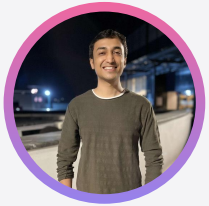
Grafana Labs

Auto-instrumentation for GPU performance using eBPF

SREcon EMEA ‘25

Nikola Grcevski
Principal Software Engineer
Beyla/OpenTelemetry



Annanay Agarwal
Staff Software Engineer
AI/ML

Agenda

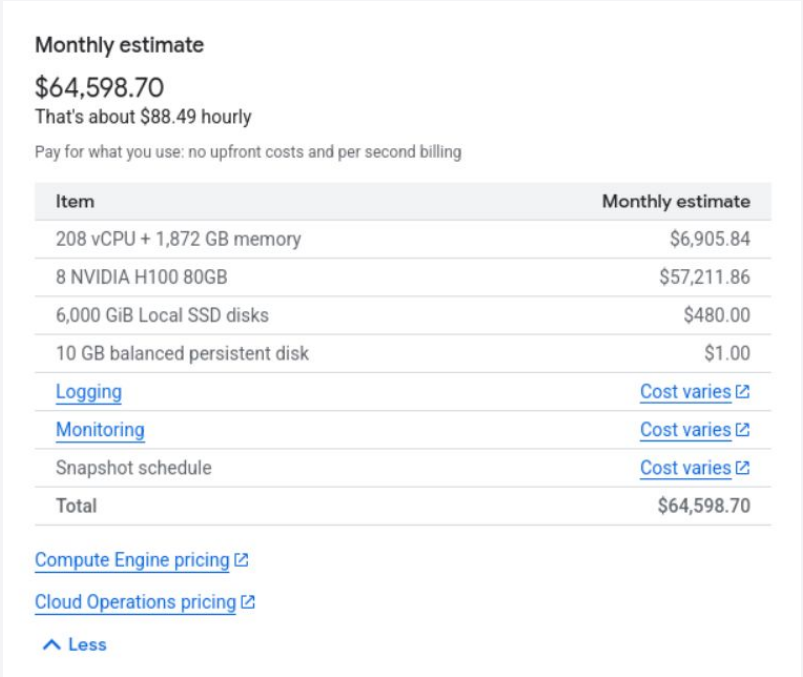
Define the problem space

GPU monitoring basics

eBPF to supercharge GPU observability

Why do we want to monitor GPU workloads?

- LLM/AI workloads rely on GPUs
  - LLM/AI functionality is now part of growing number of software solutions
  - This trend is not going to stop
- Running LLM/AI workloads is expensive
  - We want to ensure our platforms are reliable/stable
  - We want to spend as little as possible



Monthly estimate

$64,598.70

That's about $88.49 hourly

Pay for what you use: no upfront costs and per second billing

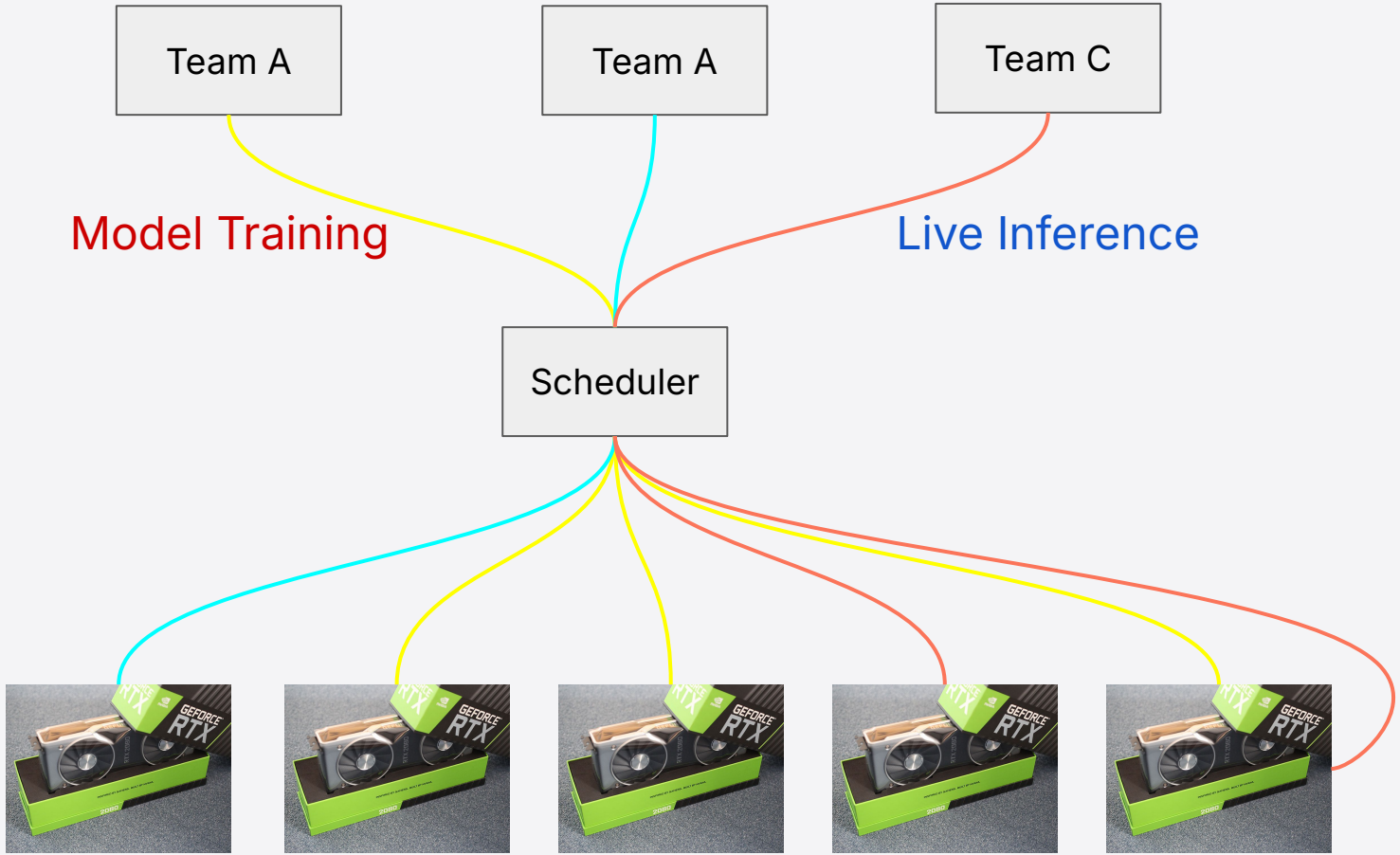
Item	Monthly estimate
208 vCPU + 1,872 GB memory	$6,905.84
8 NVIDIA H100 80GB	$57,211.86
6,000 GiB Local SSD disks	$480.00
10 GB balanced persistent disk	$1.00
Logging	Cost varies
Monitoring	Cost varies
Snapshot schedule	Cost varies
Total	$64,598.70

