IEEE SSE 2026 · DEBUGGING METHODOLOGY · AI ACCELERATORS

DeepView

A backend-agnostic debugging methodology for enabling large language models on emerging AI accelerators.

Diagnosing model-enablement failures that used to take days or weeks can now take minutes.

Kaoutar El Maghraoui · Jordan Sullivan · Seep Goel · Aishwariya Chakraborty · Priyanka Naik · Sahil Suneja · Flavia J. R. Beo
Rashed Bhatti · Pablo Gonzalez · Todd Deshane · Borja Godoy · Praveen Jayachandran · Alaa Youssef · Raghu Kiran Ganti

IBM T. J. Watson Research Center · IBM India Research Lab

github.com/IBM/deepview · open source

IBM RESEARCH

Every new accelerator repeats the same enablement loop

The hard part is not one bug; it is the repeated debugging work across models and backends.

– Demand for cheaper, faster inference is driving custom accelerators: **dataflow processors, neuromorphic chips, analog substrates, and ASICs.**

– Once a backend exposes itself through PyTorch, **model code often remains unchanged.**

– In practice, real models fail in ways that are hard to **localize, reproduce, and hand off.**

– The work repeats across the full **models × backends matrix.**

Days to weeks

of diagnosis time per model / backend pair

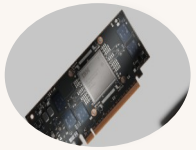
A combinatorial problem

Every backend has to be checked against every model you care about. Every model has to be re-checked on every backend. **models × backends.**

Purpose-built inference silicon is no longer hypothetical

New accelerator families are appearing across data-center, edge, and in-memory compute.

IBM's AIU accelerator family



Spyre

Data-center inference accelerator; 5nm, 32 cores, 75W PCIe card.

NorthPole

Edge inference. Brain-inspired, memory-near-compute, very low power.

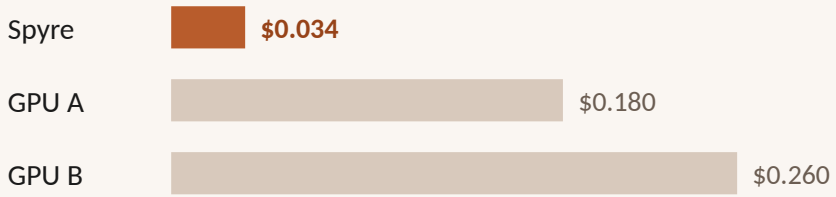
Analog AI

In-memory compute. Does the matmul where the weights live, cutting data movement.

4.2 vs 3.4

TOPS per watt (INT8). IBM-reported, workload-specific estimate, not a general GPU comparison.

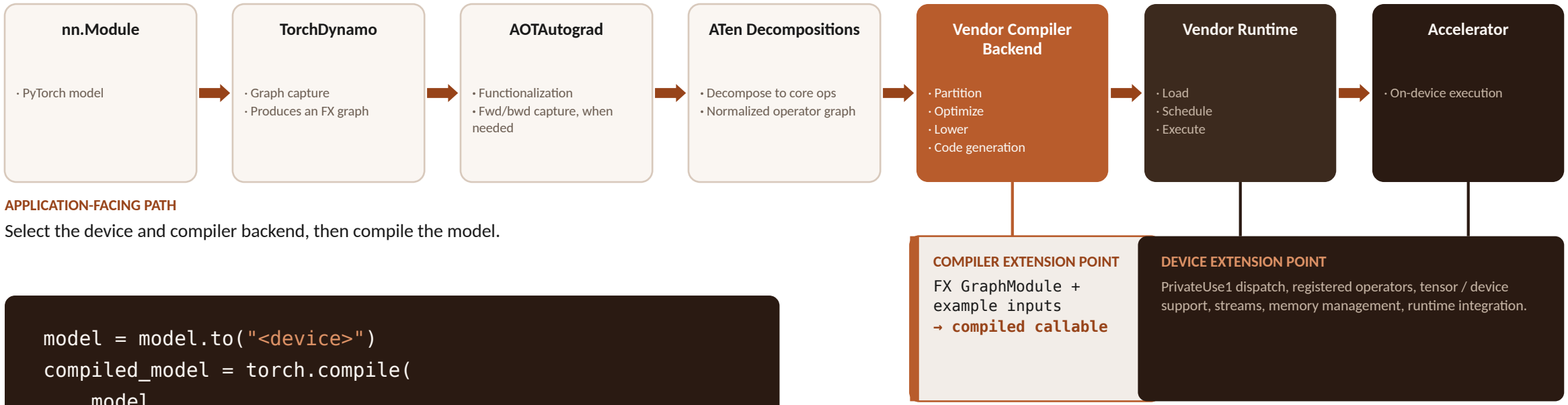
Cost per million tokens · evaluated IBM workload



