

Inductor-TV: Formal Methods for the Pytorch Compiler

https://www.researchgate.net/publication/403799320_Inductor-TV_Forma_Methods_for_the_Pytorch_Compiler
Abhilash Majumder · IEEE AIRC 2026, Savannah · IEEE COMPSAC 2026 · DOI 10.1109/AIRC69745.2026.11631461 · IEEE 11631461

TorchBench Model	Binary Folding	Pad MM	Fused SDPA	Reinplace	Joint Graph	Split Cat	GBF	Verified	Status
BERT_pytorch	0	2	0	1	1	634	0	50	SUCCESS
nanogpt	0	0	12	1	0	466	0	60	SUCCESS
llama	0	0	8	1	0	429	0	8	SUCCESS
hf_GPT2	0	1	12	1	0	427	0	60	SUCCESS
hf_DistBert	0	1	6	1	1	274	0	26	SUCCESS
hf_Bert	0	1	12	1	1	527	0	50	SUCCESS
hf_Bart	0	1	18	11	1	678	0	64	SUCCESS
hf_BigBird	0	1	0	1	0	2237	0	63	SUCCESS
hf_Roberta_base	0	1	12	1	1	530	0	50	SUCCESS
hf_Longformer	12	1	0	4	1	1538	0	50	SUCCESS
hf_Whisper	0	26	4	1	1	195	0	22	SUCCESS
hf_distil_whisper	0	146	24	1	1	1075	0	122	SUCCESS

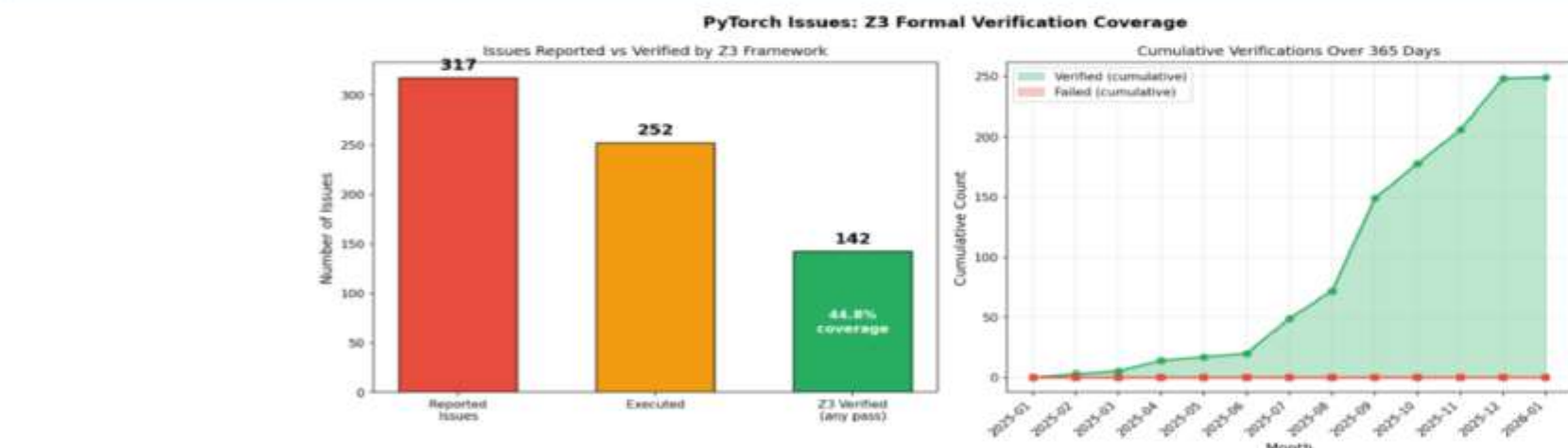
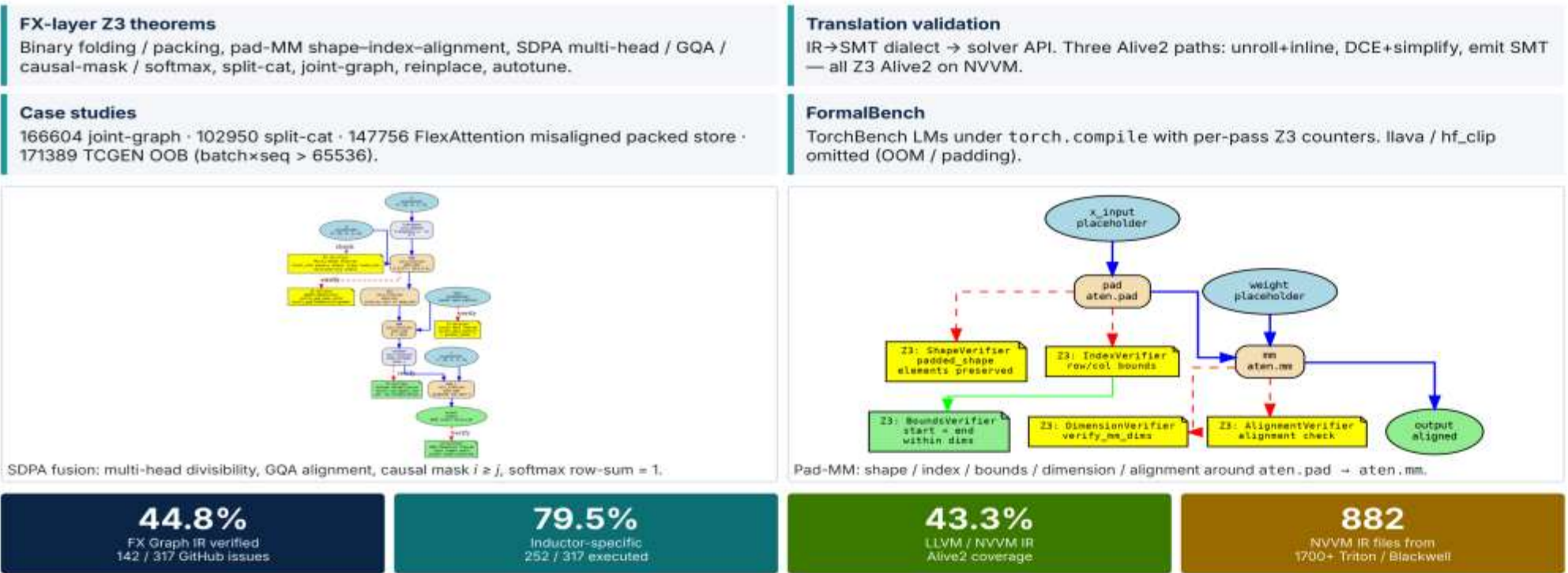