Compute Engine pricing

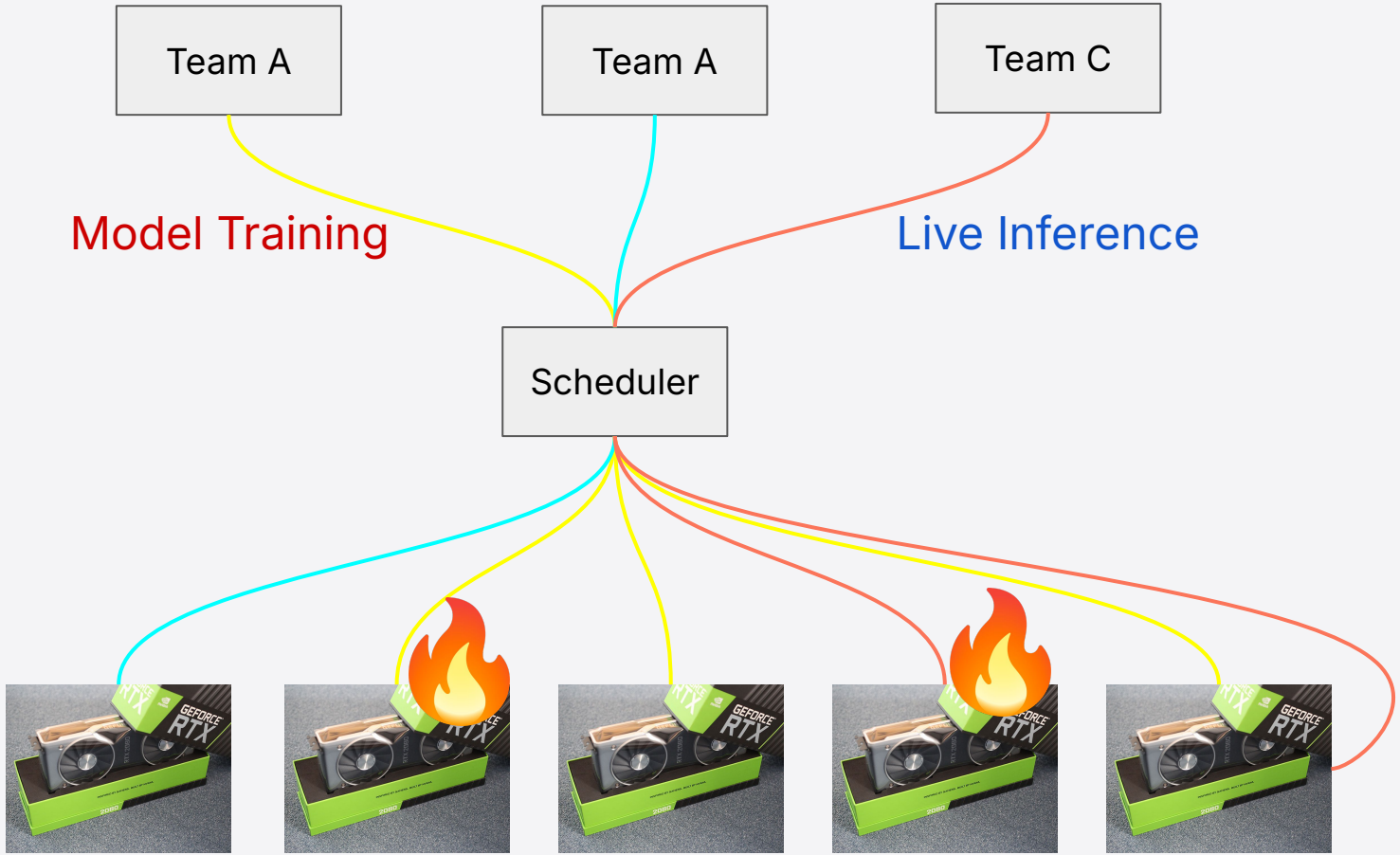
Cloud Operations pricing

Less

Different workloads need GPU processing



We want to make sure we schedule correctly



Interruptions can be costly

Component	Category	Interruption Count	% of Interruptions
Faulty GPU	GPU	148	30.1%
GPU HBM3 Memory	GPU	72	17.2%
Software Bug	Dependency	54	12.9%
Network Switch/Cable	Network	35	8.4%
Host Maintenance	Unplanned Maintenance	32	7.6%
GPU SRAM Memory	GPU	19	4.5%
GPU System Processor	GPU	17	4.1%
NIC	Host	7	1.7%
NCCL Watchdog Timeouts	Unknown	7	1.7%
Silent Data Corruption	GPU	6	1.4%
GPU Thermal Interface + Sensor	GPU	6	1.4%
SSD	Host	3	0.7%
Power Supply	Host	3	0.7%
Server Chassis	Host	2	0.5%
IO Expansion Board	Host	2	0.5%
Dependency	Dependency	2	0.5%
CPU	Host	2	0.5%
System Memory	Host	2	0.5%

Table 5 Root-cause categorization of unexpected interruptions during a 54-day period of Llama 3 405B pre-training. About 78% of unexpected interruptions were attributed to confirmed or suspected hardware issues.

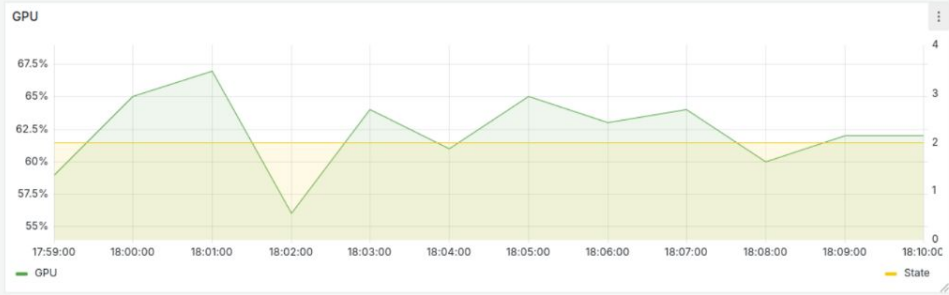
https://ai.meta.com/research/publications/the-llama-3-herd-of-models/

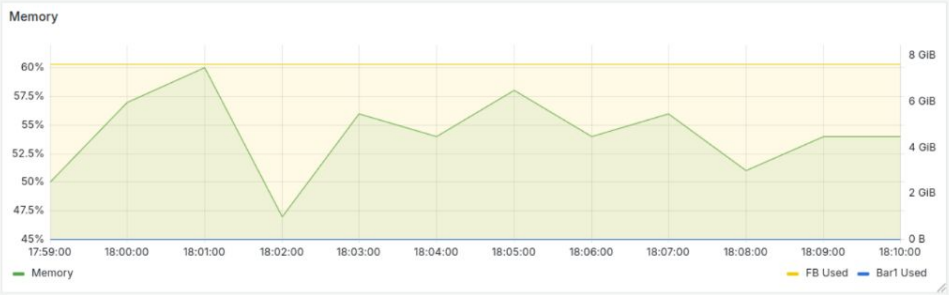
Which is why we need to know when things fail

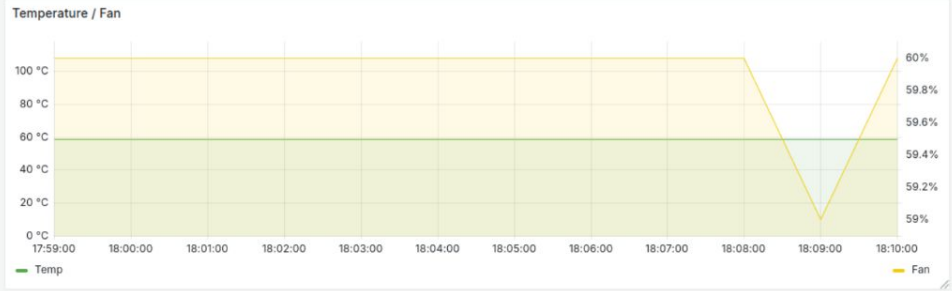
[Nvidia DCGM exporter]