Spyre specs: IBM, ISSCC 2026 / newsroom, Oct–Dec 2025. Performance and cost: IBM-reported, workload-specific estimates (Mistral-7B, INT4). Lower cost per token on the evaluated IBM workload, not a universal advantage.

From torch.compile to accelerator execution

A single API call spans graph capture, decomposition, backend compilation, runtime dispatch, and device execution.



*Compiler and device backends are **related but separate** integration surfaces.*

APPLICATION-FACING PATH

Select the device and compiler backend, then compile the model.

```
model = model.to("<device>")
compiled_model = torch.compile(
    model,
    backend="<vendor_backend>"
)
```

PrivateUse1 is an integration surface, not a user-facing device flag; it does not by itself select a torch.compile backend.

Three failure stages

Drift cannot be assessed until the model is executable, and execution depends on complete operator support.

01

COMPILE

Unsupported operators

Backend lacks an operator or decomposition required by the model.

›

02

RUN

Layer-level failure

Compilation succeeds far enough to execute, but a submodule fails during compile or runtime.

›

03

VERIFY

Numerical divergence

Execution completes, but intermediate activations deviate from a trusted CPU/GPU reference.

Resolve in order: operator coverage → layer execution → numerical agreement.

Existing tools leave the enablement loop manual

Profilers and compiler logs expose symptoms, but not a reproducible cross-backend diagnosis.

– Logs and print statements **do not scale** to hundreds of modules and backend-specific failures.

– Profilers explain timing and graph structure, **not where accelerator outputs diverge** from a reference.

– Compiler traces often identify an internal failure, **not the model layer or minimal repro** that caused it.

DeepView

Backend-agnostic diagnostics for model enablement

- One mode per failure stage
- No model-source changes
- Generates actionable artifacts: missing ops, failing layers, divergence reports

From vague failure to actionable backend issue

DeepView packages the failure so the model team and compiler team can work from the same artifact.

Before: cryptic logs and full-model reproduction. **After:** localized failure, minimal repro, structured metadata.

HOW DEEPVIEW WORKS

Three modes, one instrumentation mechanism.

Each mode targets one failure stage and emits a concrete debugging artifact.

Three modes on a shared hook

Each mode targets one failure stage and emits one concrete diagnostic artifact.