Fig. 17: FormalBench for TorchBench language models — Inductor-TV Z3 formal verification. · Fig. 18: Inductor-TV Z3 empirical study: automated formal verification of PyTorch Github issues (317 reported, 252 executed, 142 verified = 44.8%).

The underlying architecture of the framework comprises of an intensive **FX Graph IR formal verifier** for Inductor backend passes at the FX layer.

Next comes the FX Graph Capture and Lowering phase which lowers the formally verified FX Graph IR to device specific kernels — **PTX, LLVM (NVVM IR)** and **Triton IR**.

The translated NVVM or LLVM IR is made input to the custom build of **Alive** which has extensions and support of entirety of NVPTX/NVVM (LLVM) and to an extent of MLIR (NVVM dialect) translation and transformation passes. This helps to catch bugs on Inductor translated PyTorch FX graph compiled code to formally verified LLVM IR.



Slides: AIRC'26 / COMPSAC'26

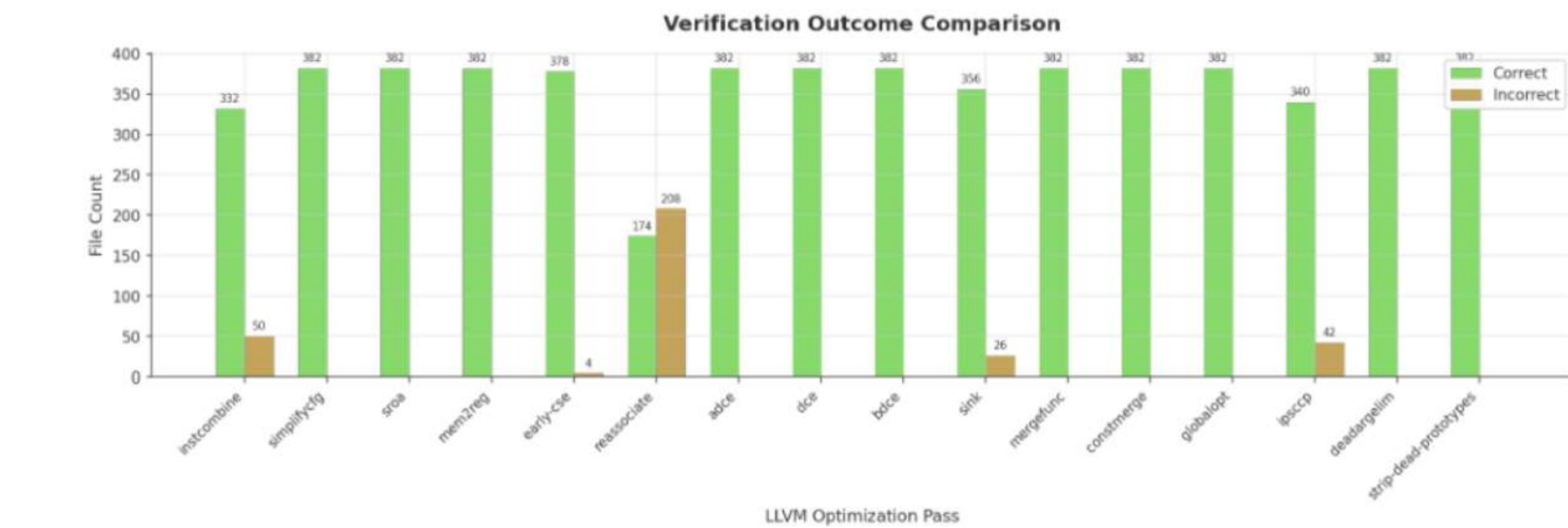


Fig. 19: LLVM Canonical Pass-Fail Estimate with customized Alive .

```
import torch
from torch.nn.attention.flex_attention import
flex_attention

flex_attention = torch.compile(flex_attention, dynamic=True)
```

Listing 2: Pytorch Inductor Lowering Issue 147756.

```
store <4 x i32> %400, ptr %336, !dbg 132
Transformation failed!
Summary:
0 correct transformations
0 failed-to-prove transformations
1 failed-to-prove transformations
0 Alive2 errors
```

Listing 3: Pytorch Issue 147756 Misaligned PTX in NVVM IR.

