



AWS Builder Center

From torch.profiler to Hardware Cycles

A Practical Profiling
Playbook for AWS
Trainium

Esha Lakhotia and Pinak Panigrahi
from Annapurna Labs, Amazon

Agenda

- What is performance engineering and why is it necessary
- TorchNeuron on Trainium: the PyTorch you know
- Capturing an end to end profile with torch.profiler
- Visualize bottlenecks with Neuron Explorer UI
- Explorer output and agentic development
- Live demos

AWS Builder Center



AWS Trainium

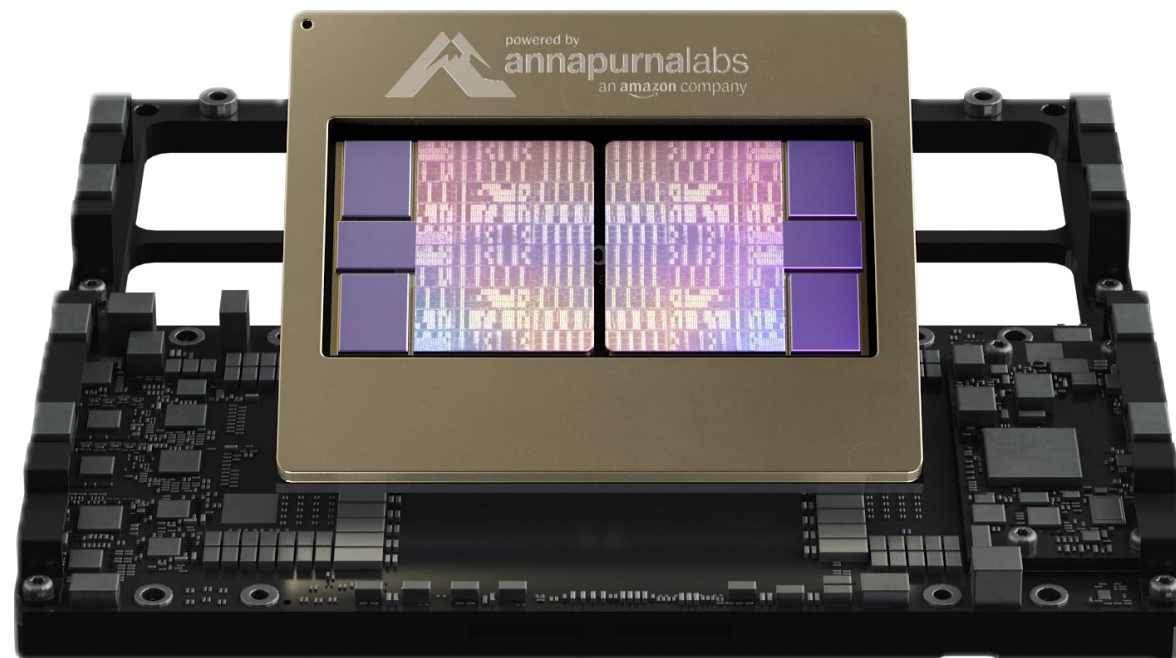
The hardware your model runs on, and how PyTorch gets you there.

AWS Trainium: Purpose-built chip for large scale-model training and inference.

Native PyTorch: PyTorch now runs natively on Trainium with zero code changes.

TorchNeuron: The native PyTorch backend for Trainium.

Why it matters: Easier access for researchers who want to experiment & innovate, and customers who want the performance Trainium offers to enable them to reach their performance goals.



Where is the time going, and what do I change?

Your model runs correctly, but it's slower than the chip can run it.



You pay for every idle cycle. Engines wait on data, memory, or a dependency, and it shows up in throughput (tokens/s, MFU), latency, and cost per token.

Spotting slow is easy. Explaining why is hard. The bottleneck hides in the layers between your model and the chip: a framework op, a runtime call, a data transfer, a kernel, a memory dependency. Today, finding it is manual work for the few experts who can read a hardware trace.