MODE 1

Unsupported operators

OUTPUT

op metadata + repro script

MODE 2

Layer localization

OUTPUT

failing submodule + repro script

MODE 3

I/O divergence

OUTPUT

per-layer MAD / cosine report

Shared instrumentation: PyTorch module hooks over the model hierarchy

Module hooks provide backend-agnostic observability

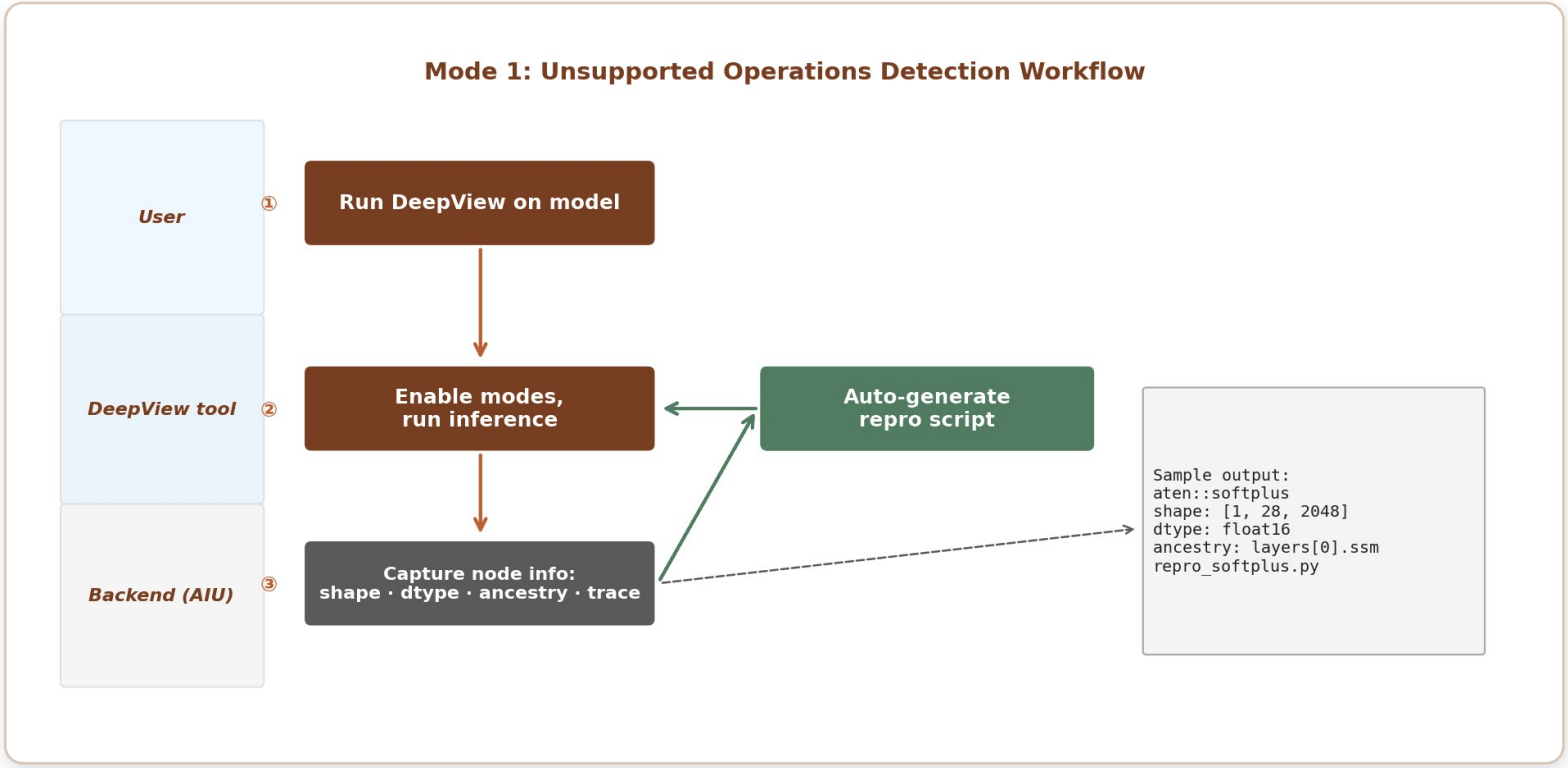
DeepView observes module inputs and outputs without modifying model source or backend internals.

- A forward hook fires **after a module successfully computes** its output.
- It receives the module, its inputs, and its outputs, **with no changes to model code or backend internals.**
- For runtime failures, DeepView uses **completed-module traces and error context** to identify the failure boundary.

```
handles = []
for name, module in model.named_modules():
    handles.append(
        module.register_forward_hook(capture(name))
    )

with torch.no_grad():
    output = model(**inputs)

for h in handles:
    h.remove()
```

