

HOW TO TRAIN YOUR TINY MOE

Or: how to optimize your training loop as far as you can

MOTIVATION

- With the rise of small models, research at home should be a thing

MOTIVATION

- With the rise of small models, research at home should be a thing
- Personally, pretraining is something I've always wanted to do

MOTIVATION

- With the rise of small models, research at home should be a thing
- Personally, pretraining is something I've always wanted to do
- I have a genuine love for making things faster

HOW FAST IS FAST ENOUGH?

HOW FAST IS FAST ENOUGH?

- MFU: How well your model utilizes your GPU
- MFU at home (~30-35% is doing VERY good)
- 4x RTX 6000 Blackwell Max-Q's
- No NVLink/fancy tricks

APPENDIX A: Blackwell GB202 GPU

Table 4. RTX A6000 vs RTX 6000 Ada vs RTX PRO 6000 Blackwell Full Specs

Graphics Card	RTX A6000	RTX 6000 Ada Generation	RTX PRO 6000 Blackwell Max-Q Workstation Edition	RTX PRO 6000 Blackwell Workstation Edition
GPU Codename	GA102	AD102	GB202	GB202
GPU Architecture	NVIDIA Ampere	NVIDIA Ada Lovelace	NVIDIA Blackwell	NVIDIA Blackwell
GPCs	7	12	12	12
TPCs	42	71	94	94
SMs	84	142	188	188
CUDA Cores / SM	128	128	128	128
CUDA Cores / GPU	10752	18176	24064	24064
Tensor Cores / SM	4 (3rd Gen)	4 (4th Gen)	4 (5th Gen)	4 (5th Gen)
Tensor Cores / GPU	336	568 (4th Gen)	752 (5th Gen)	752 (5th Gen)
Peak FP16 Tensor TFLOPS with FP32 Accumulate ¹	154.8/309.6 ²	364/728 ²	438.9/877.9 ²	503.8/1007.6 ²
Peak BF16 Tensor TFLOPS with FP32 Accumulate ¹	154.8/309.6 ²	364/728 ²	438.9/877.9 ²	503.8/1007.6 ²

HOW FAST IS FAST ENOUGH?

- ~12 hours total (to Chinchilla)
 - Chinchilla == $20 * \text{XB params}$
 - Fast enough to test data iterations
 - 2x experiments per day
 - Vastly limits the size of the model I can use
- What models can I use?

HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscale/training-time-calculator like 5 • Running | Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)
 B T

Unit
Model size unit
 B T

Calculation Breakdown:

- GPU Selection: 6000 Blackwell MaxQ (438.9 TFLOPs)
- Model Size: 8.0B parameters (8.0B)
- Training Tokens: 160B tokens (160000M)
- Total FLOPs: 7.68e+21 FLOPs
- Formula: $6 \times 8.0B \text{ params} \times 160000M \text{ tokens}$
- Effective TFLOPs/s: 614.46 TFLOPs/s
- Formula: $438.9 \text{ TFLOPs/GPU} \times 4 \text{ GPUs} \times 35\% \text{ MFU}$

Training Time:
4.82 months (144.7 days, 3472 hours)

GPU Model
Select a GPU model from the list

Number of GPUs
Total number of GPUs for training
 4 1024

Training Tokens
Number of training tokens
 M B T

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (50% is typical for low-end estimate)
 35 100

Use GPU Model from List

HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscale/training-time-calculator 13 like 5 • Running 0 Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)
 Unit
 Model size unit
 B
 T

Check to select a GPU model, uncheck to input custom TFLOPs
 Use GPU Model from List