How we answer it today

1 Capture

torch.profiler + NeuronConfig

2 Read

Neuron Explorer, framework to hardware

3 Act

Fix it yourself, or hand it to an agent

TorchNeuron Extension Points

Standard backend

PrivateUse1

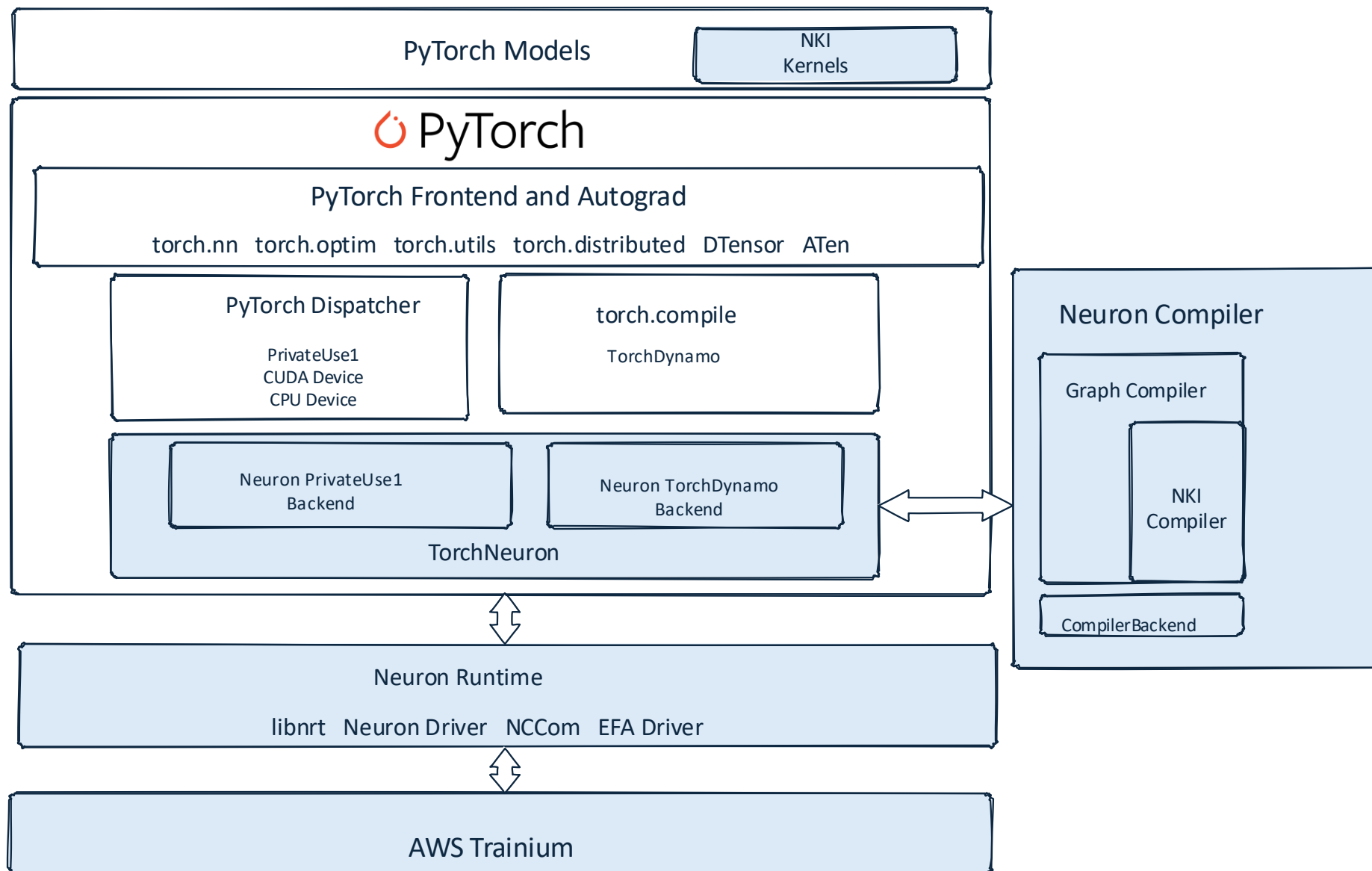
Standard execution modes

eager and torch.compile

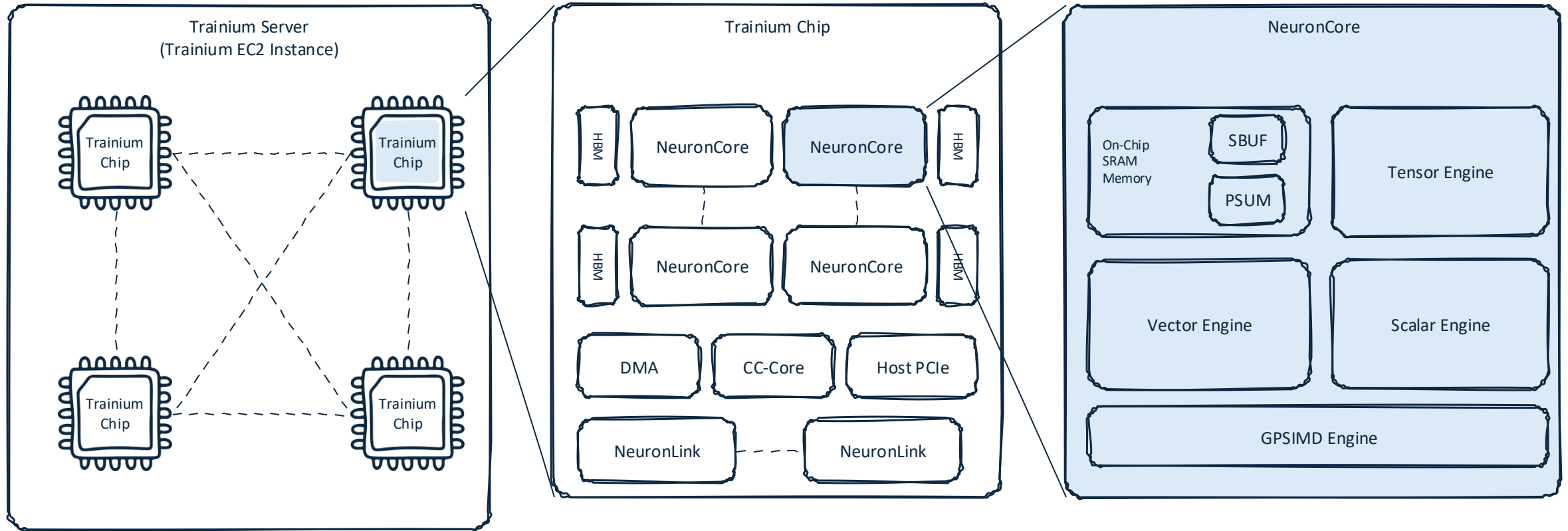
Standard APIs

torch.nn · torch.tensor · torch.autocast · torch.distributed · torch.profiler

TorchNeuron Stack



Inside Trainium Device and NeuronCore



Capture a Profile

```
3
4 from torch.profiler import profile, ProfilerActivity, schedule
5 from torch_neuronx.profilig import NeuronConfig, ProfileMode, NeuronProfiler
6
7 # Placeholder model/input – swap in your real model and data.
8 model = nn.Linear(128, 128)
9 x = torch.randn(8, 128)
10
11 # DEVICE = on-chip Trainium activity (what Explorer shows);
12 # RUNTIME = host-side runtime calls.
13 nc = NeuronConfig(
14     modes=[ProfileMode.DEVICE, ProfileMode.RUNTIME],
15     profile_output_dir="./profile_output",
16 )
17 exporter = NeuronProfiler(nc)
18
19 with profile(
20     # PrivateUse1 is the dispatch key Neuron registers under, so this captures device ops.
21     activities=[ProfilerActivity.CPU, ProfilerActivity.PrivateUse1],
22     experimental_config=nc,
23     on_trace_ready=exporter.export_trace,
