Amazon's Trainium Takes a Strike at the Training King

Trainium Best for Inference? But now good for Training too, Native Trainium PyTorch, High Quality CI, TorchTitan, GPU Impossible Profiler

10 HOURS AGO · PAID

129 1 Share

Trainium has already proven itself in the realm of inference. Anthropic is running production inference and RL rollouts on Trainium2 and Trainium3. Historically, Trainium has been great for inference but not the best fit for research training. In inference, the model architecture changes at a much slower pace compared to training, where researchers iterate on many different experiments and ablations of model architecture and need a stack that covers a broad API surface. In the past, that is where Trainium fell short.

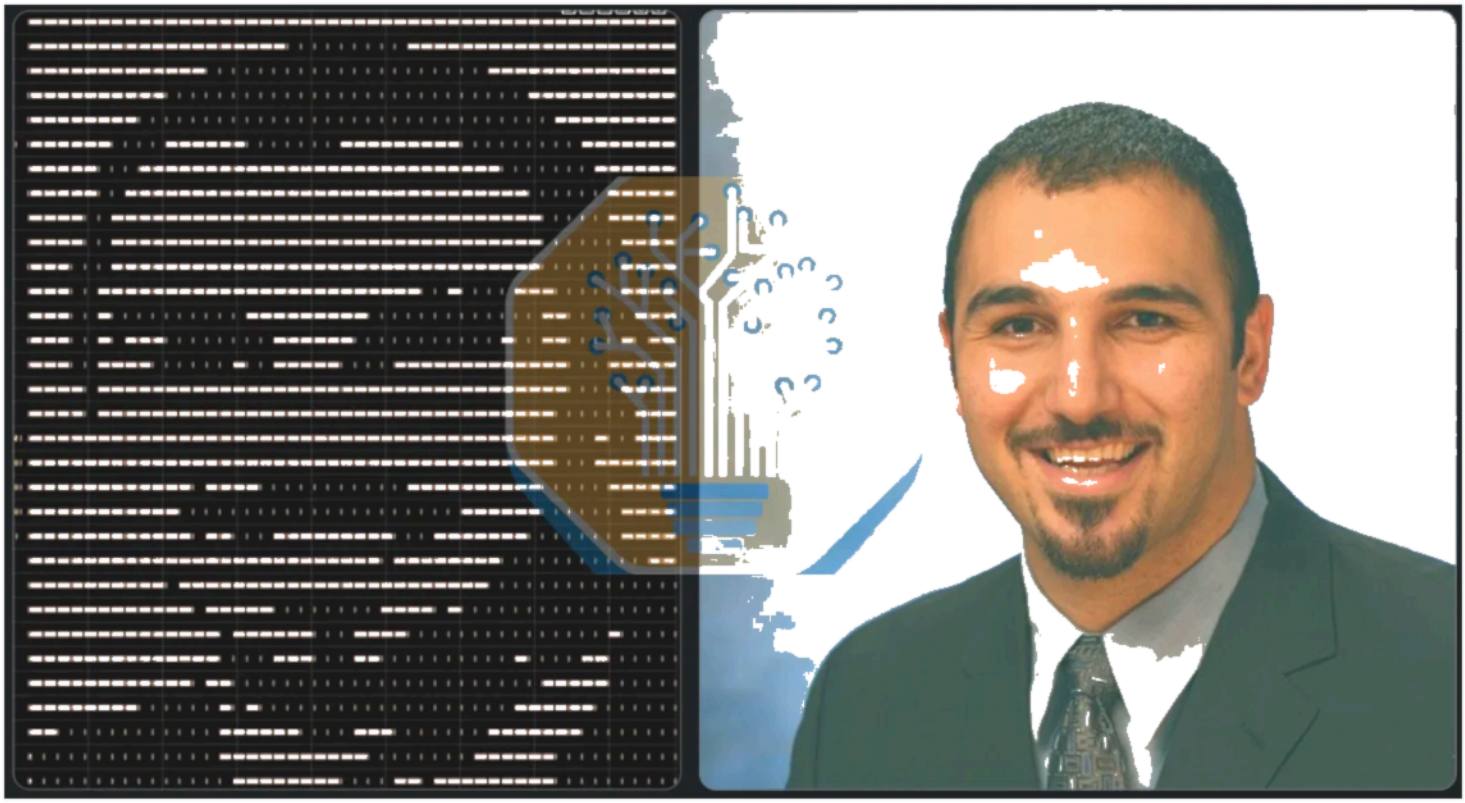
The AWS Neuron team is aggressively working to change this pitfall with the biggest shift being moving towards a first-class, native PyTorch training experience. Gone are the days of sprinkling xm.step() through your training loop and wrestling with a bespoke XLA-style execution model. Trainium now runs PyTorch the way PyTorch users expect, in eager mode, with torch.compile and the standard APIs.

The SemiAnalysis engineering team has been trying out the Trainium3 racks for the past couple of weeks and has found the experience to be great – we love the overall direction of where it is heading. To be clear, it isn't perfect yet, and we have personally run into a couple of bugs ourselves. It's nowhere close to the maturity of the CUDA ecosystem, but the direction is right. Their team is making rapid progress with the help of agentic coding.

Furthermore, unlike other accelerator companies, where we had to fight tooth and nail to get their engineers a reasonably stable CI fleet, Trainium uses up to 80,000 physical cores allocated towards automated quality control.

In this article, we walk through our experience on the Trainium3 TorchTitan and TorchNeuron stack, along with our experience of Trainium's Neuron Explorer profiler,

which is way better than both Nvidia's and AMD's profilers. We created a bunch of demos to show how Trainium's profiler is "GPU Impossible."



Visualizing Nafea Bshara, VP at AWS In Trainium Profiler Showcasing Trainium's Finegrain Profiler. Source: SemiAnalysis

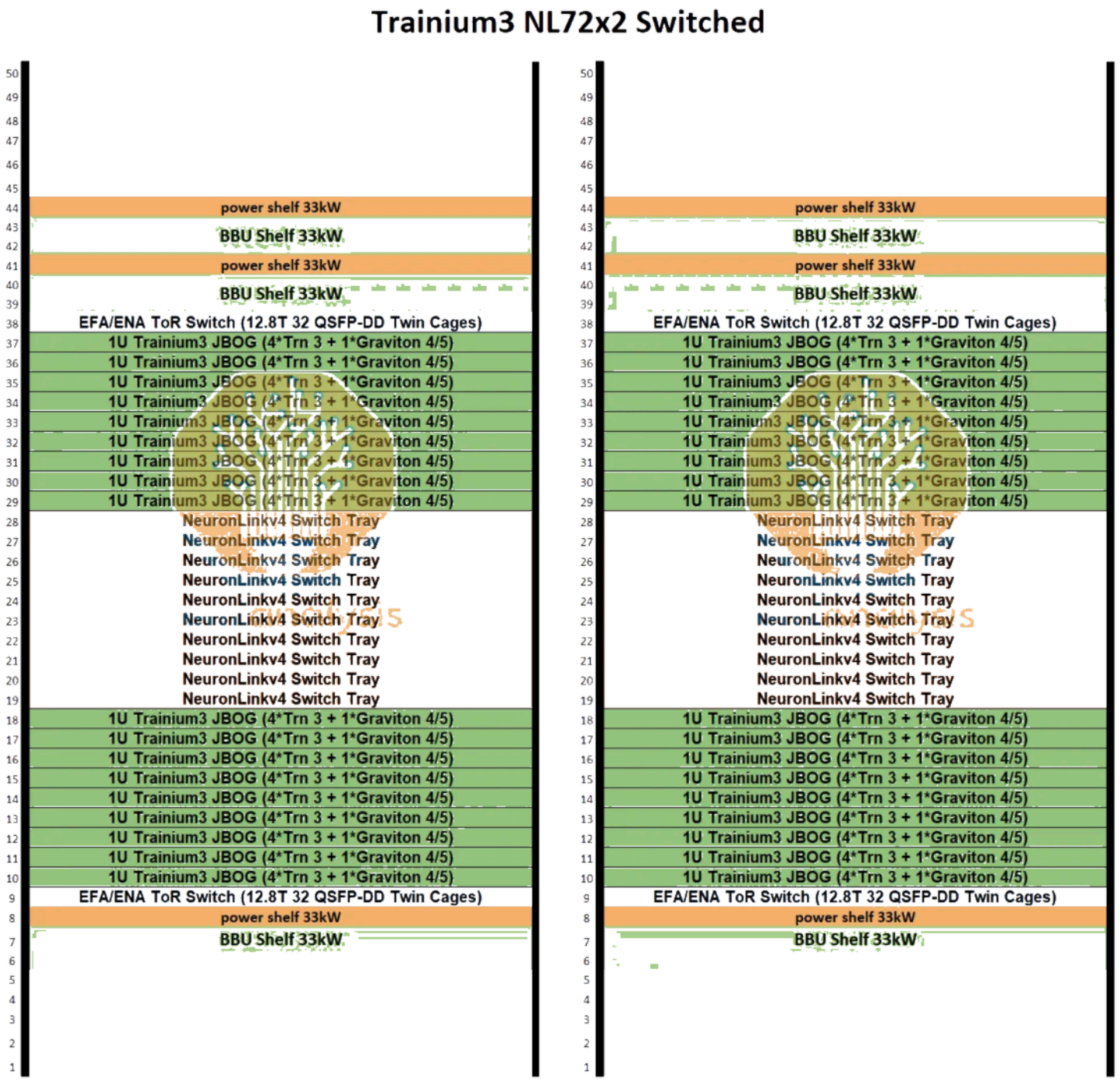
In an upcoming article, we will go deep on writing speed-of-light NKI kernels and microbenchmarking on Trainium3. For our deep dive into the Trainium hardware, check out our previous article where we do a complete hardware breakdown.