GPU Model
Select a GPU model from the list
6000 Blackwell MaxQ

Number of GPUs
Total number of GPUs for training
 Unit
 GPU count unit
 M
 B
 T

Training Tokens
Number of training tokens
 Unit
 Token count unit
 M
 B
 T

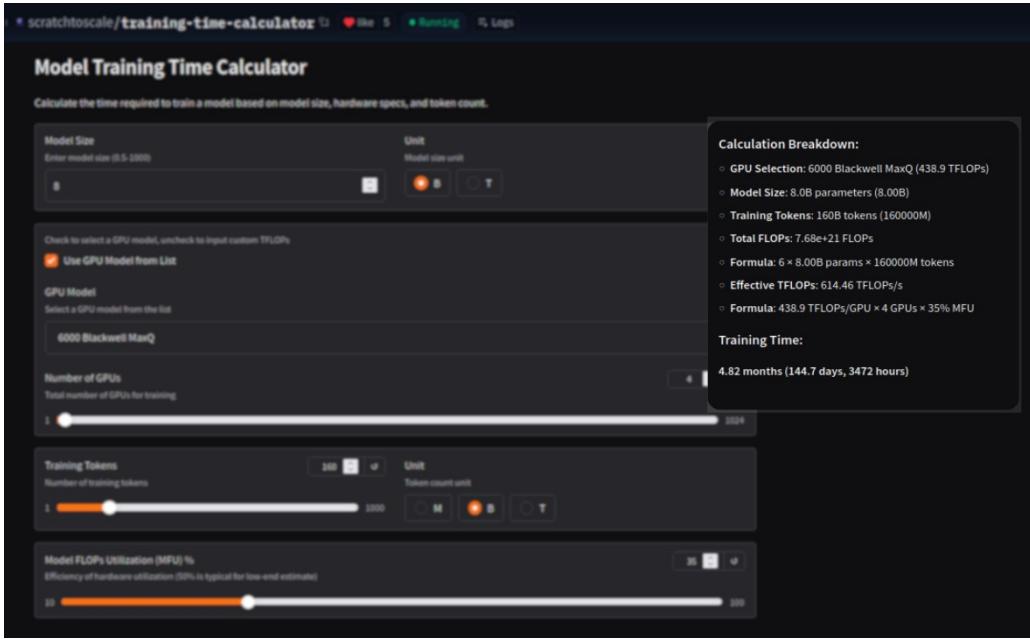
Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (35%) is typical for low-end estimate
 Unit
 MFU % unit
 GPU %

Calculation Breakdown:

- GPU Selection: 6000 Blackwell MaxQ (438.9 TFLOPs)
- Model Size: 8.0B parameters (8.00B)
- Training Tokens: 160B tokens (160000M)
- Total FLOPs: 7.68e+21 FLOPs
- Formula: $6 \times 8.00B \text{ params} \times 160000M \text{ tokens}$
- Effective TFLOPs: 614.46 TFLOPs/s
- Formula: $438.9 \text{ TFLOPs/GPU} \times 4 \text{ GPUs} \times 35\% \text{ MFU}$

Training Time:

4.82 months (144.7 days, 3472 hours)



HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscate/training-time-calculator like 5 Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)

 B M T

Unit
Model size unit
 B M T

Calculation Breakdown:

- GPU Selection: H100 (1000.0 TFLOPs)
- Model Size: 8.0B parameters (8.00B)
- Training Tokens: 160B tokens (160000M)
- Total FLOPs: 7.68e+21 FLOPs
- Formula: $6 \times 8.00B \text{ params} \times 160000M \text{ tokens}$
- Effective TFLOPs: 4400.00 TFLOPs/s
- Formula: $1000.0 \text{ TFLOPs/GPU} \times 8 \text{ GPUs} \times 55\% \text{ MFU}$

Training Time:
20.20 days (484.8 hours)

GPU Model
Select a GPU model from the list.

Number of GPUs
Total number of GPUs for training

 GPU TPU

Training Tokens
Number of training tokens

 B M T

Unit
Token count unit
 M B T

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (50% is typical for low-end estimate)

 GPU TPU

Unit
MFU % unit
 GPU TPU CPU

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (50% is typical for low-end estimate)

 GPU TPU CPU

HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscale/training-time-calculator like 5 • Running | Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)
 B T

Unit
Model size unit

Calculation Breakdown:

- GPU Selection: 6000 Blackwell MaxQ (438.9 TFLOPs)
- Model Size: 1.0B parameters (1.0B)
- Training Tokens: 20B tokens (20000M)
- Total FLOPs: 1.20e+20 FLOPs
- Formula: $6 \times 1.00B \text{ params} \times 20000M \text{ tokens}$
- Effective TFLOPs: 614.46 TFLOPs/s
- Formula: $438.9 \text{ TFLOPs/GPU} \times 4 \text{ GPUs} \times 35\% \text{ MFU}$

Training Time:
2.26 days (54.2 hours)

GPU Model
Select a GPU model from the list

Number of GPUs
Total number of GPUs for training
 1024

Training Tokens
Number of training tokens
 M B T

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (50% is typical for low-end estimate)
 100

100

HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscale/training-time-calculator like 5 • Running | Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)

Unit
Model size unit

Calculation Breakdown:

- GPU Selection: 6000 Blackwell MaxQ (438.9 TFLOPs)
- Model Size: 500M parameters (0.50B)
- Training Tokens: 10B tokens (10000M)
- Total FLOPs: 3.00e+19 FLOPs
- Formula: $6 \times 0.50B \text{ params} \times 10000M \text{ tokens}$
- Effective TFLOPs: 614.46 TFLOPs/s
- Formula: $438.9 \text{ TFLOPs/GPU} \times 4 \text{ GPUs} \times 35\% \text{ MFU}$

Training Time:
13.56 hours

GPU Model
Select a GPU model from the list

Number of GPUs
Total number of GPUs for training
 1024

Training Tokens
Number of training tokens

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (50% is typical for low-end estimate)
 100

HOW FAST IS FAST ENOUGH?

The Depression Calculator™

* scratchtoscale/training-time-calculator 13 like 5 • Running 5 Logs

Model Training Time Calculator

Calculate the time required to train a model based on model size, hardware specs, and token count.

Model Size
Enter model size (0.5-1000)
0.5 Unit Model size unit
 B M T

Check to select a GPU model, uncheck to input custom TFLOPs
 Use GPU Model from List

GPU Model
Select a GPU model from the list
6000 Blackwell MaxQ

Number of GPUs
Total number of GPUs for training
1 Unit Number of GPUs

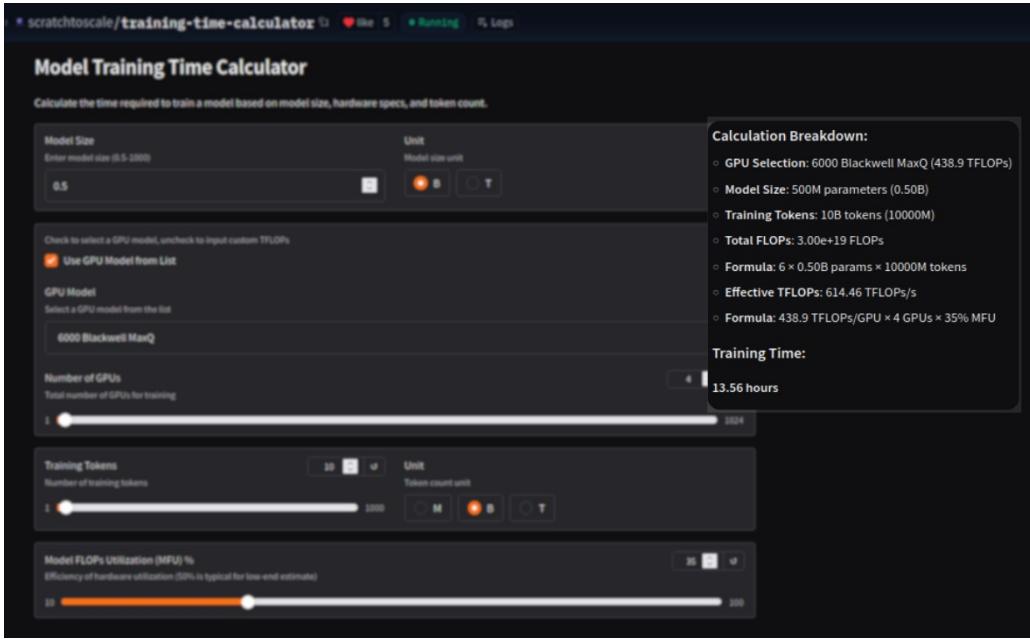
Training Tokens
Number of training tokens
1000 Unit Token count unit
 M B T

Model FLOPs Utilization (MFU) %
Efficiency of hardware utilization (35% is typical for low-end estimate)
35 Unit Model FLOPs Utilization (MFU) %
10 20 30 40 50 60 70 80 90 100

Calculation Breakdown:

- GPU Selection: 6000 Blackwell MaxQ (438.9 TFLOPs)
- Model Size: 500M parameters (0.50B)
- Training Tokens: 10B tokens (10000M)
- Total FLOPs: 3.00×10^{19} FLOPs
- Formula: $6 \times 0.50B \text{ params} \times 10000M \text{ tokens}$
- Effective TFLOPs: 614.46 TFLOPs/s
- Formula: $438.9 \text{ TFLOPs/GPU} \times 4 \text{ GPUs} \times 35\% \text{ MFU}$