24     with_stack=True,
25     # Skip 2 steps, warm up 1, profile 1 – avoids capturing startup/compile overhead.
26     schedule=schedule(wait=2, warmup=1, active=1),
27 ) as prof:
28     for _ in range(4): # must cover wait + warmup + active
29         model(x)
30         torch.neuron.synchronize() # drain async queue before export
31         prof.step()
```

Use PyTorch Profiler Built-In Tools

`torch.profiler.record_function("name")`

Wrap any code range to label it; your own regions then appear by name in the table and trace.

`with_stack=True`

Attach the Python / TorchScript source line to each op, so a hot op maps straight to its call site.

`record_shapes=True`

Record operator input shapes; pair with `key_averages(group_by_input_shape=True)` to split a hotspot by the shapes it ran on.

`profile_memory=True`

Report tensor memory allocated and released per op, to find what is driving your HBM footprint.

`key_averages()`

Aggregated per-op stats. On Neuron, sort by device time to surface hotspots

```
print(prof.key_averages().table(sort_by="self_privateuse1_time_total", row_limit=4))
```

Name	# Calls	Self CPU	Self NEURON	% NEURON
aten::mul	3	60.9us	1164.5us	49.5%
aten::add	1	26.6us	434.6us	18.5%
aten::rsqrt	1	15.6us	415.0us	17.6%
aten::mean	1	21.1us	340.8us	14.5%
Self NEURON time total: 2354.9us				