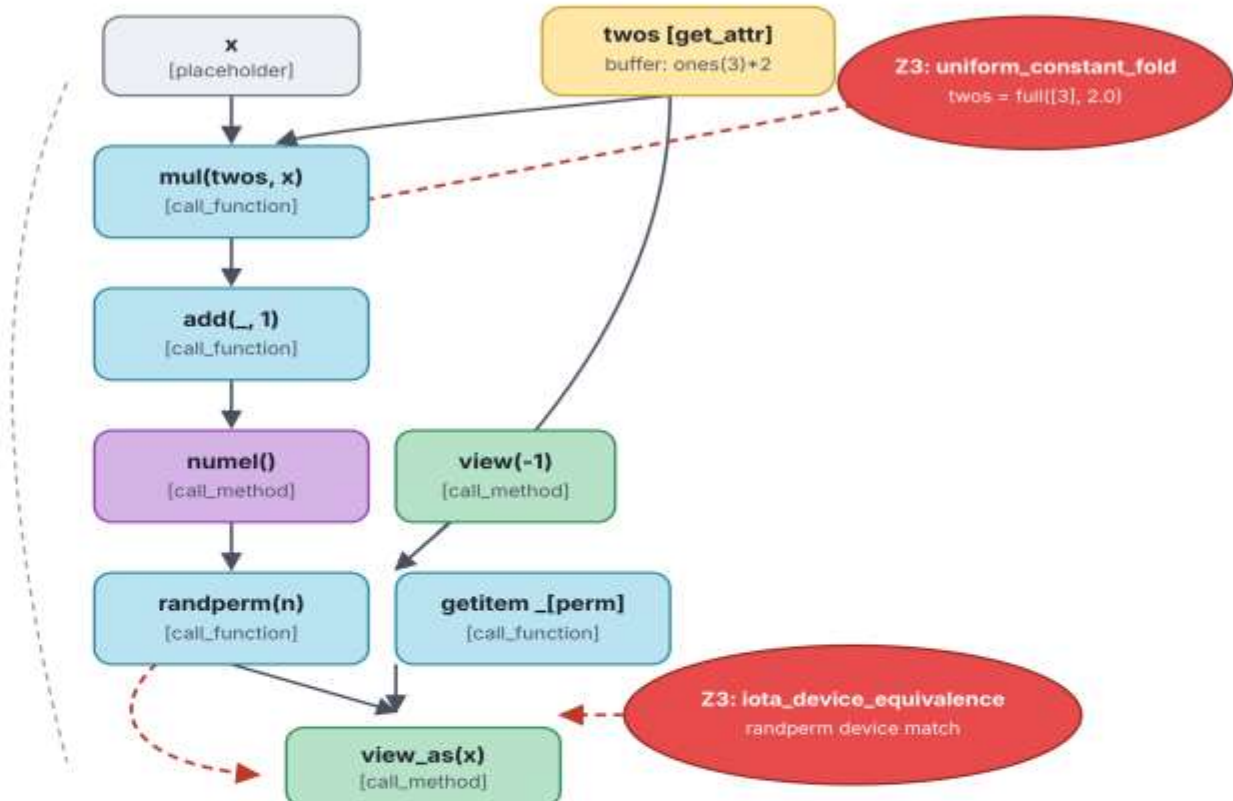


Fig. 7: Inductor-TV Z3 verification layer inside torch.fx tracing to verify Joint Graph optimization due to improper pattern matching of scalar placeholder values between tensor mutations — PyTorch Issue 166604. Red nodes inject uniform_constant_fold and iota_device_equality.