DYLAN PATEL, DANIEL NISHBALL, AND 8 OTHERS · 2025年12月5日

Read full story →





Source: SemiAnalysis

We Are Hiring

SemiAnalysis is hiring for various Member of Technical Staff roles. If you find this type of analysis interesting, apply below:

What we're looking for:

> **Highly motivated & skilled Member of Technical Staff to join our growing engineering team.**
>
> **Member of Technical Staff will play a crucial role in developing inference benchmarks & system modeling**

You will also play a role in writing and preparing our newsletter articles for publishing, with the opportunity for direct authorship recognition.

We are hiring individual contributors at all experience levels. Compensation will be competitive

As part of the interview process, you'll complete a paid coding challenge designed to reflect typical daily tasks at SemiAnalysis

Competitive Compensation depending on experience, skillset, location & business needs

Qualifications

Proactive, self-motivated, and capable of working independently in a global team.

Demonstrated experience in ML frameworks such as PyTorch or JAX through professional experience, personal projects, or personal Substack blogs

Solid understanding of at least 1 of the following: transformer architecture, LLM parallelism strategies, and/or CUDA parallel programming

Strong research skills and the ability to synthesize information from various sources to draw insights.

English language proficiency required, Chinese language proficiency preferred

Moving Towards a First-Class PyTorch Experience

Amazon has made a major course correction to its software strategy, and we believe it will meaningfully broaden Trainium3 adoption. TorchNeuron represents a native PyTorch experience instead of duct-taping Trainium onto PyTorch through the PyTorch/XLA project. The new backend is aimed at everyone who isn't a frontier-lab kernel engineer. AWS has even hired the lead of Google's native Torch on TPU effort to work on native Torch for Trainium. He was the TPU SW lead from the start of the program at Google, including XLA, xprof, Cloud TPU, etc.

The old approach made Trainium feel foreign to anyone coming from CUDA. PyTorch/XLA relied on lazy tensor graph capture and had no first-class eager mode. It also didn't support PyTorch's native distributed APIs (torch.distributed.*) or its native

parallelism APIs (DTensor, SimpleFSDP, DDP). Users had to rely on out-of-tree XLA SPMD APIs such as torch_xla.experimental.spmd_fsdp and torch_xla.distributed.spmd. For researchers who are used to native PyTorch on CUDA, switching to Trainium meant relearning how to write a training loop.

Example usage:

```
import torch
import torch_xla.core.xla_model as xm
import torch_xla.distributed.spmd as xs
from torch_xla.experimental.spmd_fully_sharded_data_parallel import
SpmdFullyShardedDataParallel as FSDPv2

# Define the mesh following common SPMD practice
num_devices = xr.global_runtime_device_count()
mesh_shape = (num_devices, 1)
device_ids = np.array(range(num_devices))
# To be noted, the mesh must have an axis named 'fsdp', which the weights and
activations will be sharded on.
mesh = xs.Mesh(device_ids, mesh_shape, ('fsdp', 'model'))

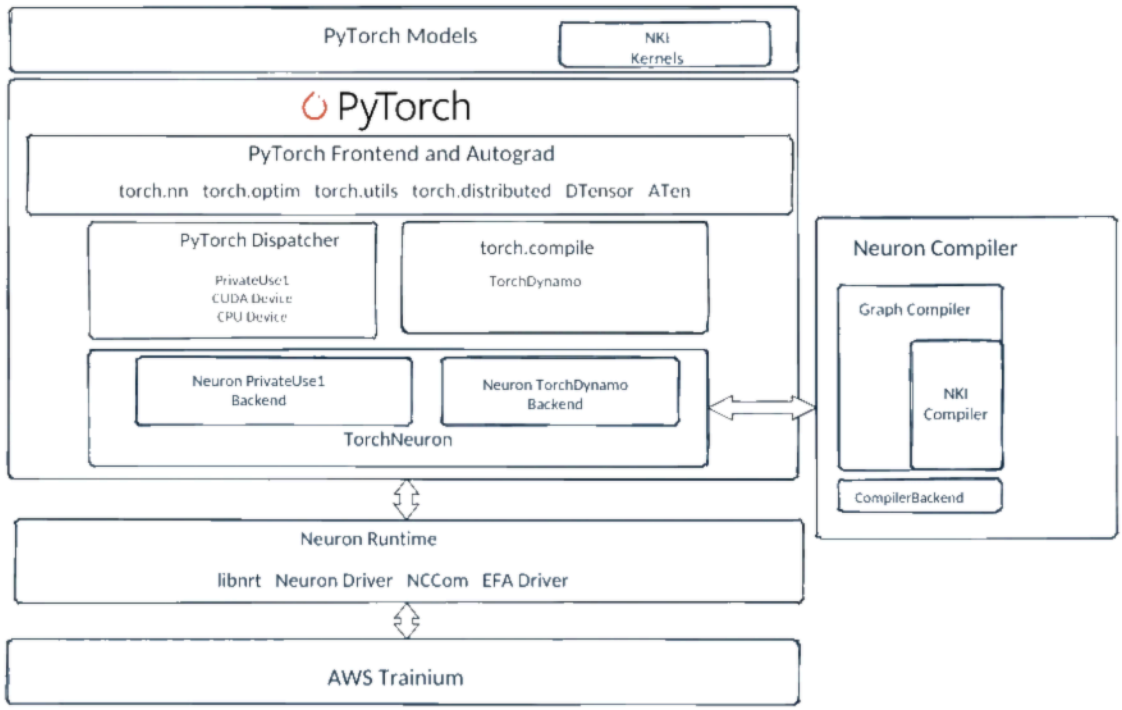
# Shard the input, and assume x is a 2D tensor.
x = xs.mark_sharding(x, mesh, ('fsdp', None))

# As normal FSDP, but an extra mesh is needed.
model = FSDPv2(my_module, mesh)
optim = torch.optim.Adam(model.parameters(), lr=0.0001)
output = model(x, y)
loss = output.sum()
loss.backward()
optim.step()
```

Source: AWS

The pain of converting GPU research codebases by adding XLA markers everywhere was a massive turnoff to anyone who used PyTorch/XLA before.

TorchNeuron Stack - 1



Source: AWS

At a high level, the only change needed is replacing .to("cuda") with .to("neuron"), and in theory the vast majority of the PyTorch ATen operators will work decently well out of the box.

```
model = MyModel().to('cuda')
optimizer = torch.optim.AdamW(model.parameters())

for batch in dataloader:
    optimizer.zero_grad()
    output = model(batch)
    loss = criterion(output, targets)
    loss.backward()
    optimizer.step()

# Update device to neuron to deploy on Trainium

model = MyModel().to('neuron') # <-- this change
optimizer = torch.optim.AdamW(model.parameters())
# same code as other accelerators
for batch in dataloader:
    optimizer.zero_grad()
    output = model(batch)
    loss = criterion(output, targets)
    loss.backward()
    optimizer.step()
```

Source: AWS

In theory, with PyTorch's torch.accelerator hardware-agnostic API, lines of code like bf16 autocast won't even need to be changed, though some APIs in PyTorch and some

research codebases still haven't migrated to the torch.accelerator namespace and still directly call the torch.cuda.* namespace.

As most academics and researchers across robotics, biotech, world models, and self-driving mainly use BF16, Trainium Torch will not support TorchAO MXFP8 on day 0 until they have made the BF16 training experience great. We don't disagree with this sequence, as most people doing MXFP8 training are big Tier 1 labs that have enough kernel engineers to write custom kernels for the big training run.