Mode 1: unsupported operators

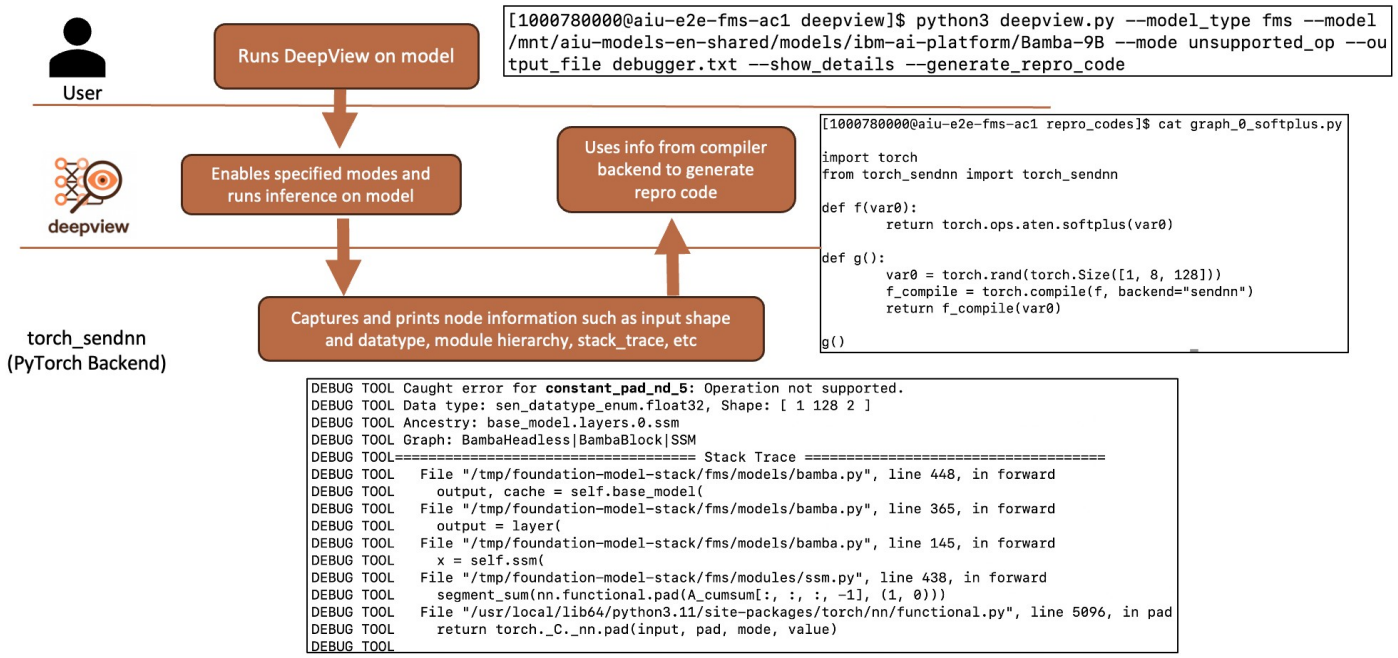
Catch the unsupported op at the FX graph level, capture everything, then generate a repro.



Mode 1 workflow: run DeepView, capture node info for each unsupported op, then auto-generate a repro script.

Mode 1 results: Bamba-9B

Bamba-9B on IBM Spyre. A state-space model, so it reaches for ops beyond the transformer set.



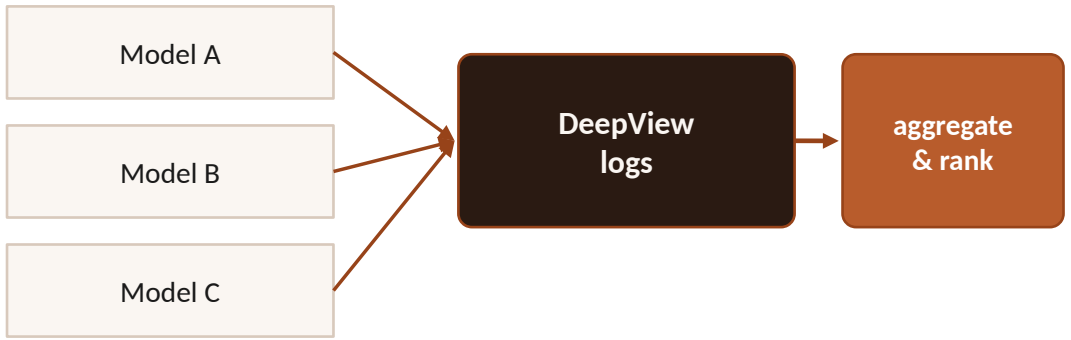
Seven ops, one run: *softplus · constant_pad_nd · copy · exp · roll · slice_scatter · select_scatter*