Introducing Neuron Explorer

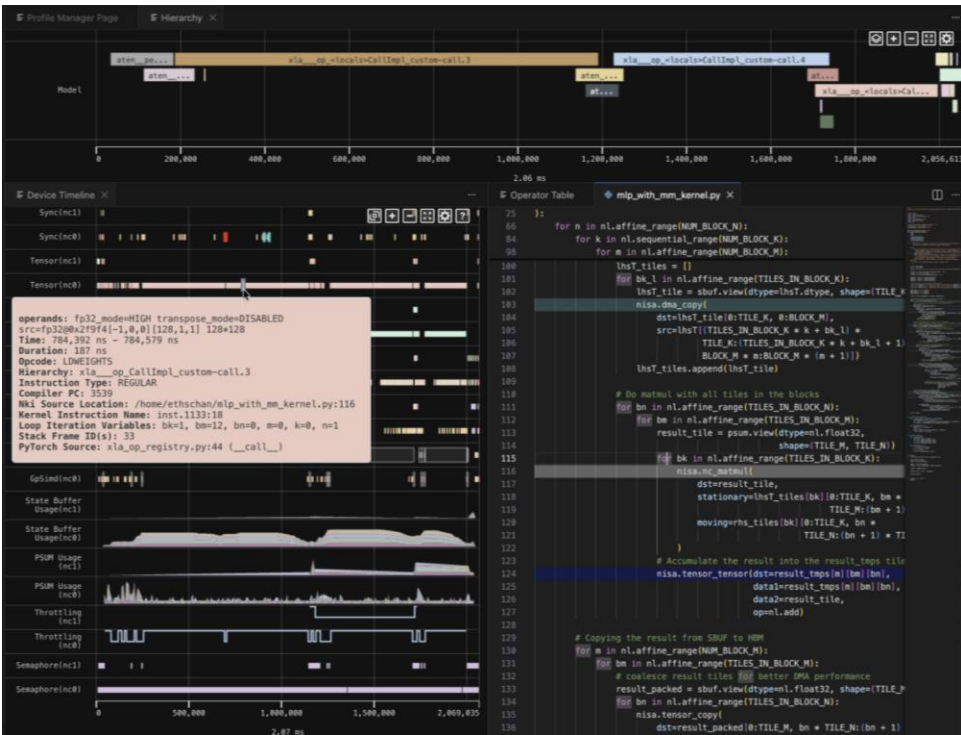
The developer toolsuite for performance analysis on AWS Trainium — from PyTorch framework layers down to cycle-accurate hardware, with context intact.

Layout customization: widget-based workspace via the Layout Manager

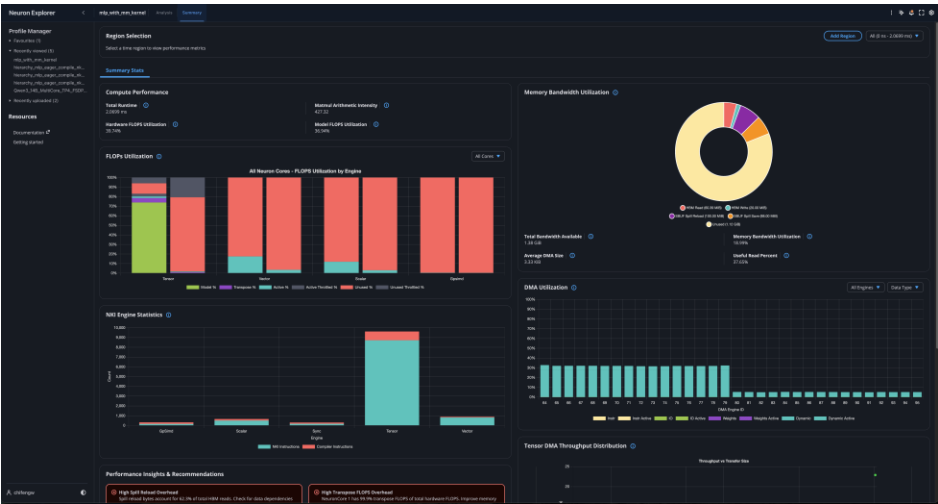
Summary view: profile overview at a glance — totals, hotspots, utilization

Main view: primary UI tying framework, system, and device together through multiple viewers