Check	Component Tested	nd96asr_v4 expected	nd96amsr_a100_v4 expected	nd96isr_h' expect
check_gpu_count	GPU count	8	8	8
check_nvlink_status	NVlink	no inactive links	no inactive links	no inactive
check_gpu_xid	GPU XID errors	not present	not present	not present
check_nvsmi_healthmon	Nvidia-smi GPU health check	pass	pass	pass
check_gpu_bandwidth	GPU DtH/HtD bandwidth	23 GB/s	23 GB/s	52 GB/s
check_gpu_ecc	GPU Mem Errors (ECC)	20000000	20000000	20000000

NVIDIA 3090 running Qwen3 PyTorch (GPU counters)







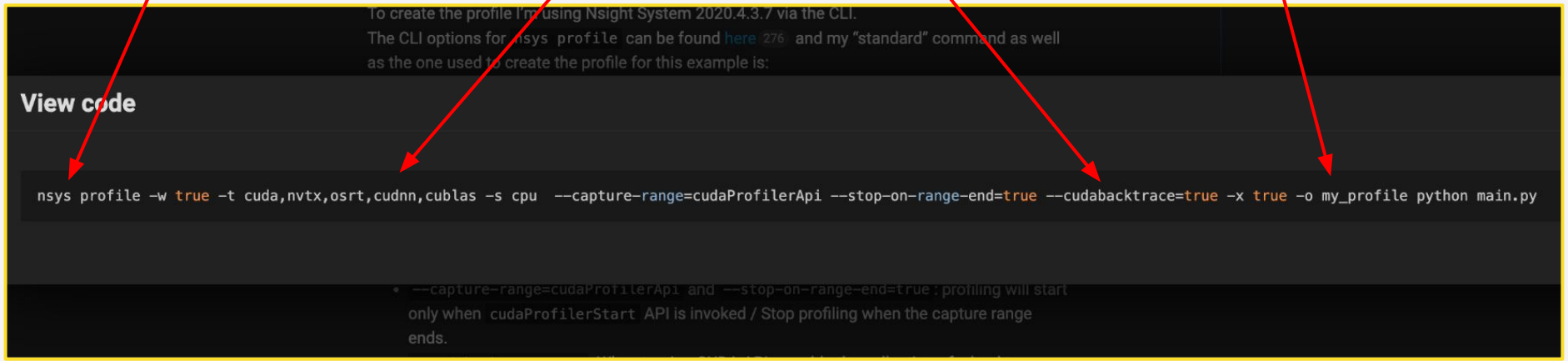
Profiling helps us get better insights

runs in the app process

Libraries to profile

Sample cpu call stack

output on local machine

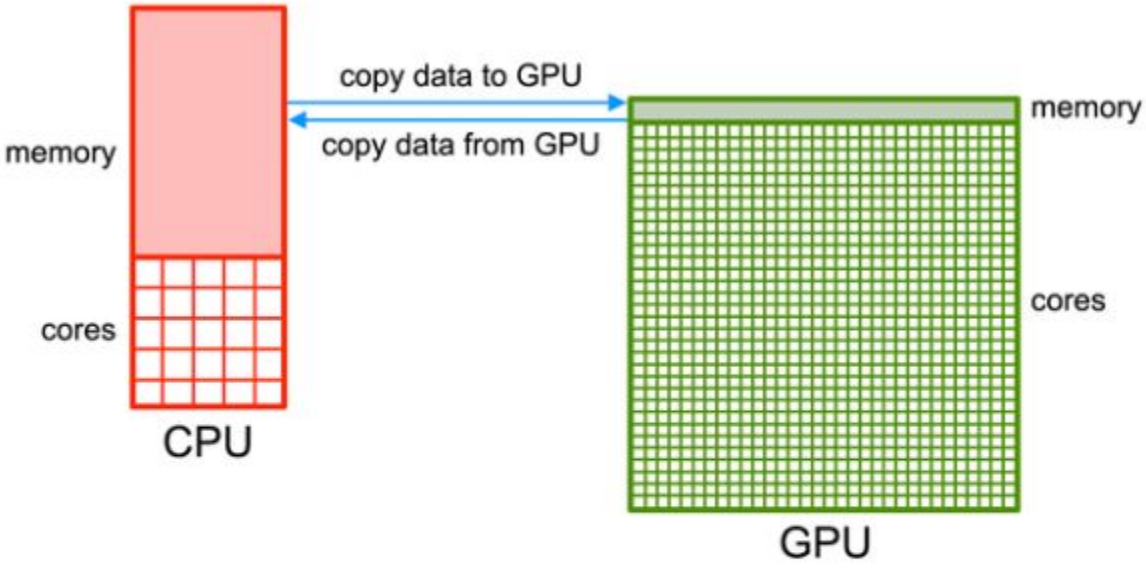


[NVIDIA Nsight systems]

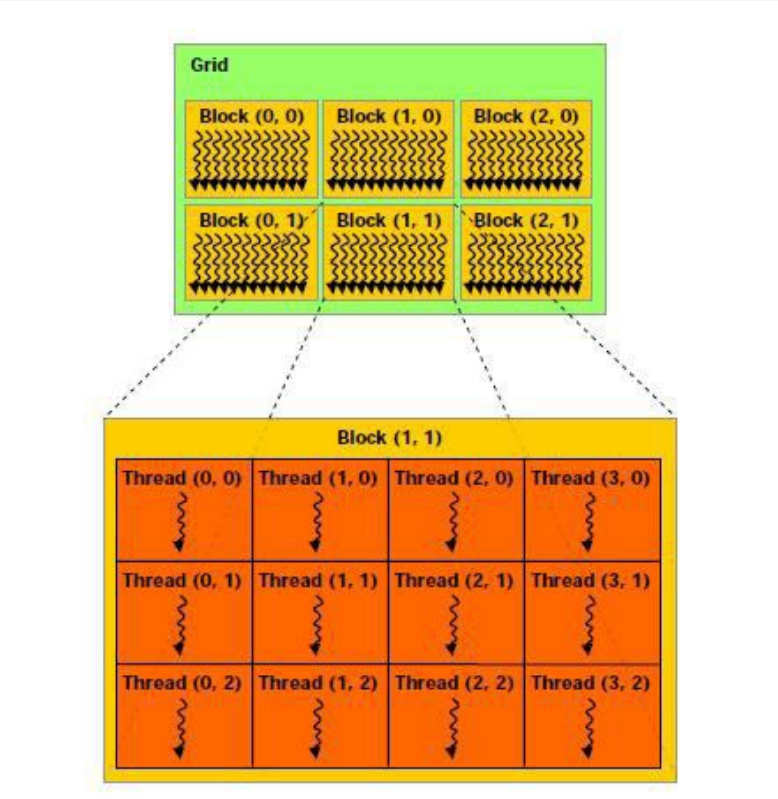
GPU profiling is great, but limited currently

- Performance overhead
- Manual invocation or instrumentation
- Lack of CPU/usage context and correlation with GPU events
- Not as easy to use and scale across your clusters
  (compared to CPU profiling tools)