OP IDENTIFICATION

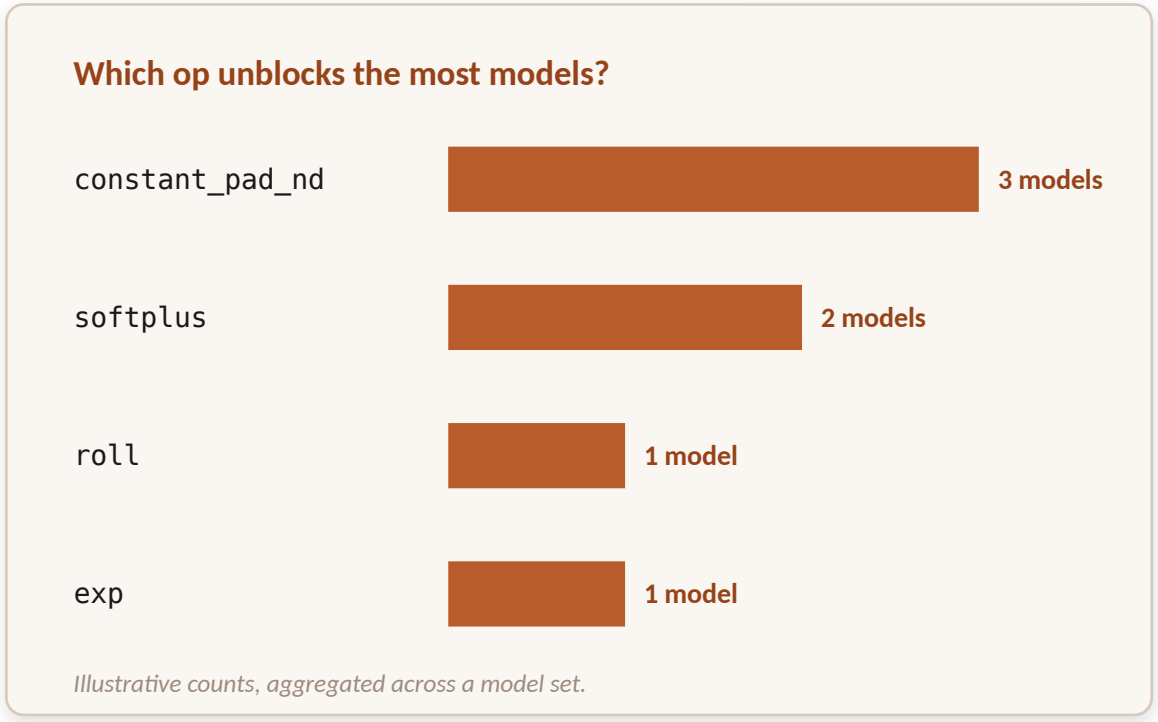
~~~5 days~~ → **minutes**

REPRO PER OP

~~30+ min~~ → **~30 sec**

This list feeds straight into the backend team's operator roadmap.
~~~

Operator prioritization

Aggregate the reports across models and you get a data-driven order for what to implement first.

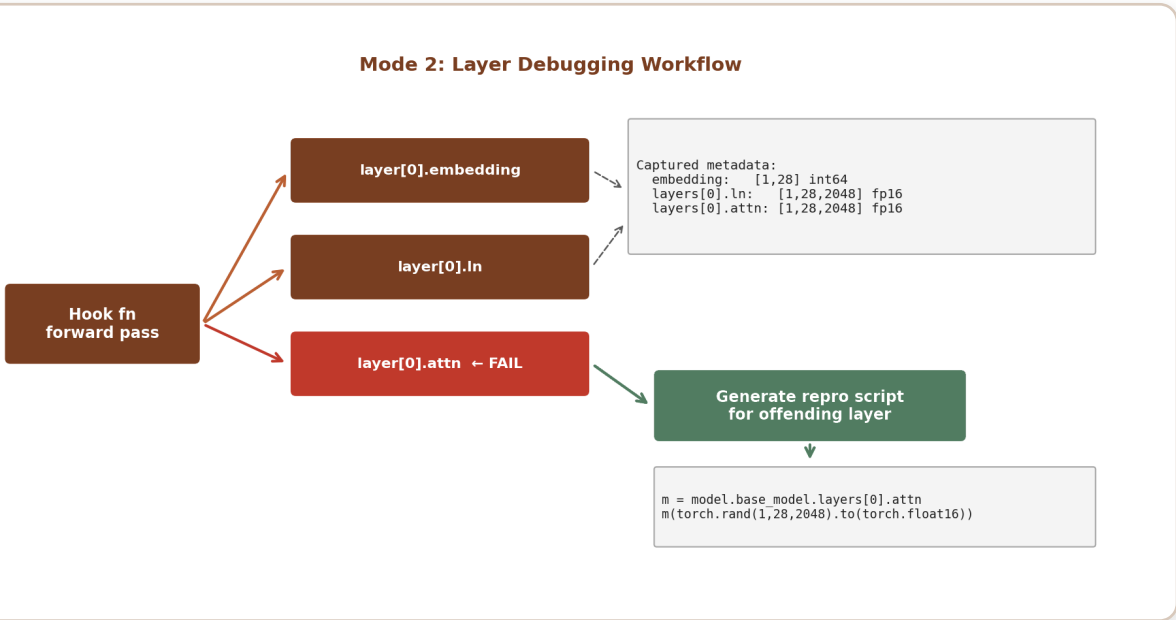


More models blocked on an op = higher priority. The compiler team implements top-down instead of guessing.