Search: find across traces, tensors, and events



One workspace: hierarchy, device timeline, and source — connected



Summary view — profile overview at a glance

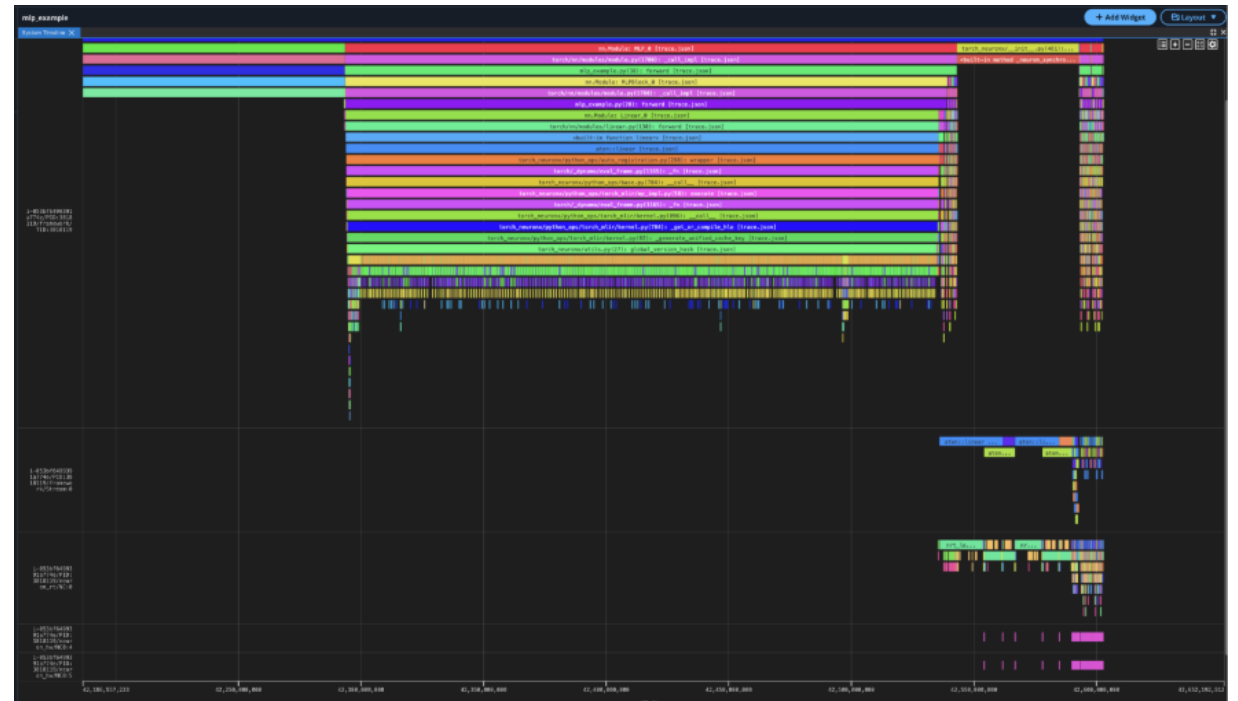
System Profiling

The host-side view: Neuron Runtime API calls, framework execution, CPU and memory activity.

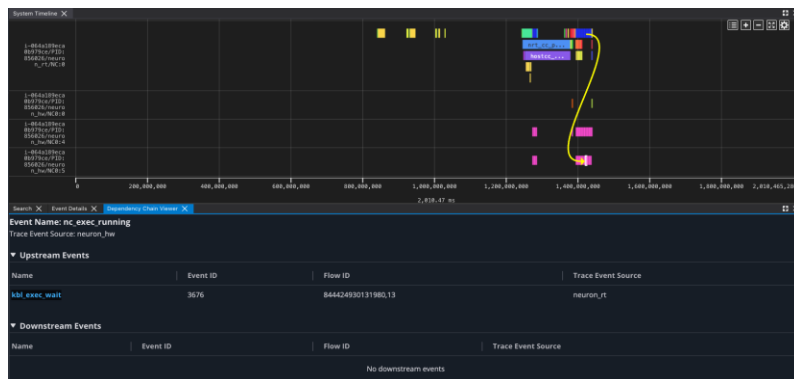
System trace view — zoom, pan, filter, group, search to reveal comm delays, sync waits, inter-rank dependencies

Dependency chain view — bidirectional flow between Runtime APIs and ATen operators

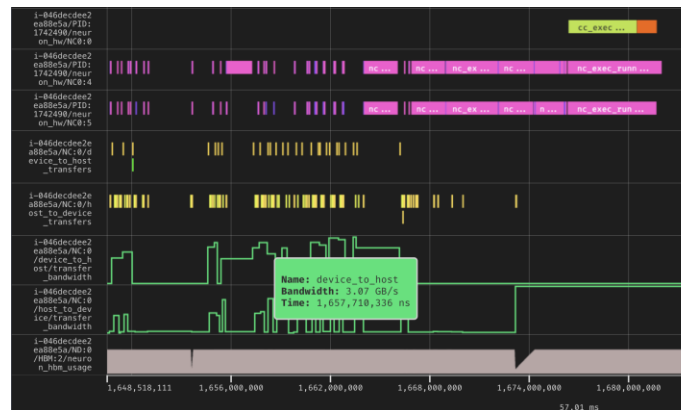
Source code linking — jump from any system event to the source line