What’s so special about GPUs



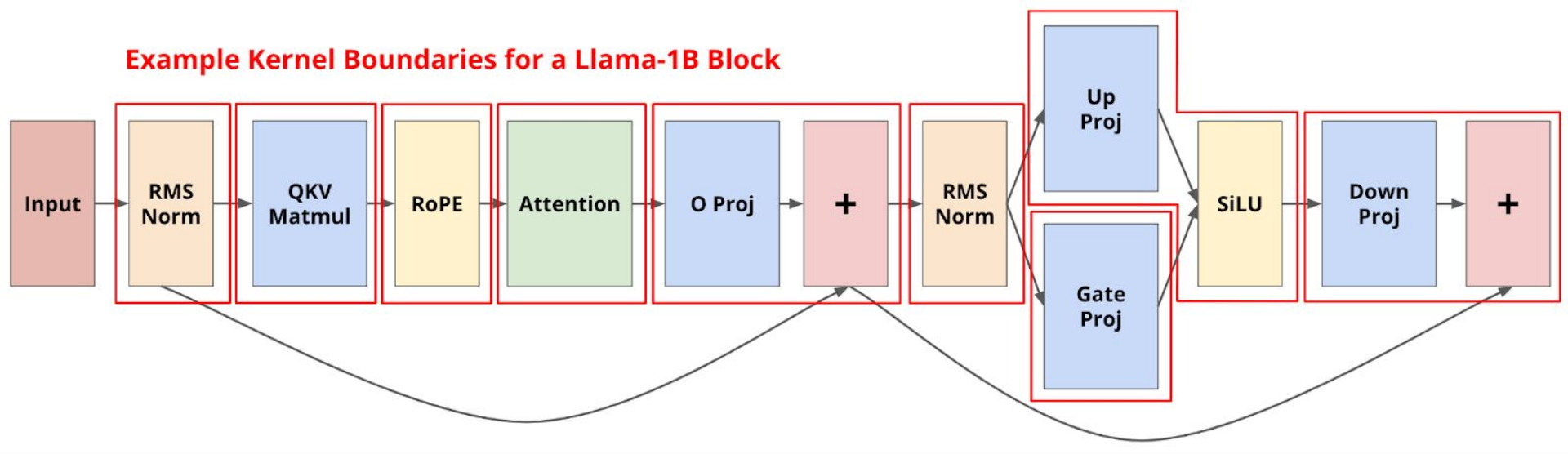

```
data = open("input.dat");          # read the data on the CPU
copyToGPU(data);                   # copy the data to the GPU
matrix_inverse(data.gpu);          # perform a matrix operation on the GPU
copyFromGPU(data);                 # copy the resulting output to the CPU
write(data, "output.dat");         # write the output to file on the CPU
```



https://www.researchgate.net/figure/CUDA-Grid-Block-Thread-Structure-1-2_fig1_300080119

- Thousands of specialised compute cores
- Latency is a fuzzy concept
- Many CUDA kernels are launched for various LLM operations

Kernel boundaries require synchronization and copies



What insights can eBPF help provide here?

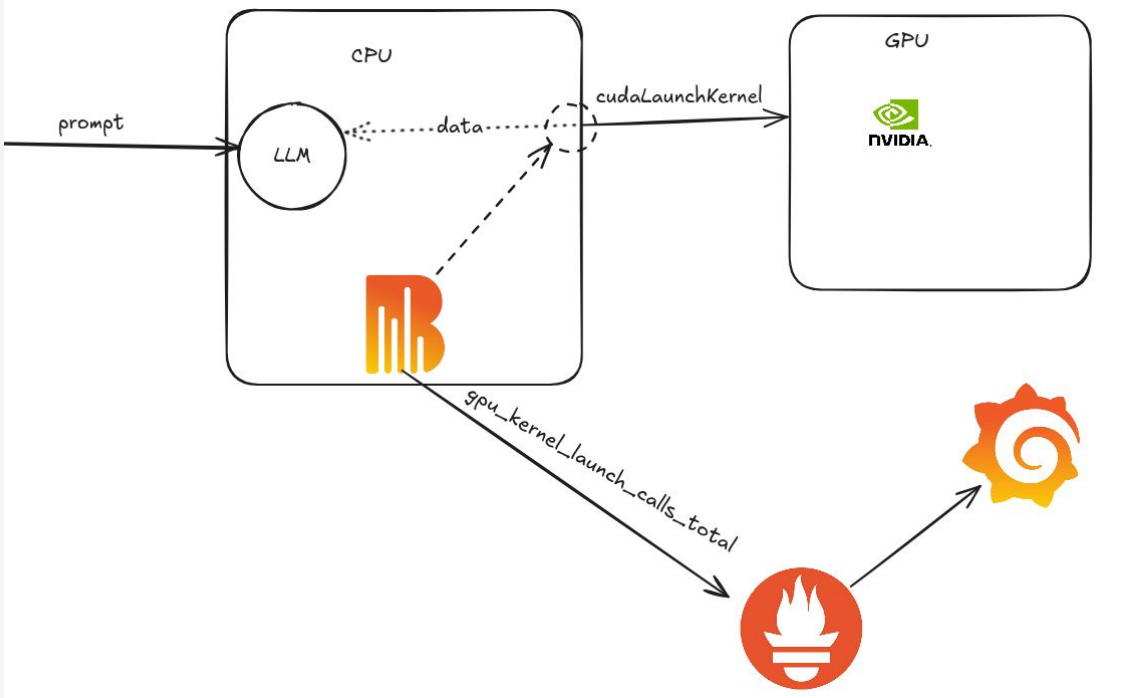
What and why eBPF?

eBPF allows programs to run within the Linux kernel in a safe and efficient manner

- Zero-code instrumentation
- Launch at any point in time
- Framework agnostic
- Extremely low overhead
- Collect data from multiple clusters and aggregate in one place
  (you can annotate easily with pod/service/cluster metadata)

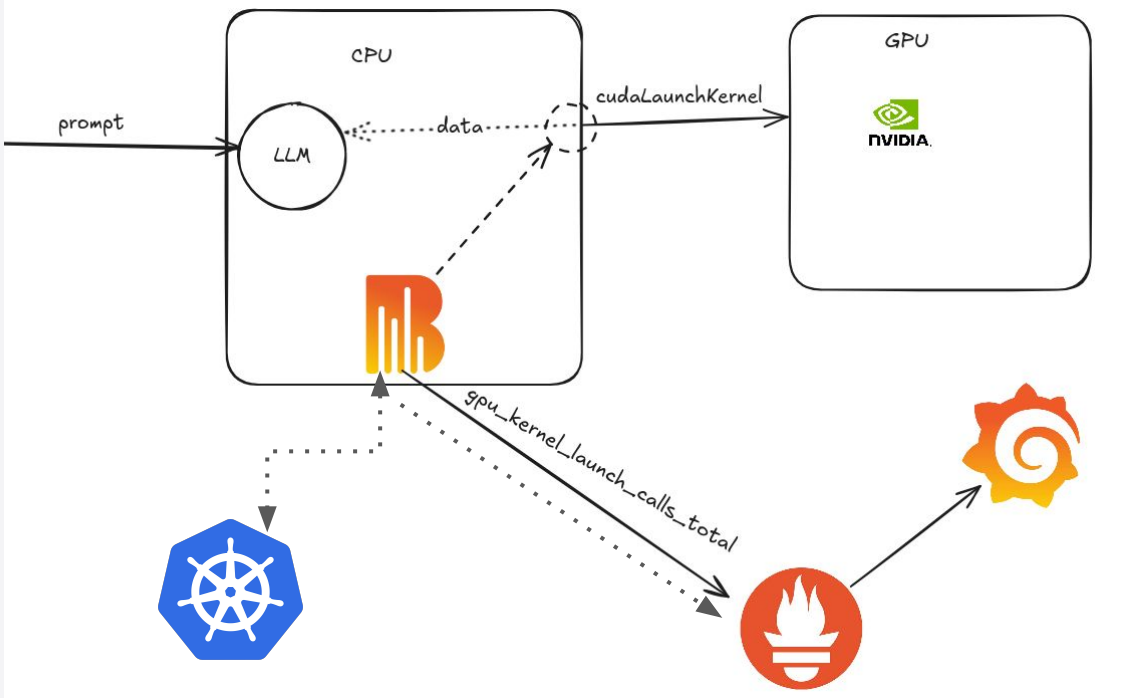
OpenTelemetry eBPF Instrumentation/Beyla

Adds probes to `cudaLaunchKernel`, `cudaMemcpy`...