Mode 2: layer localization

The ops all exist, the model compiles, then it dies at runtime with an error that tells you nothing.



Hook every submodule, stop at the first failure, and emit a minimal repro for the offending layer.

WHAT THE COMPILER SAID

```
void PerfDSCToSDSC::fillDataDscPieces
assert(skv.second <= ddsc_lds->
  dimToLayoutSize_.at(skv.first));
```

No hint where in the model this is.

WHAT DEEPVIEW SAID · GRANITE-3.2-2B

`model.base_model.layers[0].attn`

1–2 weeks of manual hunting → **minutes**

Mode 3: numerical divergence

It runs, nothing crashes, but reduced precision and different kernels push outputs off the reference.

METRIC A

Mean absolute difference

$$MAD = (1/n)\,\Sigma\,|x_i - y_i|$$

The average element-wise gap. Tells you how big the error is.

METRIC B

Cosine similarity

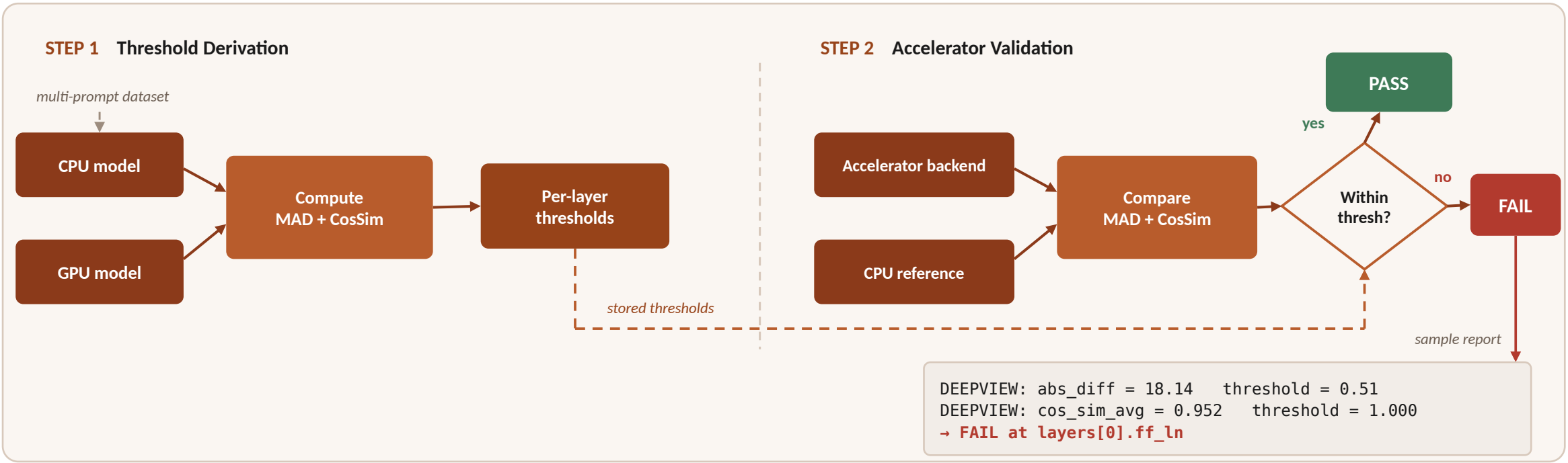
$$cos = (A \cdot B) \,/\, (\|A\|\,\|B\|)$$

Directional alignment, independent of magnitude. Tells you if the meaning is preserved.

Why both: in high-dimensional hidden states, direction carries the meaning. Two vectors can differ in size but point the same way. MAD measures how large the deviation is; cosine measures whether it still means the same thing. Neither alone is enough.

Data-driven thresholds

A fixed global tolerance fails, because natural variation differs across layers and architectures.



Step 1: derive per-layer thresholds from CPU and GPU runs. Step 2: validate the accelerator against each layer's own threshold.