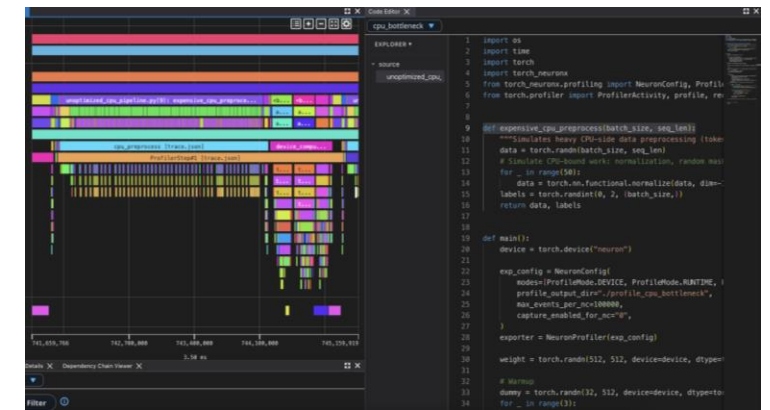
System timeline — framework + runtime activity



Dependency chain viewer



Host ↔ device transfers



Source-code linking

Device Profiling

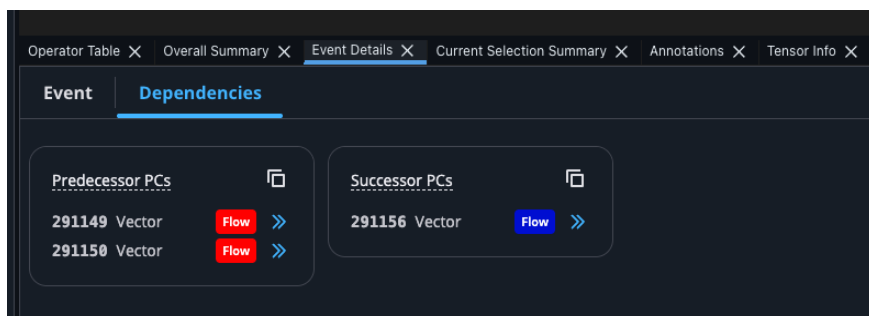
The cycle-accurate view of how your model runs on the NeuronDevice.

Device trace view: per-engine activity across DMA and compute: Tensor, Vector, Scalar, GPUSIMD

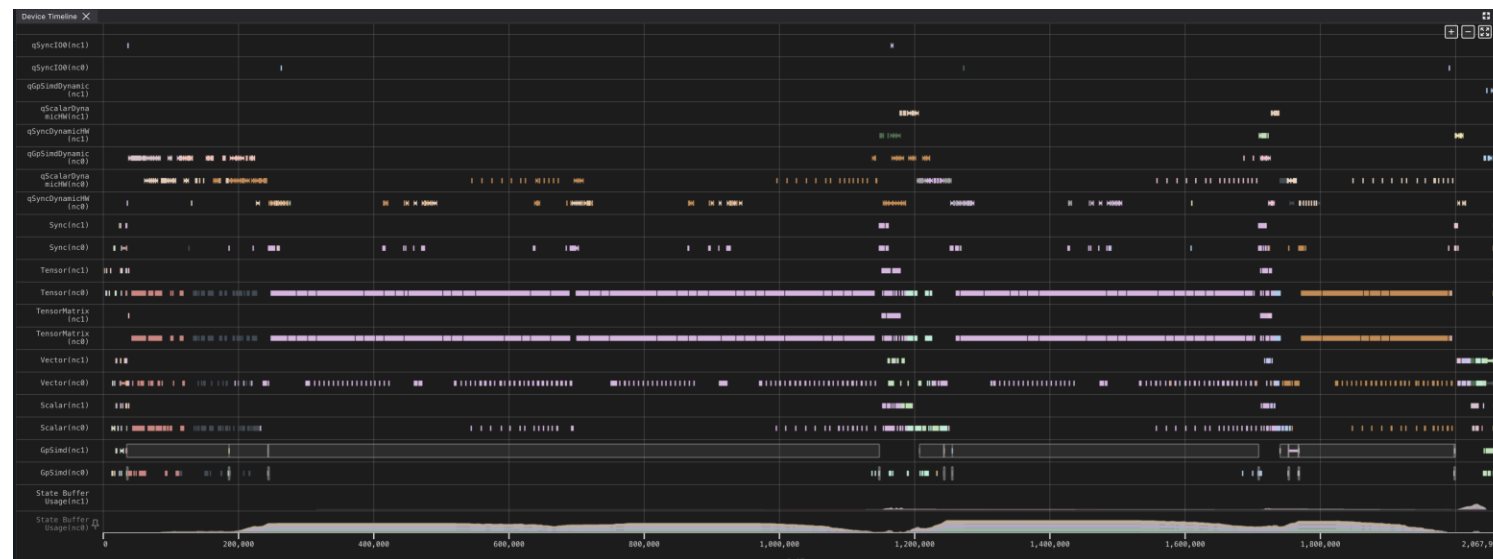
Device dependencies: how on-device events chain and where engines stall