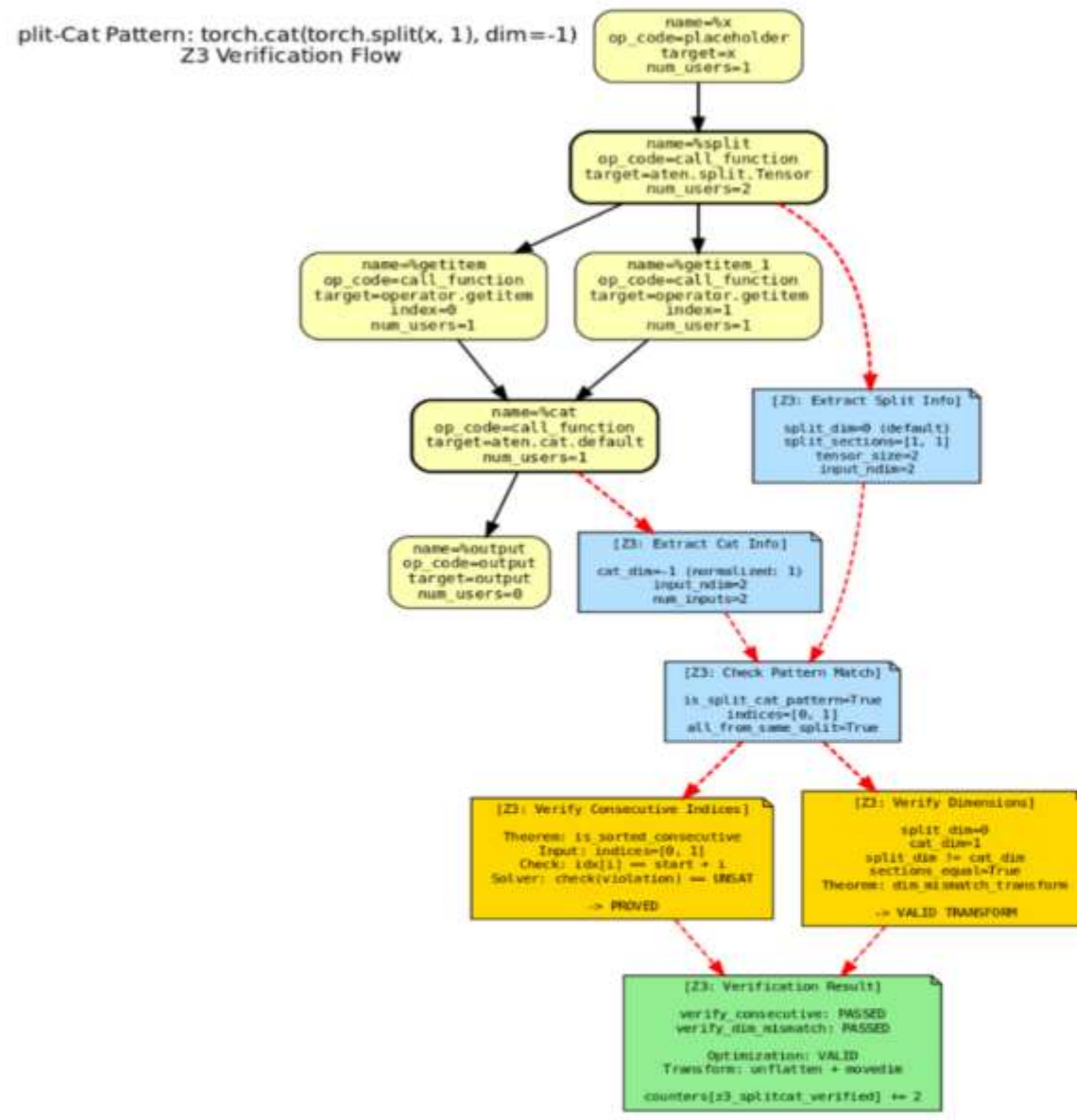
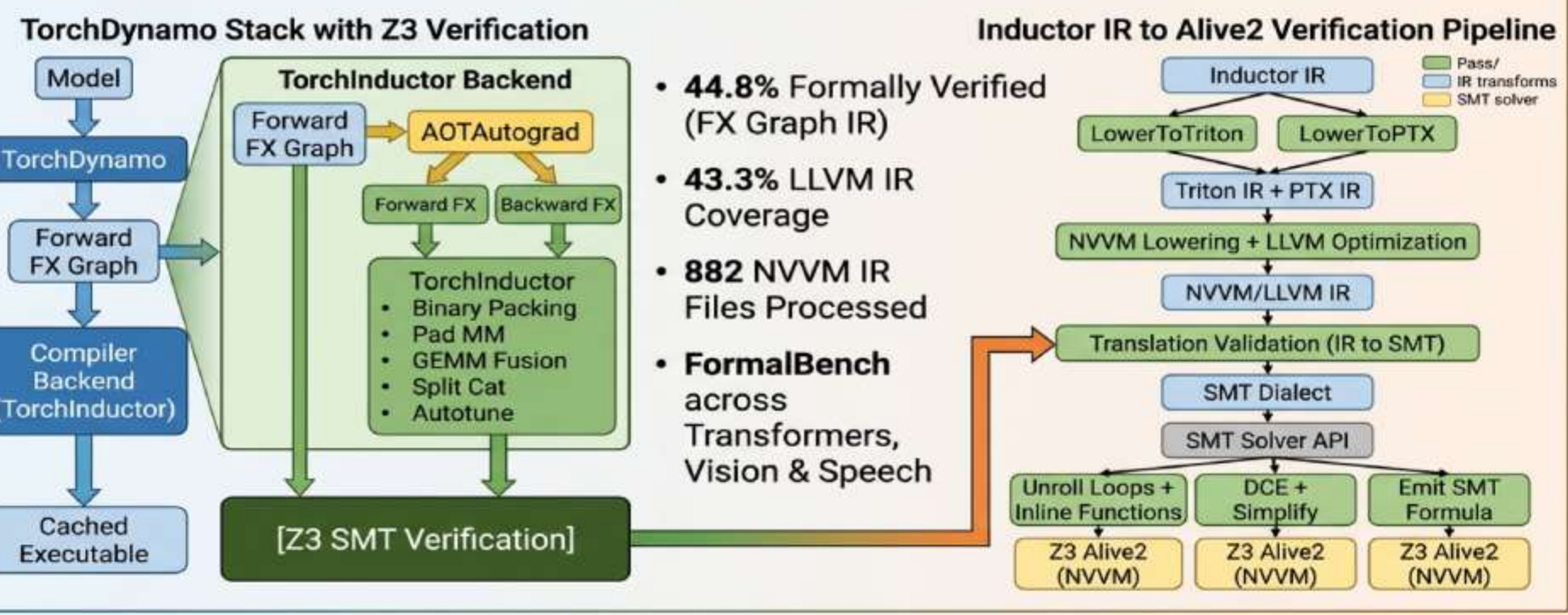


Fig. 11: Inductor-TV Z3 layer inside torch.fx tracing to verify split-cat pattern optimization for Inductor Issue 102950. Red lines inject Z3 theorem solvers on FX graph nodes.