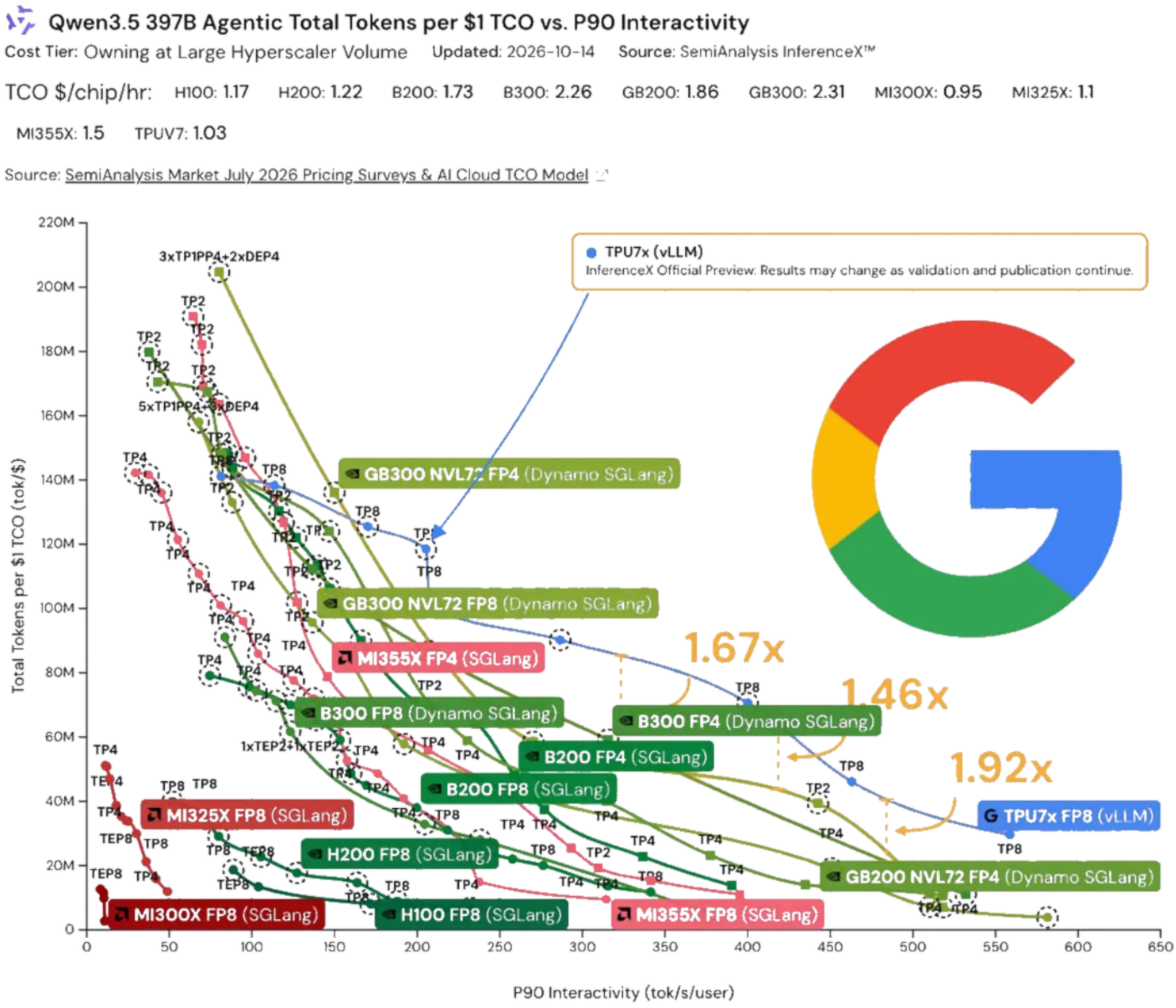
Trainium Torch Ecosystem

PyTorch is popular not only because of core PyTorch but also because of the entire ecosystem of libraries, including Hugging Face Transformers, PEFT, diffusers, TRL, Accelerate, TorchTitan, etc. We are amazed to see that Trainium Torch already supports a ton of these libraries in the ecosystem, including Hugging Face Accelerate, which an insane number of layers of indirection. Trainium supports one of the most popular clean open-source training codebases called TorchTitan with many Tier 2 labs forking it to serve as the foundation of their internal training codebase. Trainium's new PyTorch experience fully supports that too.

We saw how Google was able to successfully partner with Inferact/RadixArk to bring TPUs into the vLLM/SGLang ecosystem, and also land Meta as a production TPU customer. Furthermore, by bringing TPUs into the vLLM/SGLang ecosystem, TPUs were already able to automatically get day 0 support for Muse Spark Open and commitments for day 0 TPU support for popular upcoming models.

While it is true that the increase in agentic coding has made the engine code itself less important, what vLLM/SGLang brings to the table is the ecosystem. No matter the open source model, on day 0, the global ML community trusts vLLM/SGLang as the go-to engine for inference. No vibe-coded engine has that reach.

We are starting to see Trainium rapidly invest in the vLLM/SGLang community with hardware access and early access to Trainium on SGLang built on native PyTorch (TorchNeuron), similar to the TorchTPU approach.



Source: SemiAnalysis

TorchTitan Experience

TorchTitan is one of the most popular open-source pre-training frameworks, and we tested Neuron's support to gauge the user experience of a researcher training a non-trivial model with Trainium. Specifically, we trained a Qwen3-32B on 16 Trainium2 chips, 64 LNC=2 logical cores in total, and we also tried doing it on a rack-scale Trainium3 UltraServer with 64 physical chips.

Overall, we liked the user experience, though there were some rough edges that the Amazon team already has on its roadmap to fix.

While we will provide the Trainium2 MFU below for completeness, the part to focus on is that for ML research ablations, what matters most isn't a speed-of-light-

optimized stack but decent performance and broad support across model architecture modifications, so researchers can iterate without fighting the stack.

With Neuron's official TorchTitan image, we were able to launch the run without any issues. We measured **11,099 tokens per second in average training throughput on eager mode, at 23.89% MFU**. For compiled mode, we saw **14,156 tokens per second and 30.47% MFU, a 1.28x throughput improvement over eager mode.** We tried changing the model architecture slightly in the modeling code and it worked great also.

Neuron also offers training recipes for more modern model architectures, so we tested its gpt-oss 20B training recipe. Neuron supports gpt-oss training through integration with GraphTrainer, TorchTitan's experimental trainer module. Specifically, Neuron adds Trainium-compatible configurations of SimpleFSDP, a compilation backend for PyTorch computation graph capture, and custom NKI kernels for attention and MoE operations. In terms of user experience, the recipe worked out of the box, and everything went well.

One thing that was not yet implemented using the upstream APIs was TokenSwitch. In the age of MoE, for inference and training, MoE combine and MoE dispatch are among the most important comms ops and take anywhere from 60-70% of the communication volume, and TokenSwitch is PyTorch's first-class API for them. Trainium doesn't support it yet and instead relies on a custom TorchTitan class called NeuronEPTokenDispatcher. Moving it onto the TokenSwitch API looks like a simple fix, and the AWS team has committed to collaborating with the Meta PyTorch team to modify the TokenSwitch ABC such that it can support Trainium too.

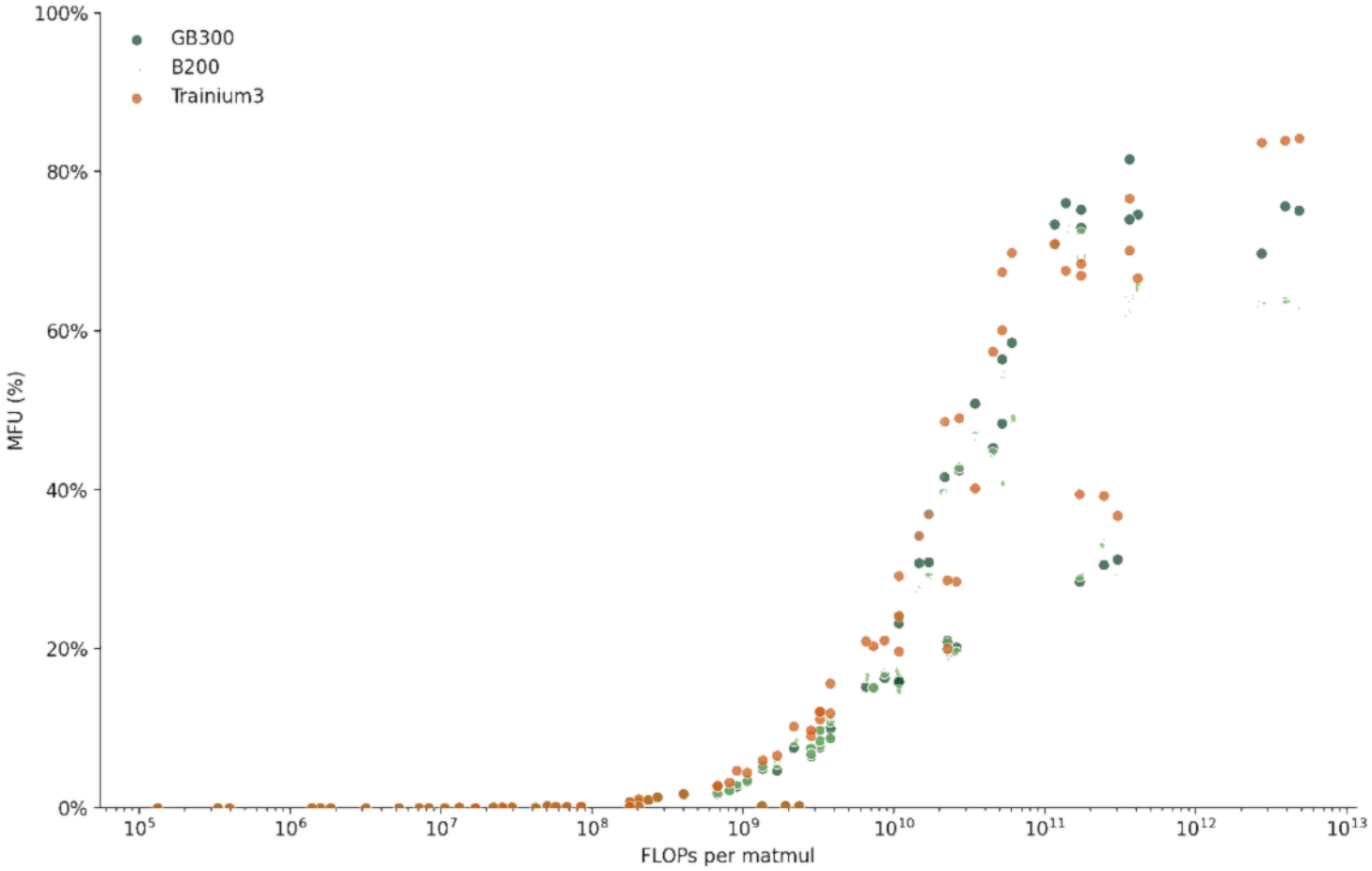
The part where it was slightly disappointing was the Neuron compiler took a long time to compile, leading to a long wait in the first step. Concretely, the first step took 184.6 seconds for eager mode (255.7 seconds for compiled mode), which is 30-55x longer than the average step time, excluding the first step. For gpt-oss, this training run came at a ridiculously high compilation time of 27 minutes and 17 seconds, which leads to a terrible user experience, especially for researchers who experiment with new model architectures. The reason for this is that historically the Trainium compiler stack has mainly been focused on inference, where compile time doesn't matter because compilation is done offline and cached once per image and doesn't affect production inference, whereas in training it does. While this is the right call for

inference at the time, for researchers iterating on model architecture, that's a meaningful friction point.