Source code linking: hardware event → source line

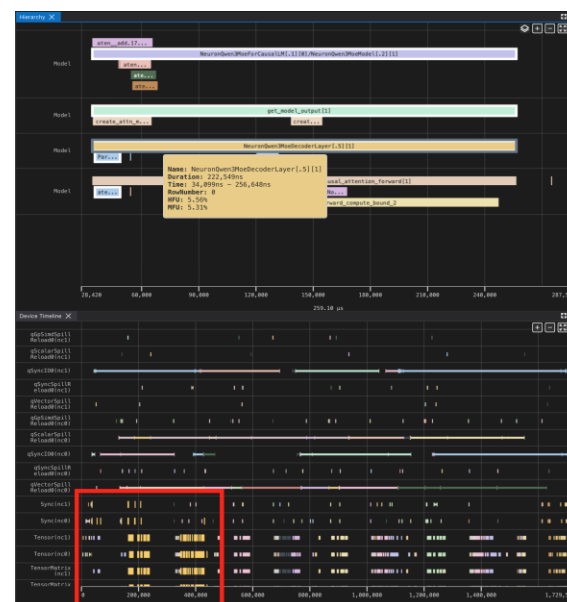
NKI regions & annotations: kernel regions and in-kernel annotations on the device trace



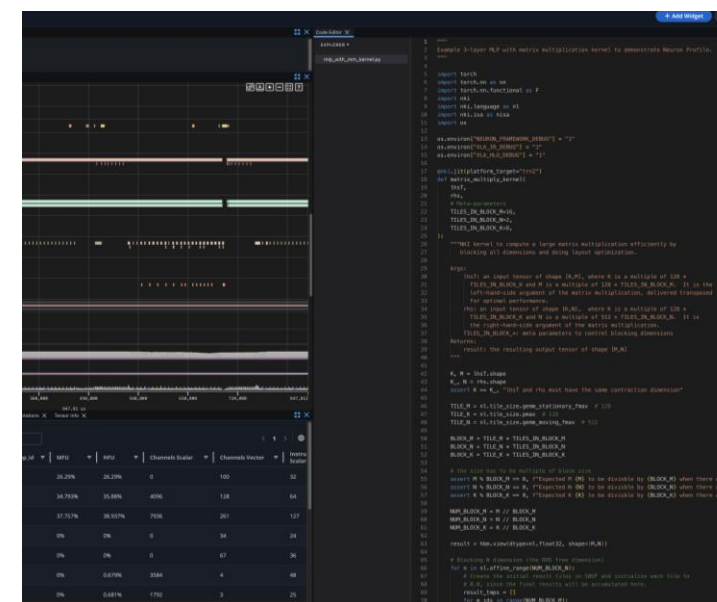
Event dependencies: predecessor / successor PCs



Device Trace View: activity on the DMA and compute engines, Hardware FLOPs Utilization, memory utilization over time, etc



NKI engine regions



Source code linking

Live Demos

Find host and device bottlenecks
with Neuron Explorer UI



AWS Builder Center

Beyond the UI: Explorer Output

Every profile also lands as structured output; the same data behind every Explorer view

One schema, every layer. Framework, HLO, and device events are all tables you can join down to per-instruction and per-DMA rows.