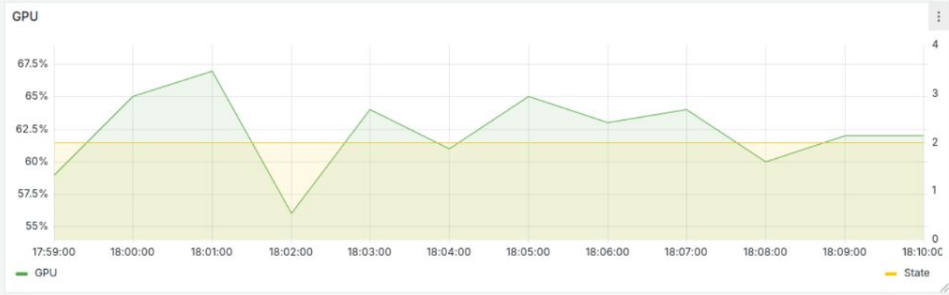


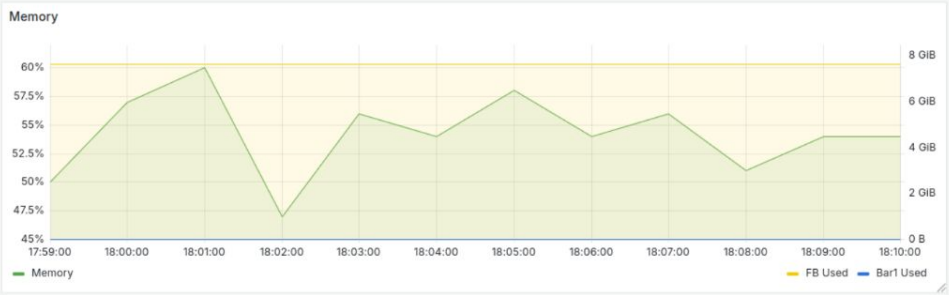
OpenTelemetry eBPF Instrumentation/Beyla

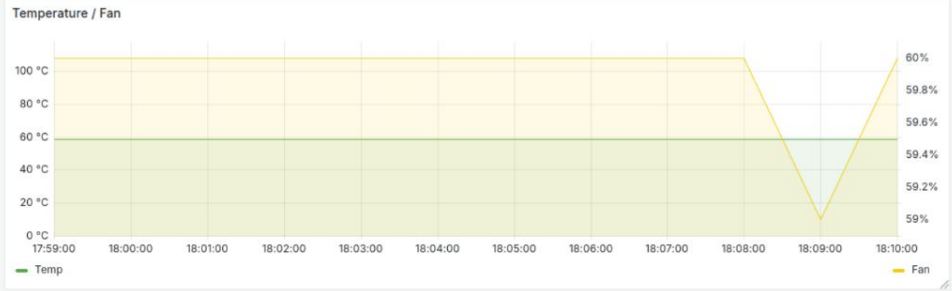
We talk to the Kubernetes API to enrich the eBPF data (From PIDs to PODs)



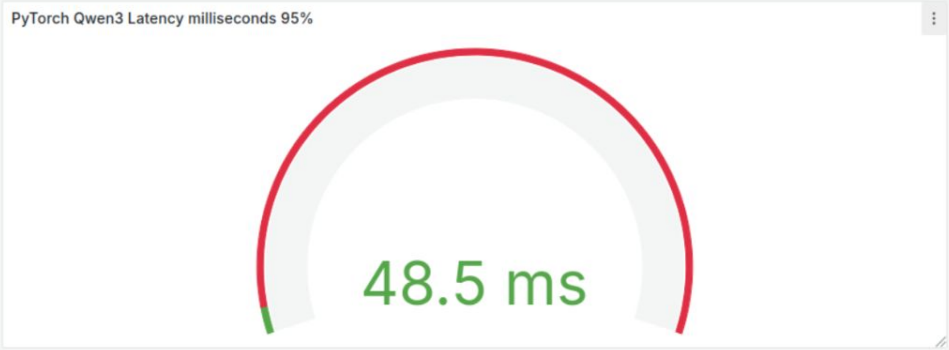
NVIDIA 3090 running Qwen3 PyTorch (GPU counters)