Inductor-TV : Formal Methods for the PyTorch Compiler

Graphical Abstract — Journal of Algorithms in Complex Systems



Graphical abstract — TorchDynamo stack with in-pass Z3 on Inductor FX, then LowerToTriton / LowerToPTX → NVVM/LLVM IR → Alive2 translation validation (Journal of Algorithms in Complex Systems). · Fig. 15-16: PTX Tensor Memory (tcgen05) structure and wgmma vs. tcgen05.mma from the PTX ISA.

PTX Memory Model Formal Verifier

Definition IV.1 (Scope). A scope S defines the set of threads that can observe an operation. PTX defines:

- cta — threads in the same Cooperative Thread Array
- gpu — all threads on the GPU
- sys — all threads including CPU

Definition IV.2 (Morally Strong). Two operations X and Y are morally strong at scope S if both include scope S (or larger) in their synchronization scope.

Definition IV.3 (Causality Order). Operation X causes Y (written $X \xrightarrow{\text{causes}} Y$) if:

- 1) X and Y are by the same thread and $X \xrightarrow{\text{po}} Y$ (program order), or
- 2) X is a write, Y is a read, and Y reads from X , or
- 3) $\exists Z : X \xrightarrow{\text{causes}} Z \wedge Z \xrightarrow{\text{causes}} Y$ (transitivity)

Theorem IV.1 (Coherence). For any memory location ℓ , there exists a total order $\xrightarrow{\text{co}}$ (coherence order) over all writes to ℓ such that:

- 1) If write $W_1 \xrightarrow{\text{co}} W_2$ to location ℓ , then $W_1 \xrightarrow{\text{co}} W_2$.
- 2) If read R reads from write W to ℓ , and $W \xrightarrow{\text{co}} W'$, then R does not read from any write W'' where $W \xrightarrow{\text{co}} W''$.

Theorem IV.2 (Release-Acquire Synchronization). Let thread T_1 execute:

```
st.release.scope[ℓ] ← v
```

and thread T_2 execute:

```
r ← ld.acquire.scope[ℓ]
```

If T_1 's load reads the value v written by T_2 's store, and both operations are morally strong at scope S , then all writes by T_1 that precede the release in program order are visible to all reads by T_2 that follow the acquire in program order.

Theorem IV.3 (Fence-Induced Ordering). Let F be a fence.synchronize operation. For any two memory operations X and Y by the same thread where $X \xrightarrow{\text{po}} F \xrightarrow{\text{po}} Y$:

- 1) All threads at scope S observe X before Y .

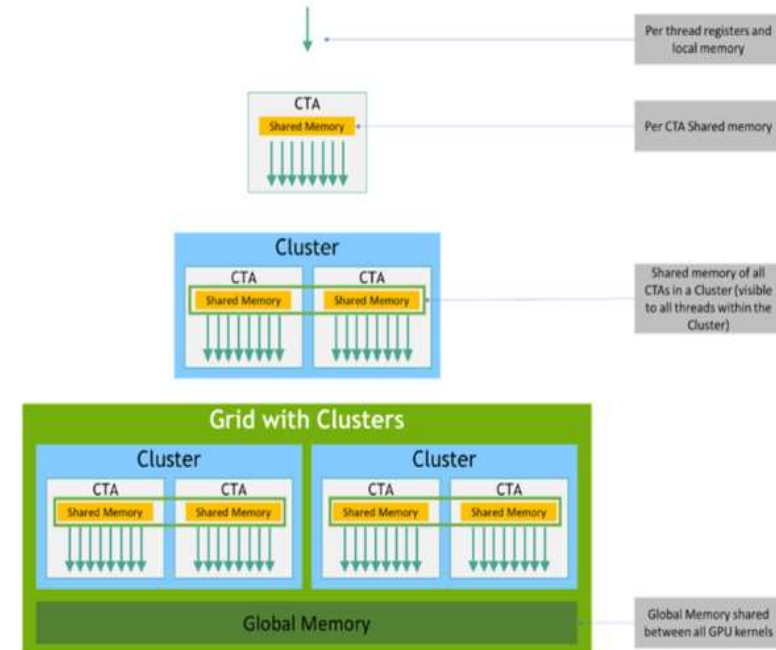


Fig. 14: PTX Memory scope and heirarchy provided in PTX ISA.