The Neuron team is already addressing this directly. Upcoming TorchNeuron releases introduce a new compiler backend with a broad set of precompiled NKI kernels and a compile stack re-architected for fast compilation, targeting up to a 2 orders of magnitude reduction in time-to-first-step for both eager and compiled modes, with minimal or no performance regression. If they deliver on that, compilation time stops being a conversation.

Training MatMul Performance

One of the most important things for experimental training is to have decent performance across a broad range of shapes as ML researchers try lots of different things. We benchmarked a single Trainium3 logical core (LNC=2) on the PyTorch nn.functional.linear API to understand its matrix multiplication performance, and we compared it with GB300 and B200 for reference. We swept through the matrix shapes of projection operations in major open weight models, and we set the M dimension to 1, 128, and 2048 to represent different batch size scenarios.



Source: SemiAnalysis

In our benchmark, Trainium3 achieved a visibly higher MFU than GB300, starting from the mid-range, where the matrix multiplication performs 1 GFLOP of computation. It was only after 100 GFLOP that GB300 and B200 started outperforming Trainium3, and Trainium3 won again after 1 TFLOP.

FlexAttention API

FlexAttention is a high-level API that allows implementing different attention variants using PyTorch, while offering backward pass auto-generation and reasonable performance. Good FlexAttention support shows a hardware's flexibility and ease-of-use to researchers. We ran the [FlexAttention benchmark suite] on Trainium3 to see its FlexAttention support.

semianalysis

Configuration	Configuration Space
Batch Size	1, 4
QKV Sequence Length (BF16 Prefill)	512, 2048, 8192
Attention Heads (Q / KV)	8 / 8, 8 / 2
Attention Head Dimension	64, 128
Attention Mask Pattern	Dense, causal, ALiBi, sliding window

Source: SemiAnalysis

Overall, Trainium3 passed 89 out of 96 test cases, a 93% pass rate. All 7 failed cases were at large batch size (4) and sequence length (8192), and **all failed due to compilation exceeding the time budget rather than numerical errors**. These issues will be fixed in later versions of TorchNeuron.

Additionally, we tried implementing custom attention variants not in the FlexAttention benchmark suite for Trainium3, and it matched the numerics of Nvidia GPUs.

Pattern	FWD + BWD Pass	FWD Fail	BWD Fail	Total Cases
Dense	21	0	3	24
Causal	24	0	0	24
ALiBi	24	0	0	24
Sliding Window	20	0	4	24
Total	**89**	**0**	**7**	**96**

Source: SemiAnalysis

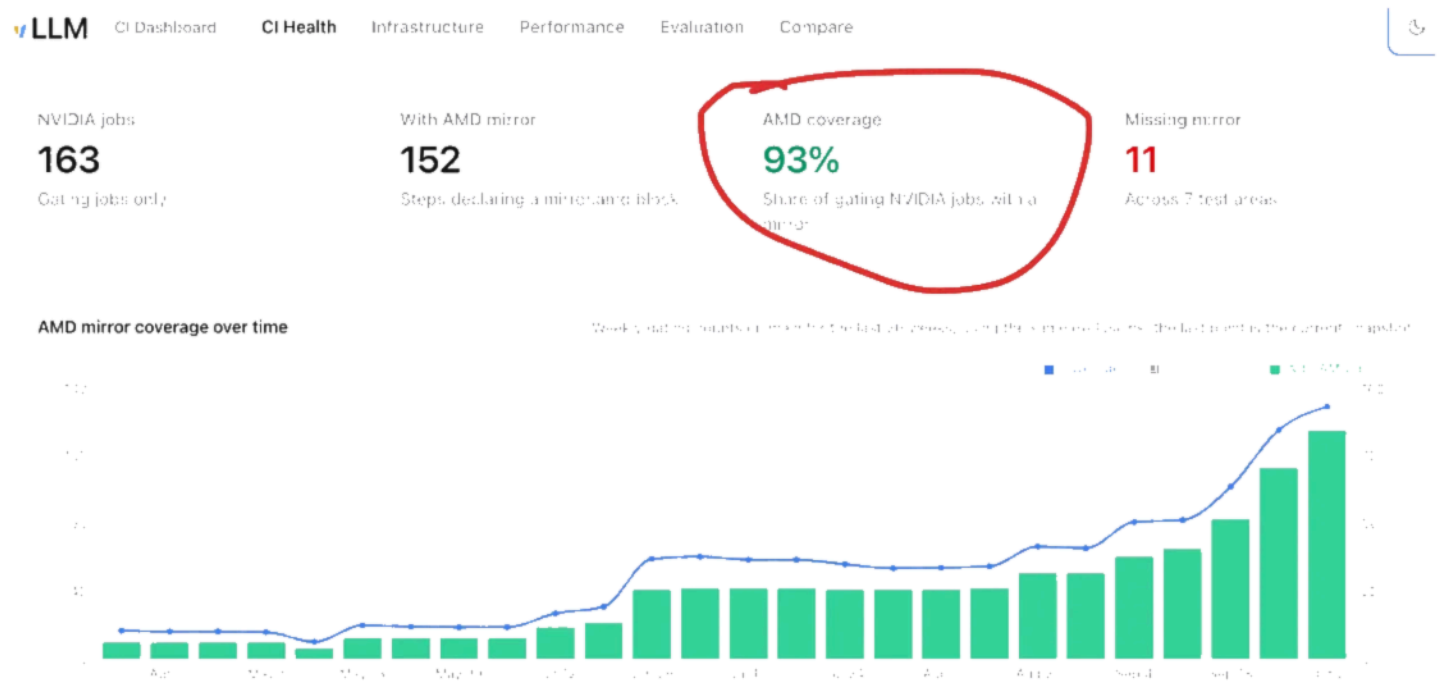
Automated Quality Assurance

SemiAnalysis is known in the global ML community to be the "CI Karen," setting a high bar across all hardware vendors and doing whatever it takes to get senior leadership of AI chip companies to actually provide their engineers with enough resources and CI fleet to maintain a high bar of automated quality control.



Source: SemiAnalysis

One recent achievement, which took over 9 months of fighting with AMD senior leadership, was getting their engineers enough stable nodes to achieve 93% parity on gating test groups.



Source: vLLM Dashboard

Inference is a narrow problem. A lab serving a handful of production models needs a forward pass, mostly fixed architectures, and a small set of hot kernels.

Training is a generality problem. Each of the thousands of operators needs a backward pass, and research codebases sometimes accidentally run operators in higher precision, which must also be supported. To ensure that Trainium works across a wide range of model architectures and can do eager mode and compile mode well, AWS collected a giant list of top 100k+ Hugging Face models that spread across a whole range of different normalization, MLP, MoE layouts and attention types. Not only do they test that it doesn't crash, but they also test performance, which ops use CPU fallbacks and don't have native support yet, where graph breaks are and most importantly, they test numerics against CPU and Nvidia baselines. They test tensor level parity and loss curve divergence also. It is easy to finish an exam within 10 minutes, but it doesn't matter how fast you finish an exam if you get a 0% on it. Same thing with training stacks. It is easy to get decent MFU%, but it is hard to get decent MFU% while not introducing numeric issues.