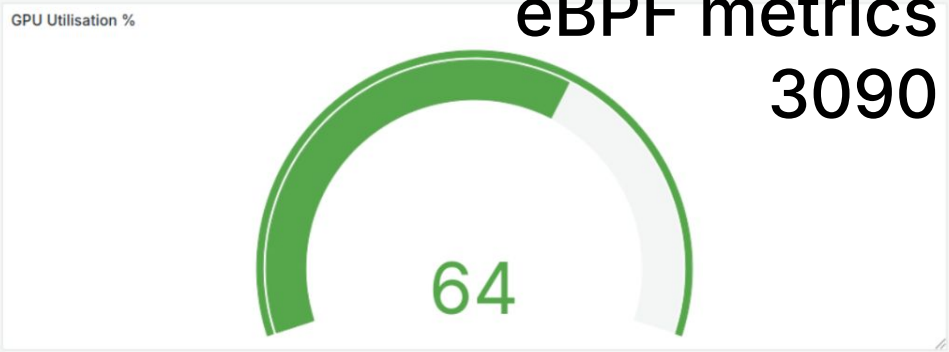


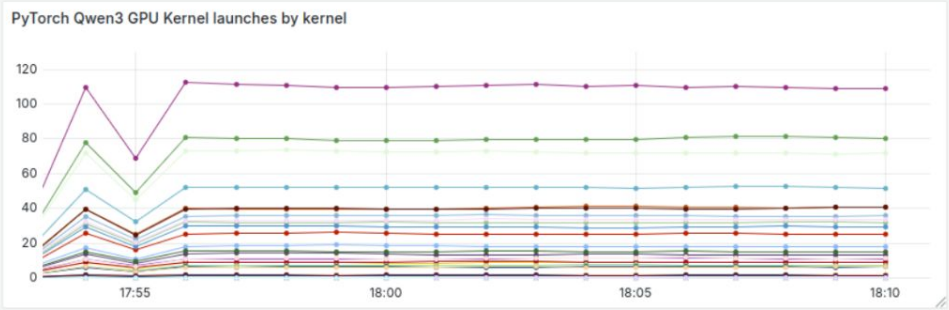


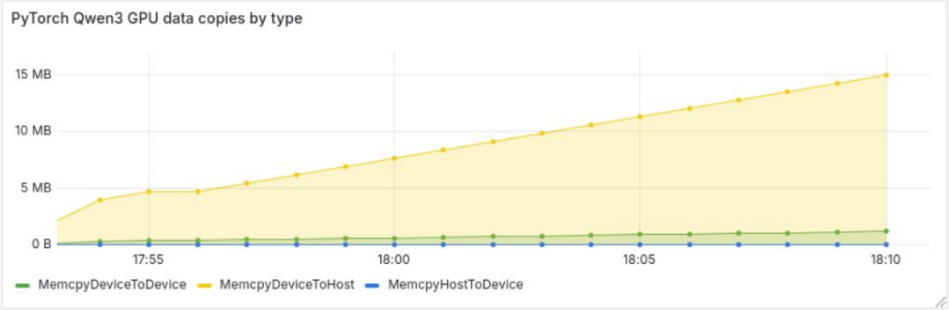


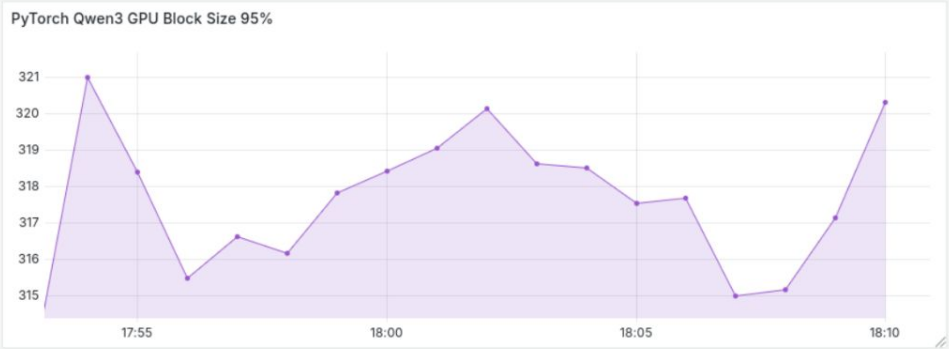
eBPF metrics 3090









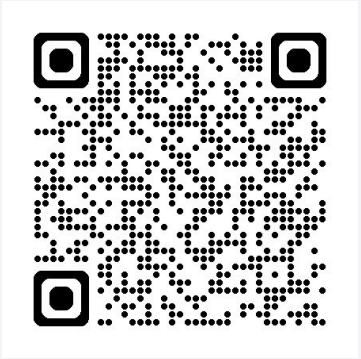


Can we optimise this?

Megakernels allow us to optimise the cuda kernel launches and memory copies

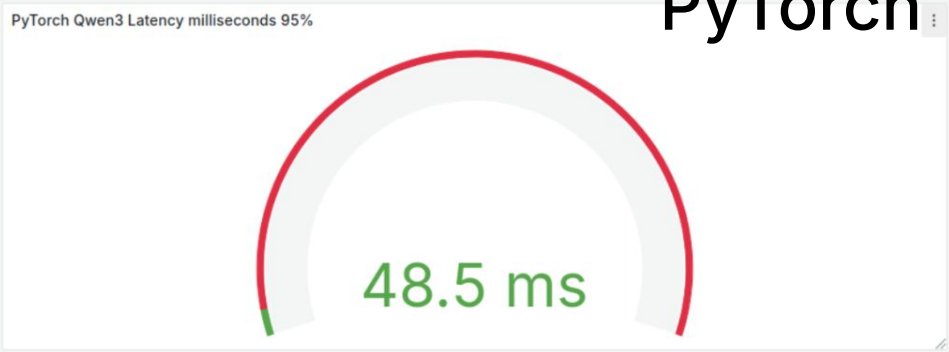
- Manual approaches (ThunderKittens — Stanford)
  https://hazyresearch.stanford.edu/blog/2025-05-27-no-bubbles

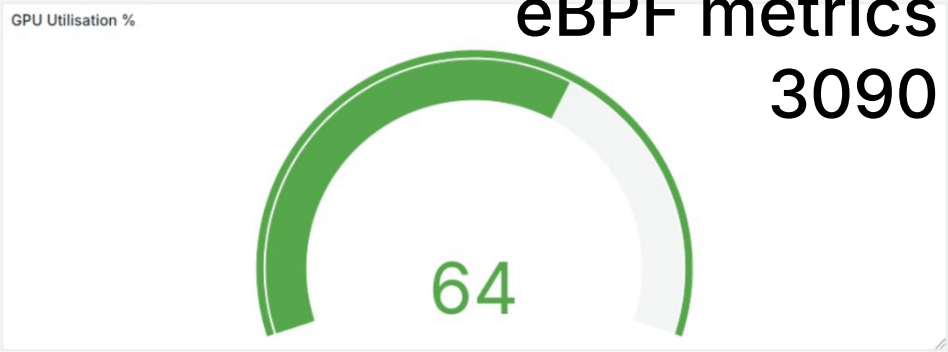
- Automatic (Mirage – Carnegie Mellon)
  https://github.com/mirage-project/mirage