Thresholds are stored keyed by model, dtype, prompt-set hash, and layer path; a weight hash invalidates stale entries. Derive once, reuse on every backend.

Mode 3 results: divergent layers

The same feedforward-norm layer fails across three different models on Spyre. That's a strong signal.

MODEL	MAD obs.	MAD thr.	cos obs.	
Granite-3.2-8B	18.140	0.509	0.952	FAIL
Granite-3.3-8B	28.968	0.351	0.813	FAIL
Mistral-7B-v0.3	6.726	0.0618	0.760	FAIL

MAD is orders of magnitude over threshold; cosine drops below 1.0, Mistral lowest at 0.760. Failure localized to layers[0].ff_ln.

Why silent drift is dangerous

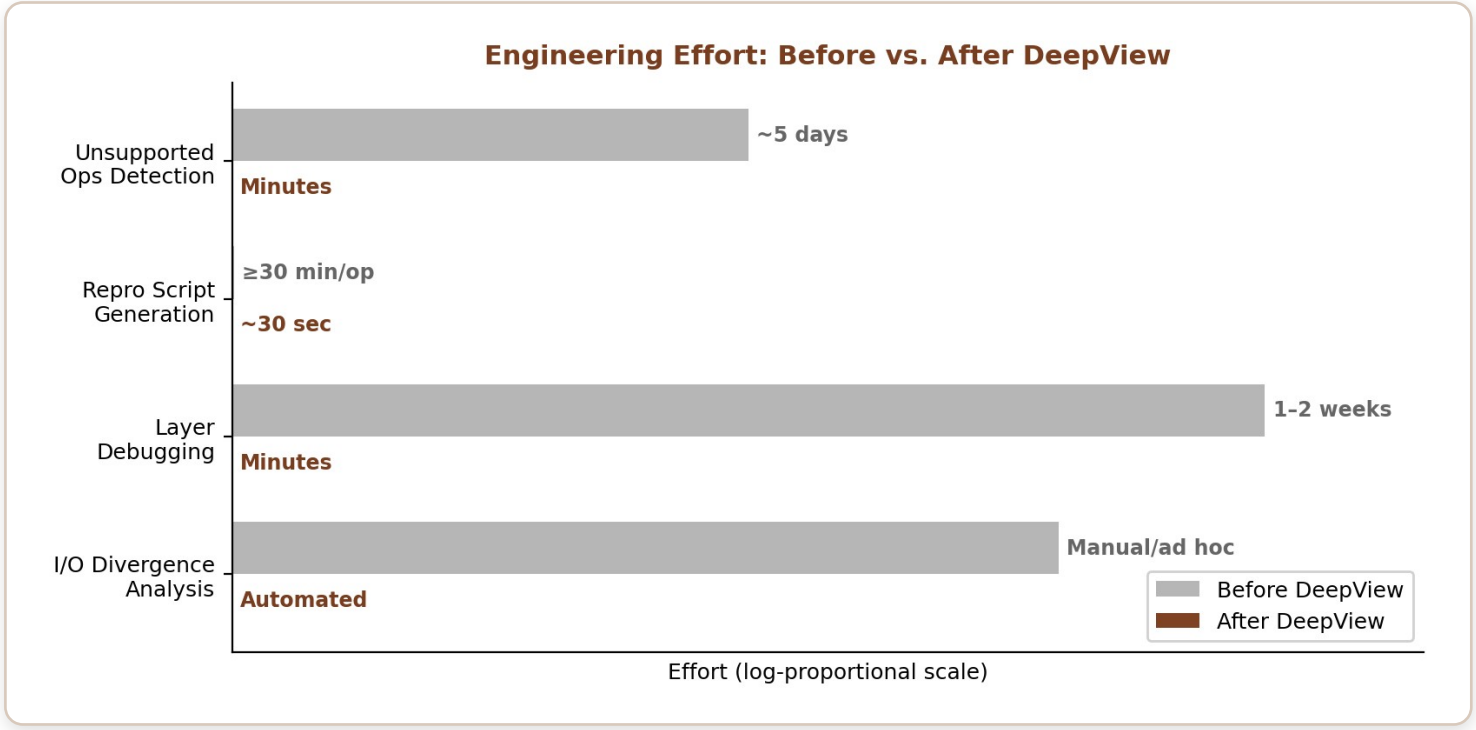
Even while a layer is failing the threshold, the model still emits a plausible answer, a working bubble-sort, for example. Nothing looks wrong from the outside. Later layers absorb some of the error, which is exactly what hides it.