Task	Classes	Models	% of All	Eager %	Mean OpCov %	Compile %
text-generation	460	52,589	50.1	87.6	98.3	80.9
image-classification	62	19,367	18.5	48.4	98.6	41.9
automatic-speech-recognition	18	11,567	11.0	83.3	93.9	33.3
audio-classification	5	3,406	3.2	40.0	95.3	20.0
object-detection	8	3,105	3.0	75.0	96.8	25.0
video-classification	7	2,166	2.1	85.7	99.4	85.7
text-to-speech	3	2,136	2.0	66.7	96.2	66.7
robotics	5	1,955	1.9	100.0	98.5	100.0
zero-shot-image-classification	17	1,697	1.6	76.5	98.1	76.5
text-classification	10	1,608	1.5	70.0	98.0	50.0
document-question-answering	6	1,033	1.0	100.0	98.4	100.0
token-classification	6	997	1.0	100.0	98.4	100.0
image-text-to-text	31	507	0.5	58.1	98.4	41.9
image-segmentation	10	465	0.4	60.0	96.4	50.0
feature-extraction	31	400	0.4	74.2	97.6	54.8
image-feature-extraction	30	362	0.3	56.7	98.6	43.3
fill-mask	10	309	0.3	90.0	98.2	100.0
mask-generation	6	188	0.2	50.0	98.7	33.3
depth-estimation	6	119	0.1	33.3	98.5	33.3
zero-shot-object-detection	3	72	0.1	100.0	94.9	66.7
text-to-video	3	27	0.0	33.3	98.7	33.3
sentence-similarity	4	19	0.0	75.0	99.1	50.0
Other	84	715	0.7	78.6	97.7	69.0
Total	841	104,909	100	78.7	98.1	69.6
Total (Equal Class Weight)	—	—	—	71.1	—	58.5

Source: AWS

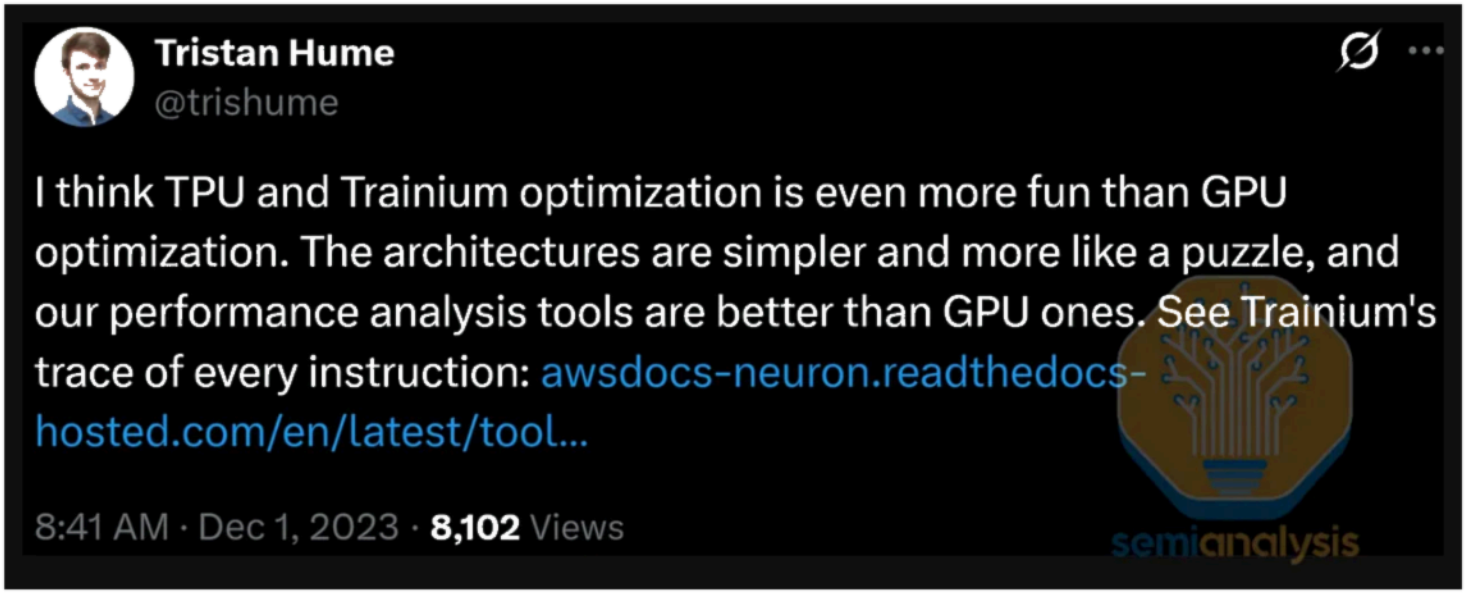
Although the pass rates aren't at 100% yet, what matters is the trajectory. AWS is climbing rapidly, with agent swarms systematically fixing the remaining failures. We think it's a strength that AWS's internal summary table, the one senior leadership sees, shows gaps transparently rather than hiding them.

We say this from experience. When SemiAnalysis and upstream maintainers have tried to fix quality control culture at other hardware vendors, we've found that senior leadership is genuinely shocked when shown the real parity pass rates. At one vendor in particular, there was so much middle management filtering that by the time CI numbers reached senior leadership, they showed 99.99%, while the actual gating parity pass rate at the time was below 5%.

Neuron Explorer - Better than Nvidia and AMD Profilers

Trainium's profiler is by far leading the industry and is SOTA status. Here we share our initial impressions and plan on sharing our full user experience in a future Trainium kernel article.

Trainium's profiler is loved by all the performance engineers at Anthropic. Anthropic's head of performance (who is ex-Jane and Finch Street) has publicly said that performance analysis on Trainium is way better than performance analysis on Nvidia GPUs (and obviously better than on AMD GPUs).



Source: Tristan Hume on Twitter

We agree with Tristan Hume. In our experience, the profiling experience was amazing, after getting over the initial hump of the profiling setup being slightly confusing (but not impossible). There are multiple environment variables to set, different file types to save, and various commands to capture and view the profile traces. This is only the kernel profiling setup, and the PyTorch profiling setup is similar but with slight differences, making things even more confusing. With the help of Codex (since Claude refuses to write Trainium kernels), our best effort working setup is as follows:

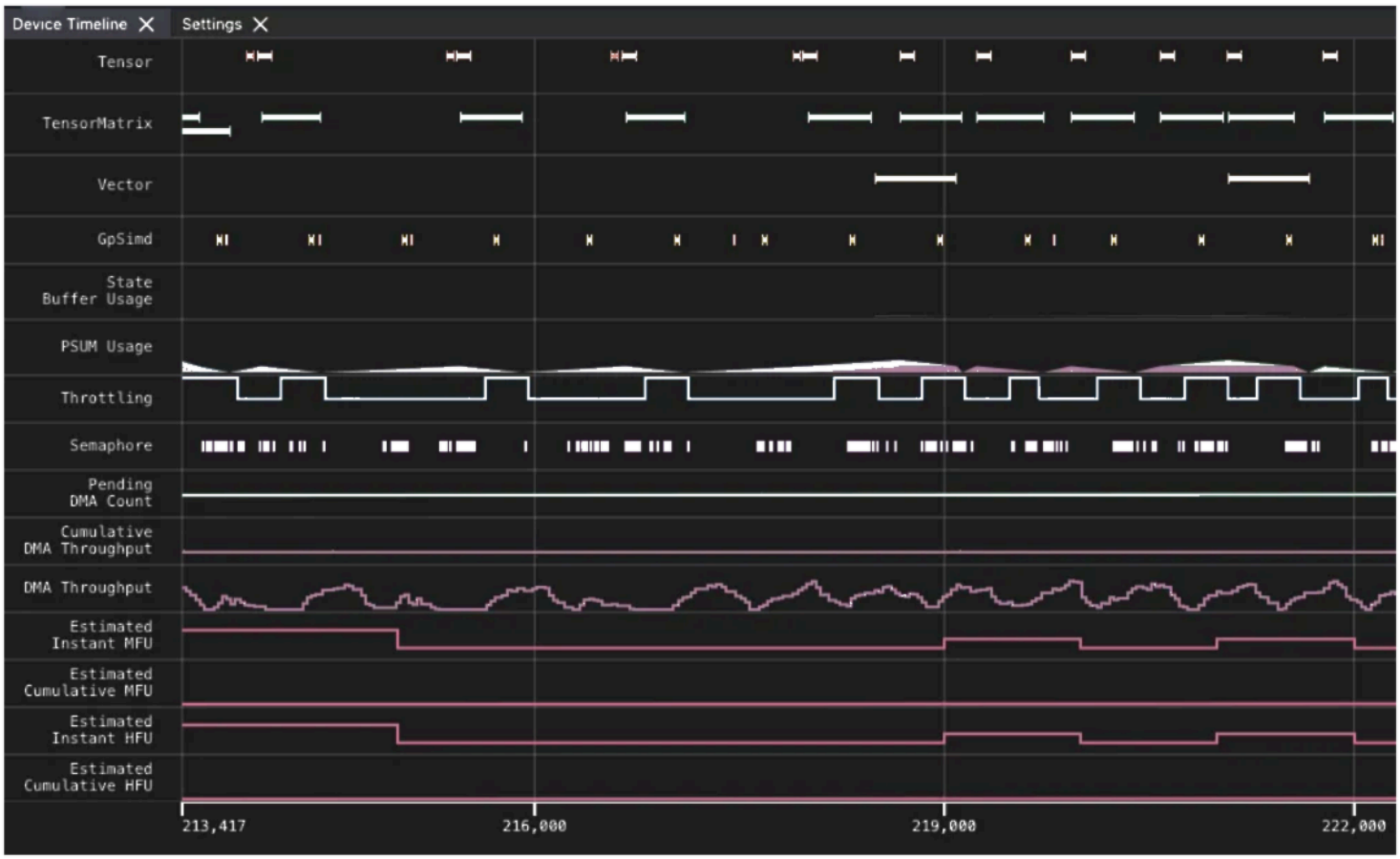
Launch a persistent Neuron Explorer server

Profile the kernel with neuron-explorer capture command

Upload the profile trace as an artifact to the Neuron Explorer server as a web request

We shared feedback with AWS that we wished the documentation could be clearer on the workflow, explaining what environment variables need to be set for what reasons, and what purpose each file type serves.

Once the setup is done, the profiler user experience is great. Neuron Explorer is co-designed with the hardware. We particularly like the timeline separating hardware units in different rows, so that bubbles are visible at a glance. In contrast, GPU programmers have to roll their own setup for intra-SM profiling as the official Nsight tools don't support that.