PTX Memory Model Formal Verifier — TCgen

There is a PyTorch Issue 171389, where out of bounds (UB) was noticed during Inductor compilation when $\text{batch-size} \times \text{seq-len} > 65536$ or 64 KB. This is noticed in the PTX generated which uses more than 64 KB allocation prior to performing **mad.lo** (matrix add intrinsic) and subsequently MMAs on Tensor Core Memory.

```
// Listing 16 — Issue 171389 PTX OOB, rechecked in NVVM by Alive
setp.lt.s32 np7, %r442, %0539;
setp.lt.s32 np8, %r445, %0539;
setp.lt.s32 np9, %r448, %0539;
setp.lt.s32 np10, %r443, %0539;
setp.lt.s32 np11, %r1, %0;
and.prd np5, np3, np2;
and.prd np6, np5, np4;
and.prd np7, np5, np6;
mad.lo.s32 %r446, %r442, %0445, %r439;
mma.sync.aligned.m8n8k16.row.col.f32.bf16.f32
(%r446,...), (%r488,%r490), (%r190,%r200), (%r202,%r203);
```

TCgen memory model

Encodes tcgen05 lanes 0-127 × columns 0-511 (2 KB/row = 256 KB TMEM) and the wgmma / tcgen05.mma dataflow: SMEM A,B → Tensor Cores → RF D → TMEM A,D. wgmma writes accumulator D into the register file; tcgen05.mma reads/writes TMEM (A, D) directly. The same OOB is re-checked in NVVM IR with customized Alive after NVPTX lowering. Predicates setp.lt.s32 ... 65536 are treated as the legal TMEM window; a failed bound is reported as UB before the mma.sync.aligned.m8n8k16 fires.

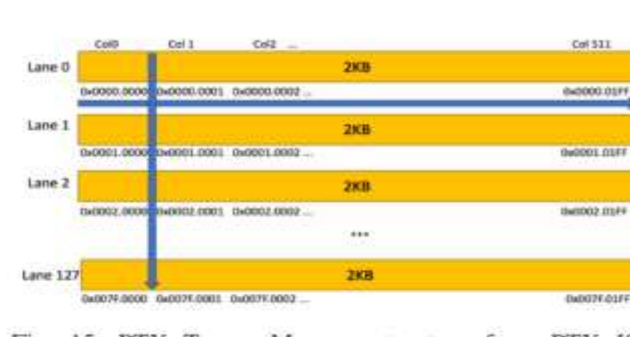


Fig. 15: PTX Tensor Memory structure from PTX ISA (TCgen05).

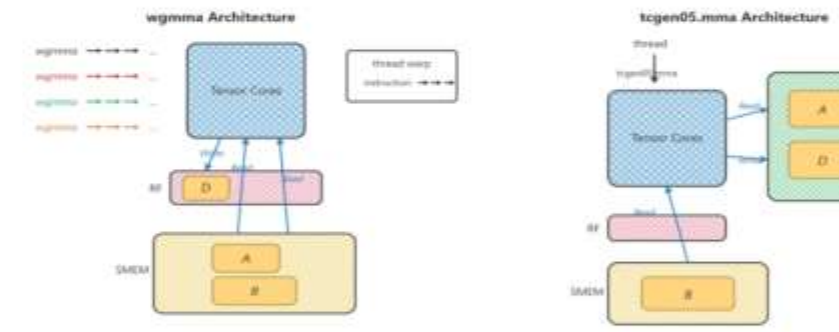


Fig. 16: PTX Tensor Memory MMA from PTX ISA.

UB fires when generated PTX allocates >64 KB before mad.lo and subsequent MMAs on Tensor Core Memory. Inductor-TV's TCgen checker flags the setp.lt.s32 ... 65536 bound and the mis-sized MMA address.

Fig. 20: NVVM IR Alive coverage by pass — 43.3% correct. Brown = incorrect (reassociate, instcombine, ipscpp, sink). 882 files.