The payoff: an exact layer and metric to start optimization from, instead of “the numbers seem a bit off.”

Scope: snapshots of the Spyre stack at a point in time. They show what the diagnostic surfaces, not a verdict on Spyre coverage. The stack has moved on.

Time to diagnosis

The headline across all three modes. Gray is before DeepView, terracotta is with it.



Before numbers are from internal issue trackers and engineering reports across bring-up campaigns, not a controlled study. Order-of-magnitude evidence.

CI/CD integration

This is a software-services-engineering venue, so here's what makes it usable in practice.

CLI, built for CI/CD

pip-installable, one deepview command. Flag-driven, non-interactive, writes to file. Drop it into a pipeline for automated compatibility checks.

Extensible by design

A ModelHandlerBase abstraction with handlers for Foundation Model Stack and Hugging Face. Also importable as a library. New model families need a handler, **not a change to the modes.**

`--model_type` `--mode` `--generate_repro_code` `--show_details`

Overhead · modes 1 & 2 one inference pass, light hooks. Under 5% on Bamba-9B.

Overhead · mode 3 a reference run plus a per-layer re-run, roughly 2–3× on Granite-3.2-8B. It's a bring-up diagnostic, not on the serving path.

Generalization across families

You don't have to handle every checkpoint. Under the variety, there are only a handful of shapes.

1M+ models
on Hugging Face

↓

collapse to a few model families

Transformer

Decoder LLMs: Granite, Llama, Mistral

Mixture of Experts

Sparse routing on top of the block

Mamba2 / SSM

State-space models: Bamba

→ Vision transformer

The current frontier for the tool

Model-agnostic in practice: it's all module hooks, so it already ran on Granite-Vision and SmolVLM with no model-specific work.

Related work

Prior work makes execution possible or measures performance; DeepView diagnoses model-enablement failures across backends.

Compilation and hardware backends

TVM · XLA · MLIR / IREE · Glow · torch.compile (Inductor)

Lower a model onto hardware and let vendors add backends. They make execution possible, but assume a working path and do not localize why a specific model fails.

Profiling and graph introspection

PyTorch Profiler · Kineto · Nsight · torch._dynamo.explain

Report timing, kernels, and graph structure. They are not built to compare accelerator output against a trusted reference or to emit a minimal repro.

Numerical and correctness checks

PyTorch Numeric Suite · accuracy harnesses · per-operator tests

Check end-to-end accuracy or a single operator. DeepView adds per-layer divergence with data-derived thresholds, backend-agnostic, with reproducible artifacts.

DeepView's contribution: operator coverage, layer localization, and per-layer numerical divergence in one backend-agnostic, hook-based workflow that emits reproducible artifacts.

Future work: deeper failure attribution

Extend DeepView from module-level localization to compiler, kernel, and runtime diagnosis.

Kernel-level localization

Map a failing or divergent module to the generated kernel or fused region.

Capture:

- FX node or graph partition
- generated kernel name
- input shapes and dtypes
- fusion boundary
- backend error or numerical metrics

Compiler-pass bisection

Checkpoint intermediate compiler IR and outputs across the lowering pipeline.

Identify the first pass that introduces:

- an invalid graph transformation
- a shape or layout mismatch
- unsupported lowering
- numerical divergence

Runtime and system failures

Extend diagnostics beyond operator correctness to:

- memory allocation failures
- graph breaks and recompilation
- dynamic-shape failures
- distributed execution errors
- latency and throughput regressions

Backend portability: keep the diagnostic workflow unchanged while switching the target device and compiler backend configuration.

Thank you.

Three failures on a new accelerator, one backend-agnostic framework, days and weeks down to minutes.

missing ops **failing layers** **silent drift**

Kaoutar El Maghraoui · github.com/IBM/deepview · open source · *feedback welcome!*

References

- El Maghraoui et al., DeepView. IEEE SSE 2026 · github.com/IBM/deepview
- IBM Foundation Model Stack & aiu-fms-testing-utils · github.com/foundation-model-stack
- Aggarwal, Hinneburg & Keim, On the surprising behavior of distance metrics in high-dimensional space, 2001