Source: SemiAnalysis

We also like how the viewer visualizes dependencies. When a block is selected, its dependencies and dependents light up in different colors, which helps us reason about work overlapping, e.g. whether computation sufficiently overlaps with memory loading. The image below shows that the selected white block is:

Flow-dependent on the red block

Order-dependent on the orange block

A flow dependency of the blue block

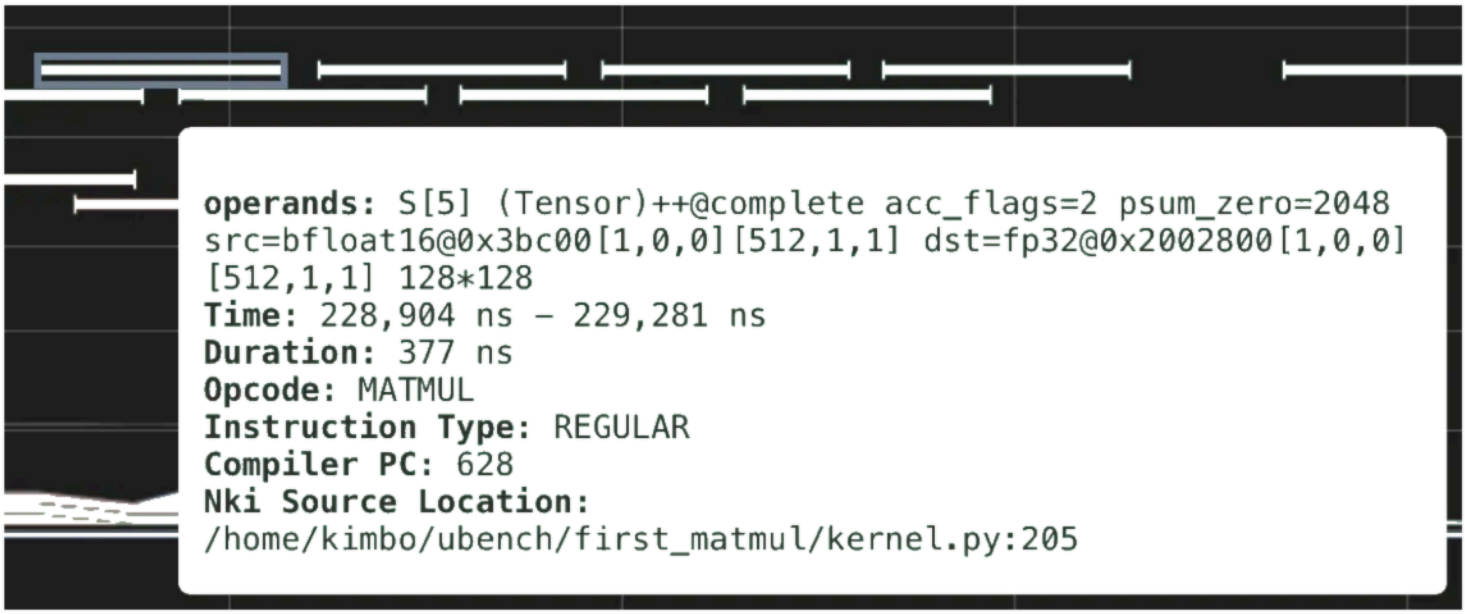
An output dependency of the purple block



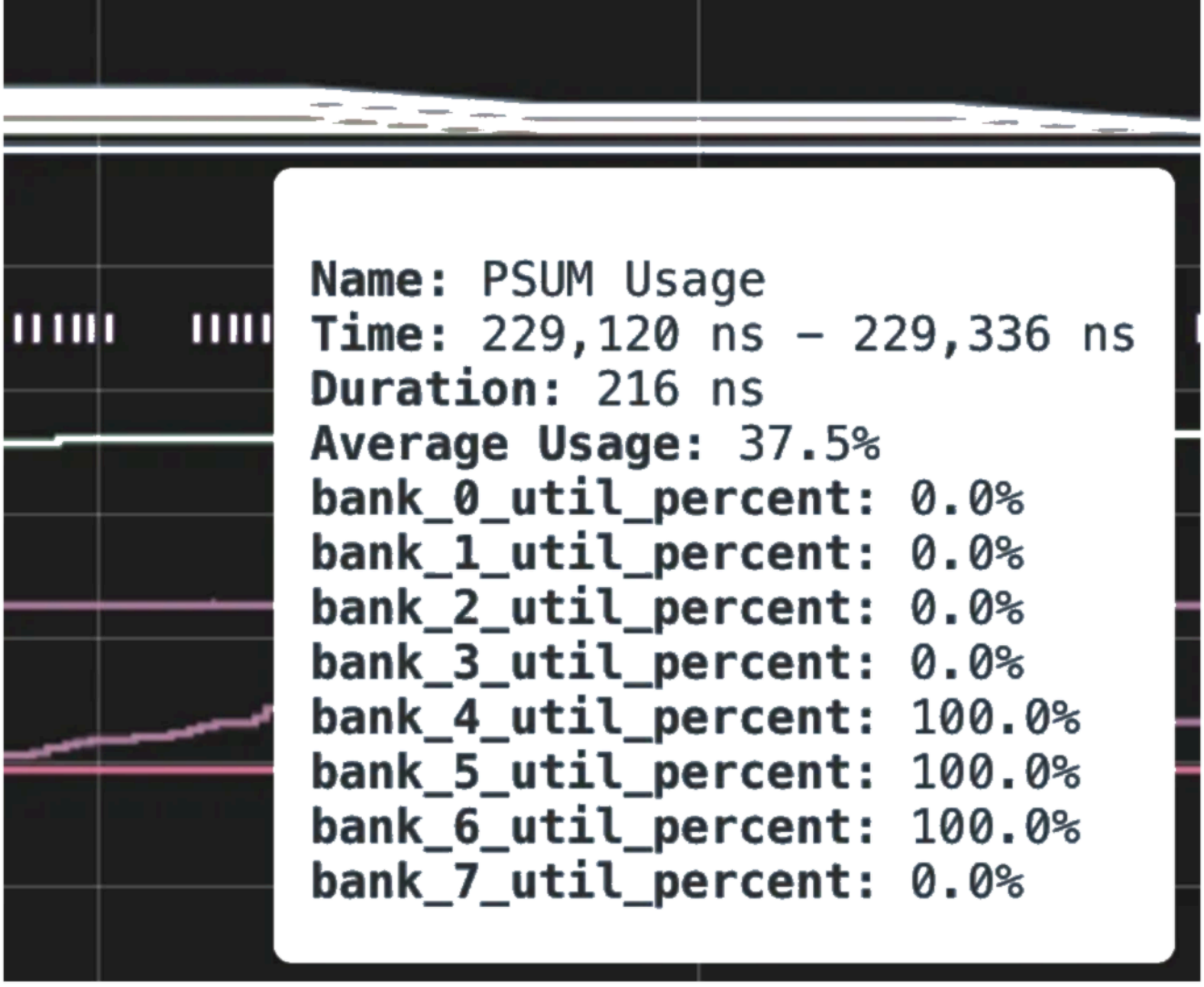
Neuron Explorer dependency visualization. Source: SemiAnalysis



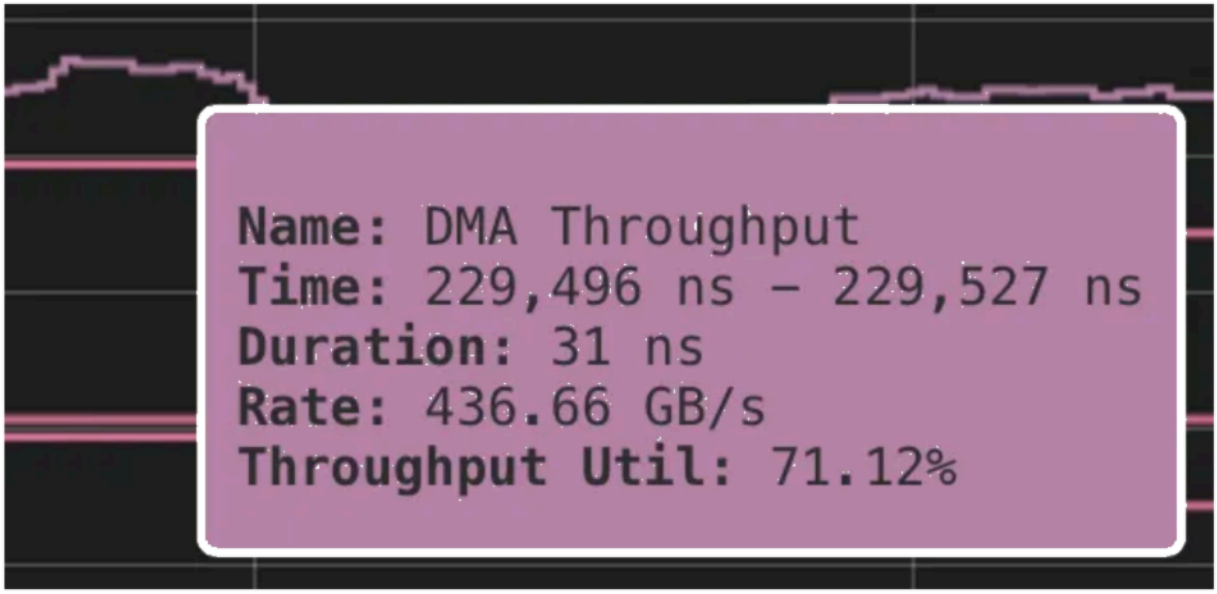
Neuron Explorer code editor mapping. Source: SemiAnalysis