Query it your way. Open it in pandas or DuckDB with plain SQL, or ask in natural language

```
neuron-explorer view --show-profile-schema
```

Kind of data	What one row represents	Example tables
Timeline of events	Each is a discrete event, with a start and end timestamp (or a single timestamp).	Instruction, DmaPacket, DmaPacketAggregated, SemaphoreUpdate, Throttle, Error, CcOp, ActiveTime
Time-series samples	Each is a value at a point in time.	DmaUsage, HbmUsage, PsumUsage, SbufUsage, CpuUsage, HostMemUsage
Dependency-graph edges	Each is a directed edge between rows in timeline tables (e.g. an instruction and the DMA it triggered).	Flow
Hierarchical aggregation	Each is a node in the compiler IR hierarchy (Framework → HLO → Penguin → BIR → Instruction), with rolled-up statistics.	FrameworkInstruction, HloInstruction, PenguinInstruction, BirInstruction, FrameworkNode
Aggregated summary	Each is a summary metric computed across the whole profile (or by some key).	Summary, OpcodeSummary, ThrottleSummary, HbmUsageSummaryByType
Reference / lookup	Each is a entity in the model or execution that other tables join to them by id to enrich rows without duplicating the detail.	TensorInfo, DmaQueuesInfo, CcStream, StackFrame, KernelInstructions
Profile metadata	Each describes a piece of profile metadata.	Metadata, NeffHeader, SystemProfileMetadata, ExecutionInfo
Diagnostics	Each is a warning emitted during ingestion. Worth checking first, rows here can indicate a data-quality problem.	Warning

Beyond the UI: Neuron Agentic Development

Performance work isn't limited to the UI, agents and engineers can act on the same trustworthy data.

Agents

Agent	Description
neuron-nki-agent	Unified NKI kernel development agent. Full lifecycle: writing kernels from PyTorch/NumPy/natural language, debugging compilation errors, profiling performance, optimizing bottlenecks, migrating between API versions, analyzing Perfetto traces, and NKI documentation lookup.
neuron-nki-writer-agent	NKI kernel authoring and modification. Translates from PyTorch/NumPy/natural language, adds shape/dtype support, refactors tiling strategies, and implements new features following Beta 3 API patterns.
neuron-nki-debugger-agent	Autonomous NKI kernel compilation error debugging. Analyzes compiler errors, searches documentation and code examples for fixes, applies corrections following simplicity over performance, and validates fixes.
neuron-nki-profile-analysis-agent	Profile and analyze NKI kernels on Neuron hardware. Captures execution traces, computes performance bounds, identifies bottleneck engines, and runs investigations to localize inefficiencies to NKI source lines.
neuron-framework-autoport-agent	A model porting agent to port GPU-compatible models to functionally accurate implementation on Neuron. Executes the full porting workflow including architecture analysis, implementation, compilation, inference testing, and validation.

Skills

Skill	Description
neuron-nki-writing	Write and modify NKI kernels. Covers new kernel creation from PyTorch/NumPy/natural language, editing existing kernels, adding shape/dtype support, refactoring tiling strategies, and implementing new features.
neuron-nki-debugging	Debug NKI compilation errors on Neuron hardware.
neuron-nki-docs	Research NKI documentation for API lookups, tutorials, error codes, and architecture details.
neuron-nki-profiling	Profile NKI kernels to analyze performance on Neuron hardware.
neuron-nki-profile-querying	Query and analyze NKI kernel profile data from neuron-explorer parquet files via SQL and Python.
neuron-framework-autoport	Port a GPU compatible model to NeuronX Distributed Inference for AWS Trainium/Inferentia. Handles the full workflow including architecture analysis, NeuronX implementation, compilation, inference testing, and accuracy validation.
neuron-framework-equivalence	Verifies functional equivalence between two implementations of the same model using a hierarchical algorithm.

Demo

Agentic Querying with
Explorer Output



AWS Builder Center



Claude Code v2.1.304
Opus 5.5 · Amazon Bedrock
~/Desktop/pytorch-conf

> /clear



> █

▸▸ auto mode on · 1 shell · ↓ to manage

enter

The Profiling Playbook

- Capture with `torch.profiler` + `NeuronConfig` — no separate tooling.
- Use PyTorch built in tools like `key_averages()` and `record_function()`
- Use Explorer UI to view and analyze end to end profiles
- Use Explorer Output and NAD to agentically/programmatically query metrics and find bottlenecks
- Fix, recapture, compare: confirm the gap actually closed.

Thank you. Questions?

Try it yourself

`pip install torch-neuronx`

github.com/aws-neuron/torch-neuronx · open source

Native PyTorch on Trainium

<https://awsdocs-neuron.readthedocs-hosted.com/en/latest/frameworks/torch/index.html>

Get started with Neuron Explorer

<https://awsdocs-neuron.readthedocs-hosted.com/en/latest/tools/neuron-explorer/index.html>

Find the team **at the AWS booth in the Community Expo**



torch-neuronx on GitHub