discussion

Preliminary results
- Built on top of egglog

- Applied to AWS transformers-neuronx inference library

- Detected 2 real-world silent errors with 12 semantic rules

Discussion
- Support fine-grained parallelisms with schedules (timing information)

- Extend to other frameworks (Deepspeed) and more models

- Integrate LLMs into debugging process

Conclusion

Machine learning models are increasingly complex and lead to silent errors

- These subtle errors cannot be detected with existing methods and cause model to have lower quality

Silent errors are reflected at the semantic level in generated IR graphs

- Rewrite transformed graph to make it equivalent to baseline graph

AERIFY automatically verifies computation graphs of large models with equality saturation

Techniques include rewrite rule generation, tensor layout analysis and bug localization