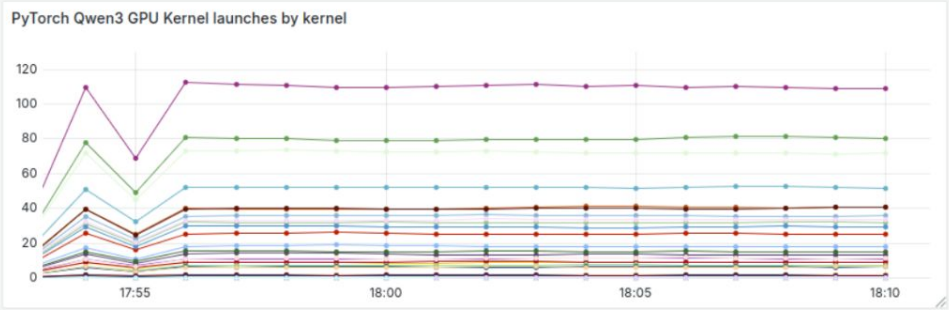


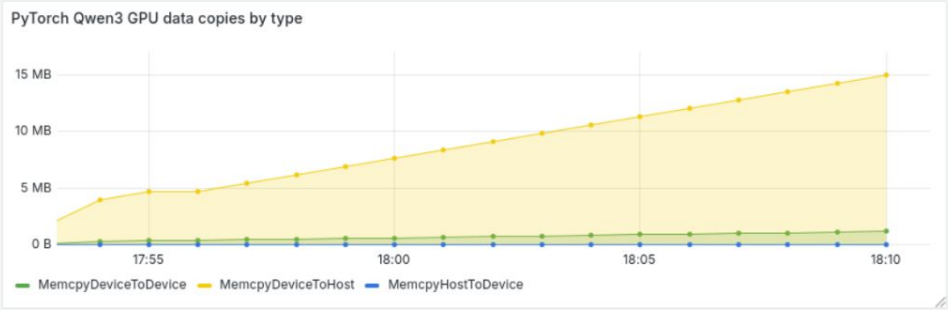
PyTorch

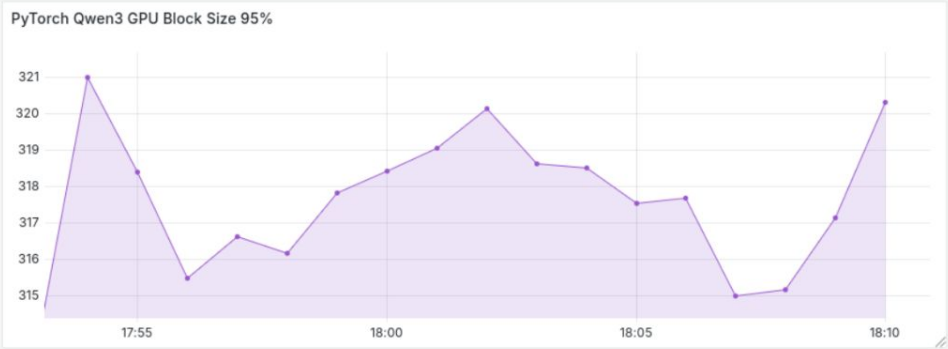
eBPF metrics
3090





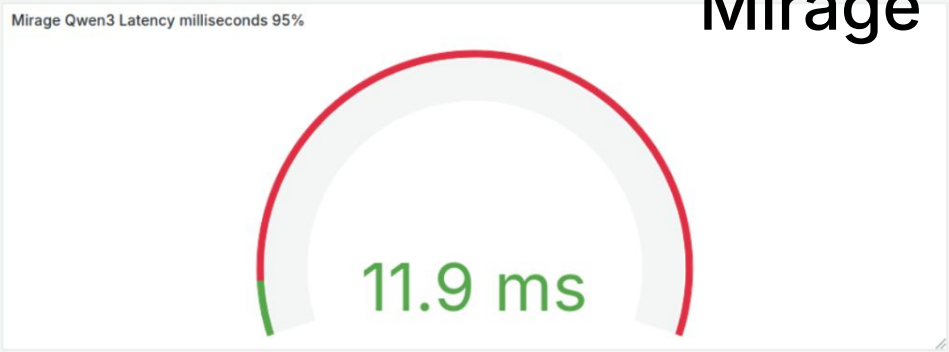


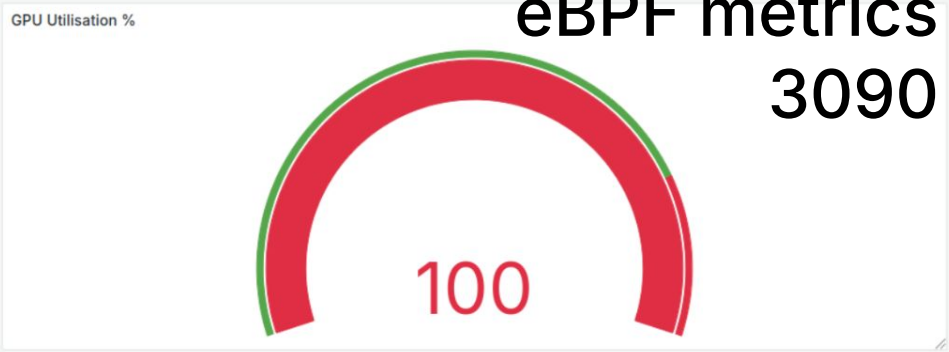


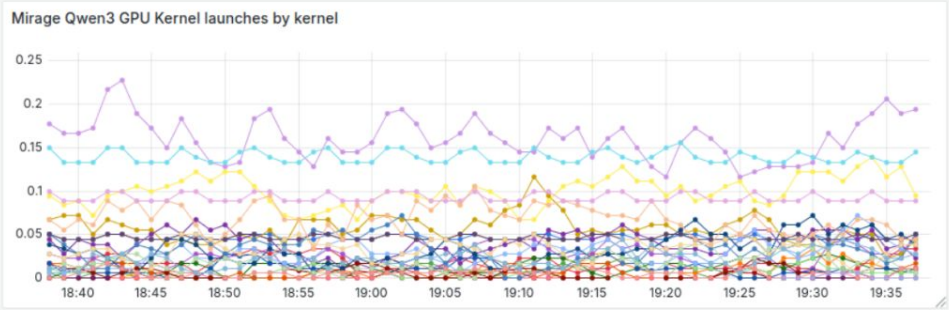


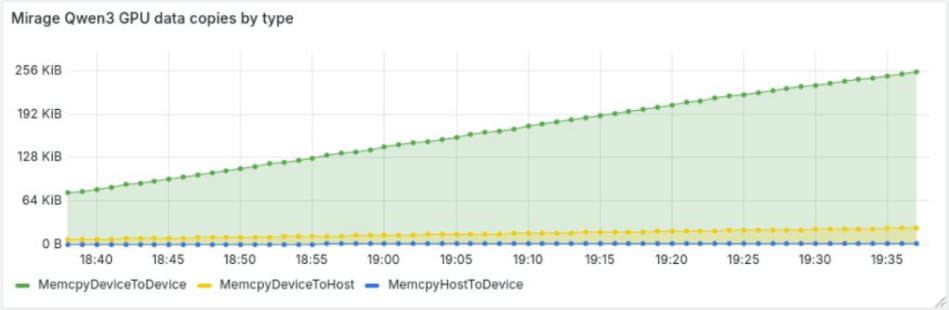
Mirage

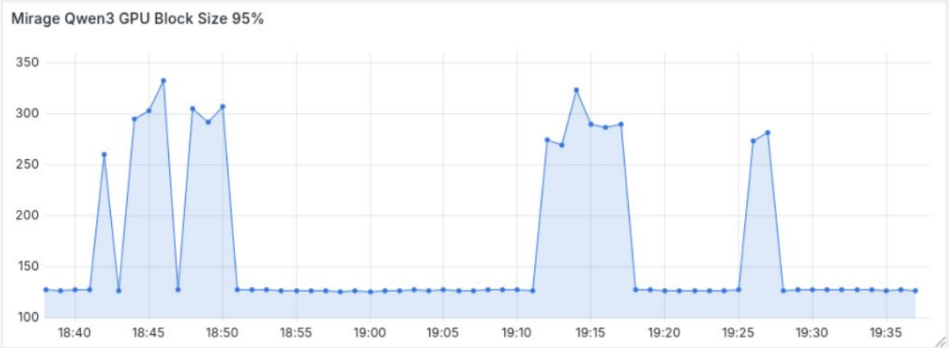
eBPF metrics 3090





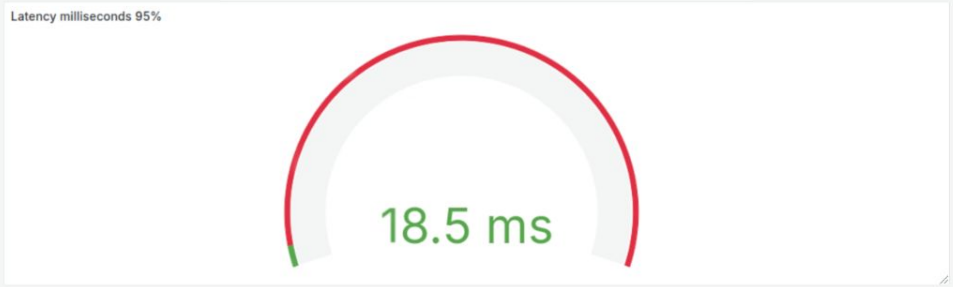


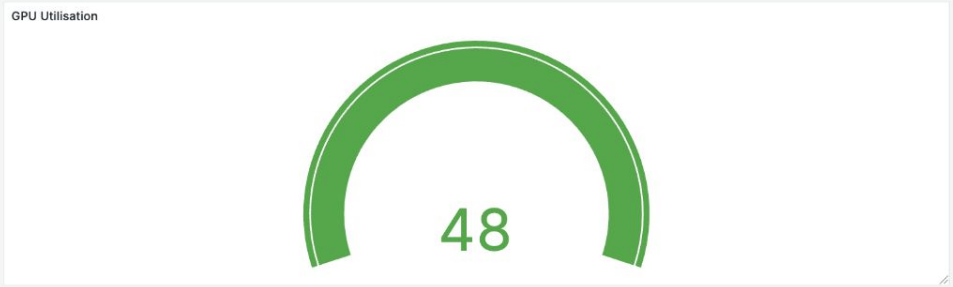




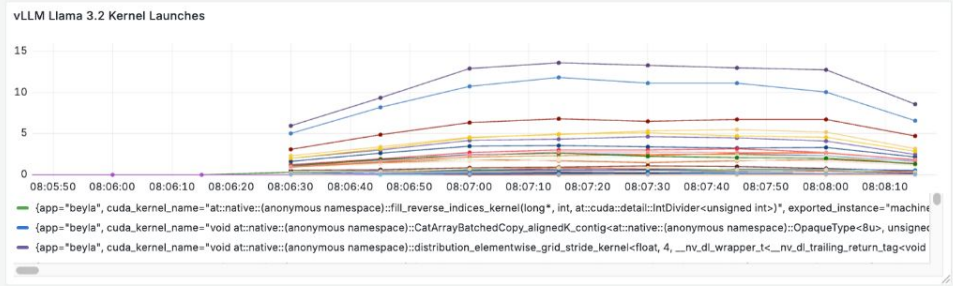
Llama 3.2 vLLM

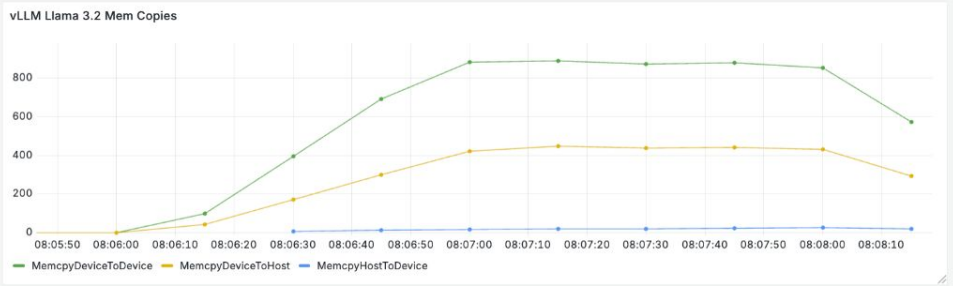
eBPF metrics A100

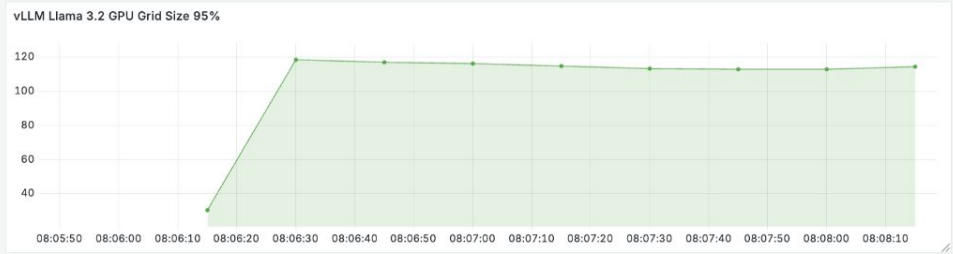