Neuron Explorer: Event information when hovering on an event. Source: SemiAnalysis



Neuron Explorer memory bank utilization visualization. Source: SemiAnalysis



Neuron Explorer microsecond-level memory throughput visualization. Source: SemiAnalysis

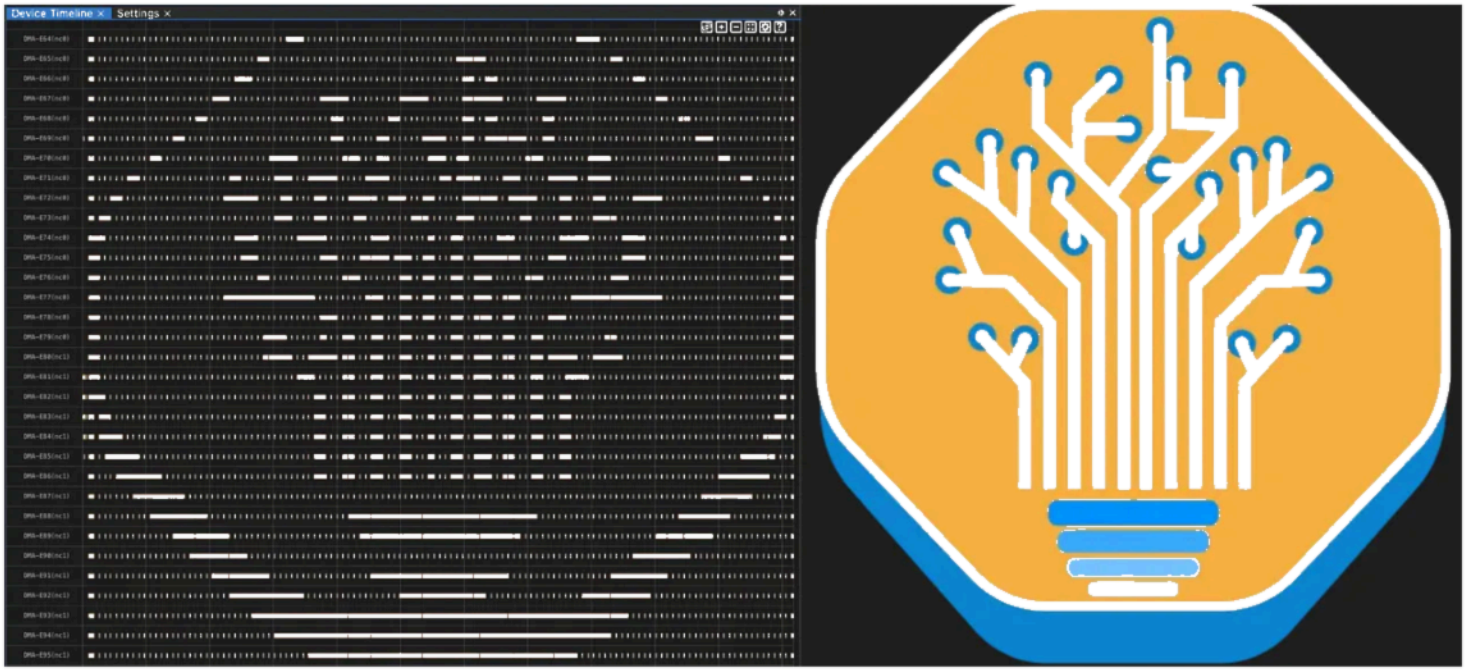
Trainium Profiler Demo - Pixel Art

Neuron Explorer has many more features, including code editor mapping, MFU estimation, memory usage, and even creating pixel art due to how precise the profiler is. The images below are all from profiler traces of running on actual Trainium

hardware, not fake profile reports. We were mainly inspired by the Bad Apple post from Daniel Liu (Anthropic Performance Engineer) and have created our own agentic coding SKILL.md for creating images. We did not have the time yet to create one to automatically generate videos frame by frame but will in the future and open source this SKILL.md for everyone to use. We mainly generate this pixel art by using the 16 DMA engines per physical core by "coloring in" regions by turning the DMA engines active, and the width of the shaded area is determined by how much data we have each DMA engine move.



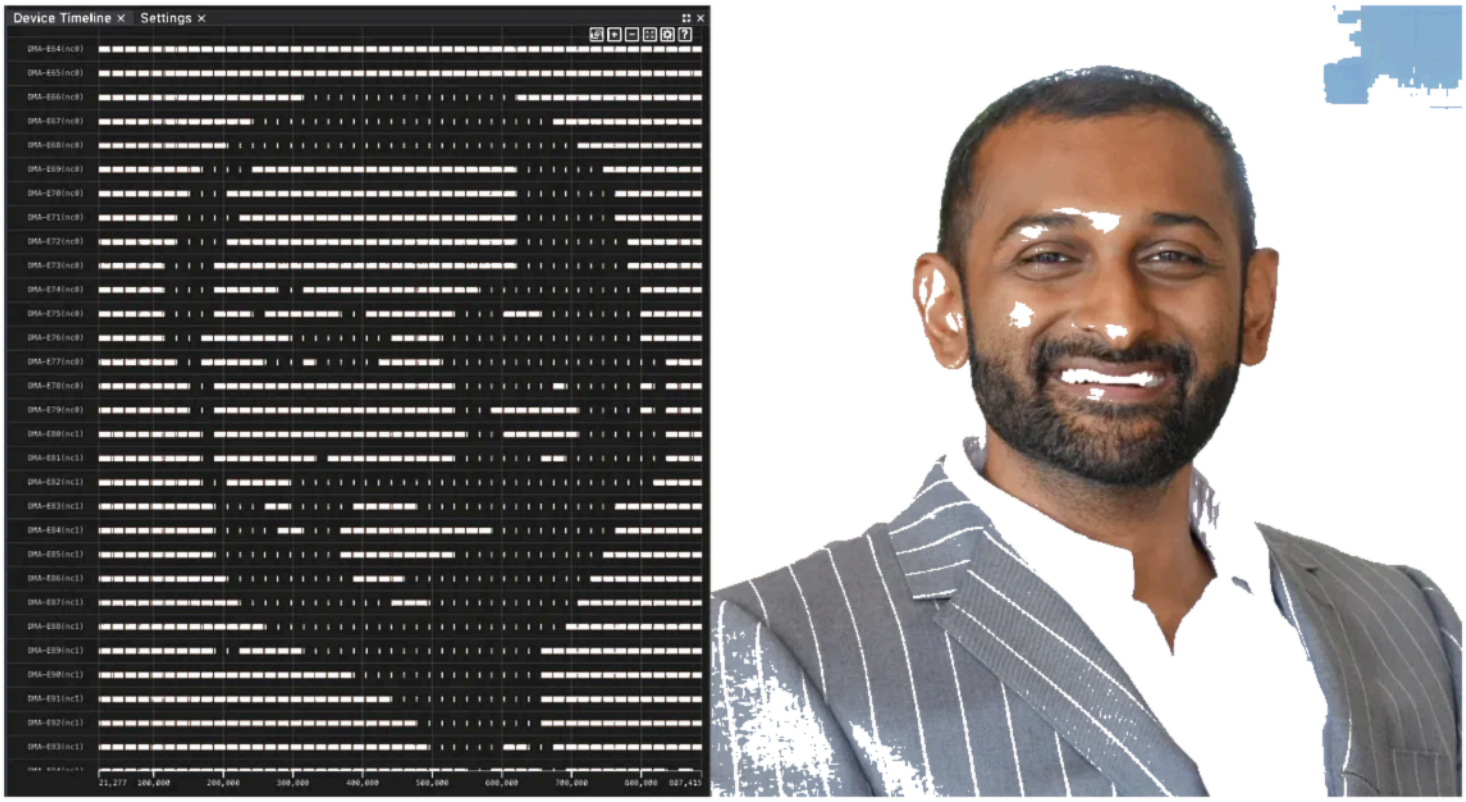
Bad Apple on Neuron Explorer. Source:



Visualizing the SemiAnalysis logo. Source: SemiAnalysis

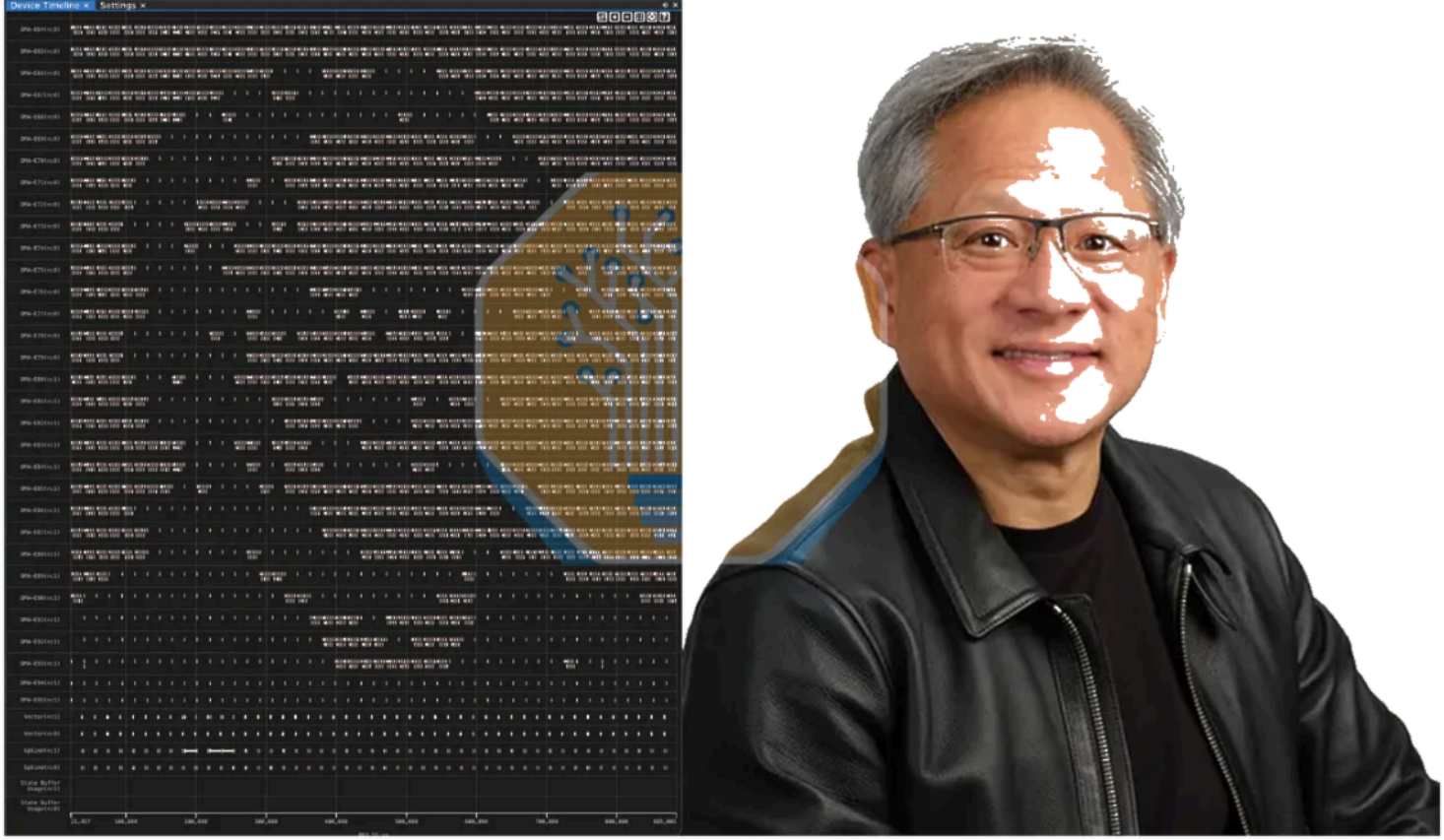


Visualizing Jeff Bezos (pre looksmaxxing). Source: SemiAnalysis



Visualizing Dylan Patel (pre looksmaxxing). Source: SemiAnalysis

Since AWS is a partner of Nvidia, Jensen should tell his engineers to book a meeting with the AWS Trainium team on the hardware features needed to do accurate fine-grained profiling such that Nvidia can learn how to add this capability to Feynman or Feynman-Next GPUs.

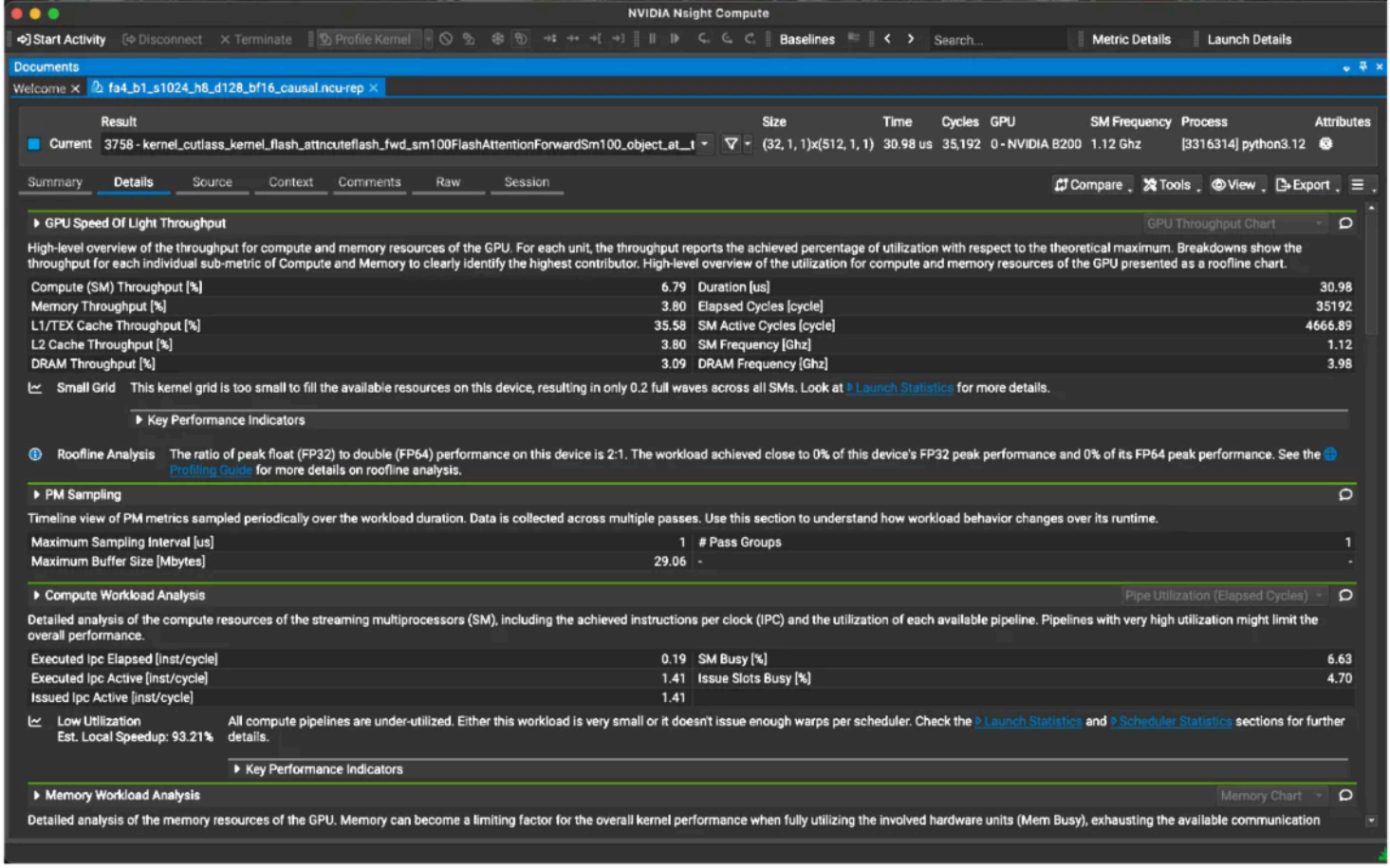


Visualizing Jensen on Trainium3. Source: SemiAnalysis

Nvidia's intra-kernel profiler Nsight Compute employs a different design. It offers aggregate metrics and detailed information on performance properties, rather than an event-based timeline viewer. The closest thing we found was Tri Dao's QuACK intra-kernel profiler, which uses Nvidia's In-Kernel Event Tracing API. While QuACK requires custom design to make Perfetto visualize accordingly, it's great to see that AWS made this a well-integrated, first-class experience in Neuron Explorer.



QuACK's intra-kernel profiler: Visualization of a QuACK GEMM kernel. Source: QuACK



Nsight Compute view of FlashAttention 4 workload analysis. Source: Nvidia Nsight

Software Must Turn Trainium's Hardware Savings into Cheaper Training

The software progress above matters because Trainium3 gives AWS a substantially lower hardware cost base to work with. As we discussed in our Trainium3 deep dive, custom silicon avoids Nvidia's accelerator margins. Our updated AI Cloud TCO Model puts cluster capex at ~$20,400 per chip for air-cooled Trainium3 PDS and ~$25,500 for liquid-cooled Trainium3 MAX, vs ~$53,600 for GB200 NVL62 under the same hyperscaler purchasing profile. This means ~52-62% savings in upfront investment per accelerator.

Including capital and operating expenses, the air-cooled Trainium3 system costs ~$0.80 per chip-hour to own and operate, compared with $1.85 for GB200. The gap narrows once we account for compute, as Trainium3 also delivers less peak throughput per chip. On marketed dense FP8 performance, PDS costs $0.32 per PFLOP-hour versus $0.37 for GB200, reducing its advantage to ~14%. The utilization achieved by the software stack determines how much of that advantage survives.

Even when assuming 35% effective MFU for Trainium3 and 50% for GB200 in FP8 training in our TCO model, that gap is enough to put Trainium3 behind on cost per effective training FLOP despite its cheaper hardware. Holding hourly costs and

GB200's 50% MFU constant, increasing the air-cooled Trainium3 system's effective MFU from 35% to around 43% would bring it to cost parity.

.



Bridging the gap between the world's most important industry, semiconductors, and business.



129 Likes · 1 Restack

Restacks



Write a comment...

© 2026 Dylan Patel · Privacy · Terms · Collection notice
Substack is the home for great culture