Beyla Metrics

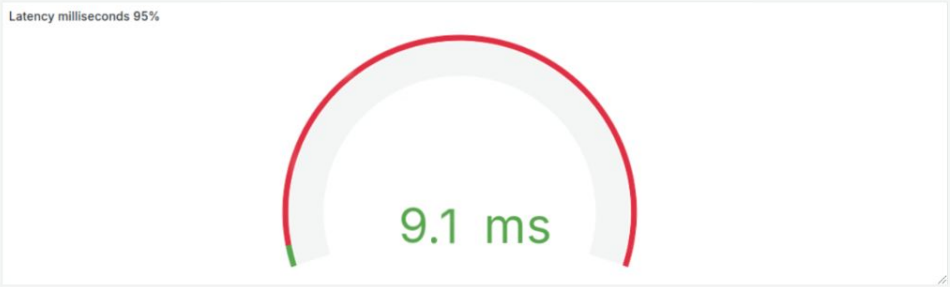


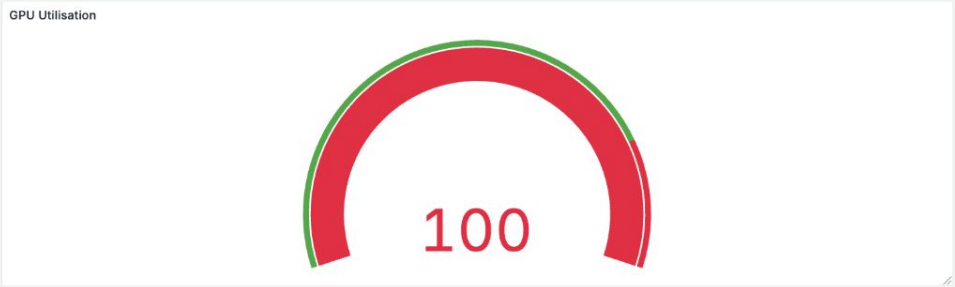




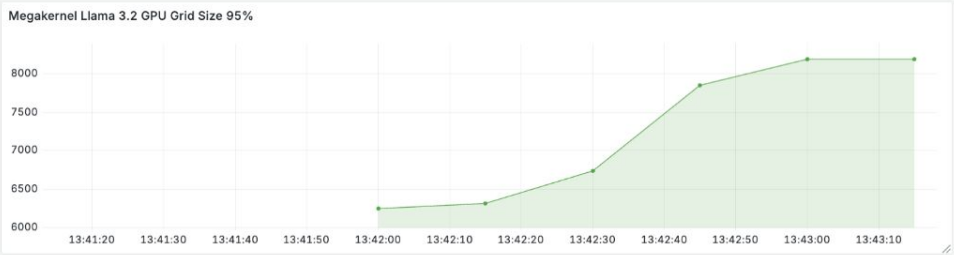
Llama 3.2 ThunderKittens

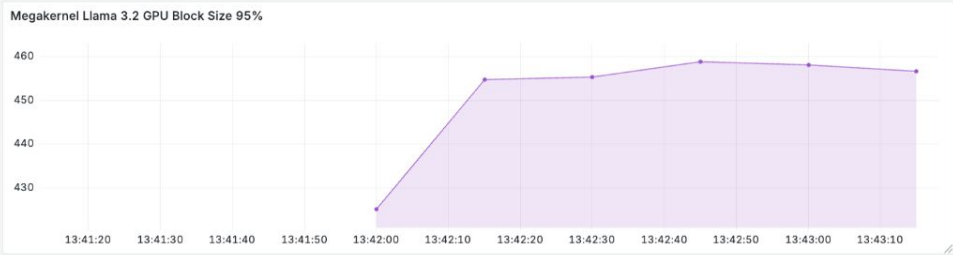
eBPF metrics A100





Beyla Metrics





Summary

GPU instrumentation is important and also tricky

- CPU concepts don’t quite apply, we need better tools

eBPF can help us get insights easily

- Zero-code, framework agnostic way to get deep insights

Optimising helps us win big

- There’s a lot of low hanging fruit if you can see it

Grafana Labs

Thank you

Check out Beyla and OpenTelemetry eBPF Instrumentation


Grafana Labs