Training Time:
13.56 hours



THE LONG ROAD TO 13 HOURS

THE MODEL

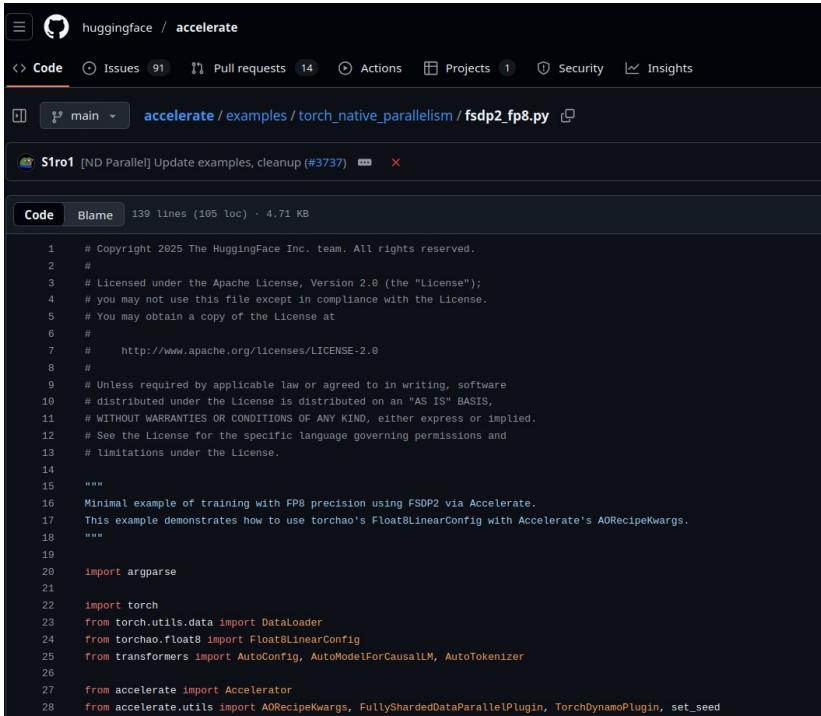
- Qwen3-MoE-500M-A330M
- Most of that 330M comes from embeddings
- At a vocab of only 32k it'd be considered Qwen3-MoE-500M-A120M

```
1 model_type: qwen3_moe
2 architectures: ["Qwen3MoeForCausalLM"]
3
4 vocab_size: 151936
5 hidden_size: 896
6 num_hidden_layers: 12
7 num_attention_heads: 16
8 num_key_value_heads: 16
9 hidden_act: silu
10
11 # === MoE ===
12 num_experts: 16
13 num_experts_per_tok: 2
14 moe_intermediate_size: 512
15
16 # === Dense MLP baseline ===
17 intermediate_size: 4096
```


ESTABLISH A BASELINE ON THE SMALLEST NODE POSSIBLE

- Focus on saving \$\$\$
- Focus on saving time
- Ensure that small variations work then scale

ESTABLISH A BASELINE ON THE SMALLEST NODE POSSIBLE



The screenshot shows a GitHub code editor interface. The repository is 'huggingface / accelerate'. The file path is 'accelerate / examples / torch_native_parallelism / fsdp2_fp8.py'. A pull request titled 'S1ro1 [IND Parallel] Update examples, cleanup (#3737)' is visible. The code editor displays the contents of the 'fsdp2_fp8.py' file, which includes a copyright notice, a license (Apache License, Version 2.0), and Python code for training with FP8 precision using FSDP2 via Accelerate.

```
1 # Copyright 2025 The HuggingFace Inc. team. All rights reserved.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 """
15 """
16 Minimal example of training with FP8 precision using FSDP2 via Accelerate.
17 This example demonstrates how to use torchao's Float8LinearConfig with Accelerate's AORecipeKwargs.
18 """
19
20 import argparse
21
22 import torch
23 from torch.utils.data import DataLoader
24 from torchao.float8 import Float8LinearConfig
25 from transformers import AutoConfig, AutoModelForCausalLM, AutoTokenizer
26
27 from accelerate import Accelerator
28 from accelerate.utils import AORecipeKwargs, FullyShardedDataParallelPlugin, TorchDynamoPlugin, set_seed
```

ESTABLISH A BASELINE ON THE SMALLEST NODE POSSIBLE

Raw BF16 using a core script:

- Batch size of 12
- Iterates through 200 batches
- Sequence length of 4096
- 9.8M tokens

Results:

- 44,942 tok/s
- Time to 10B tokens: **61.80 hours**

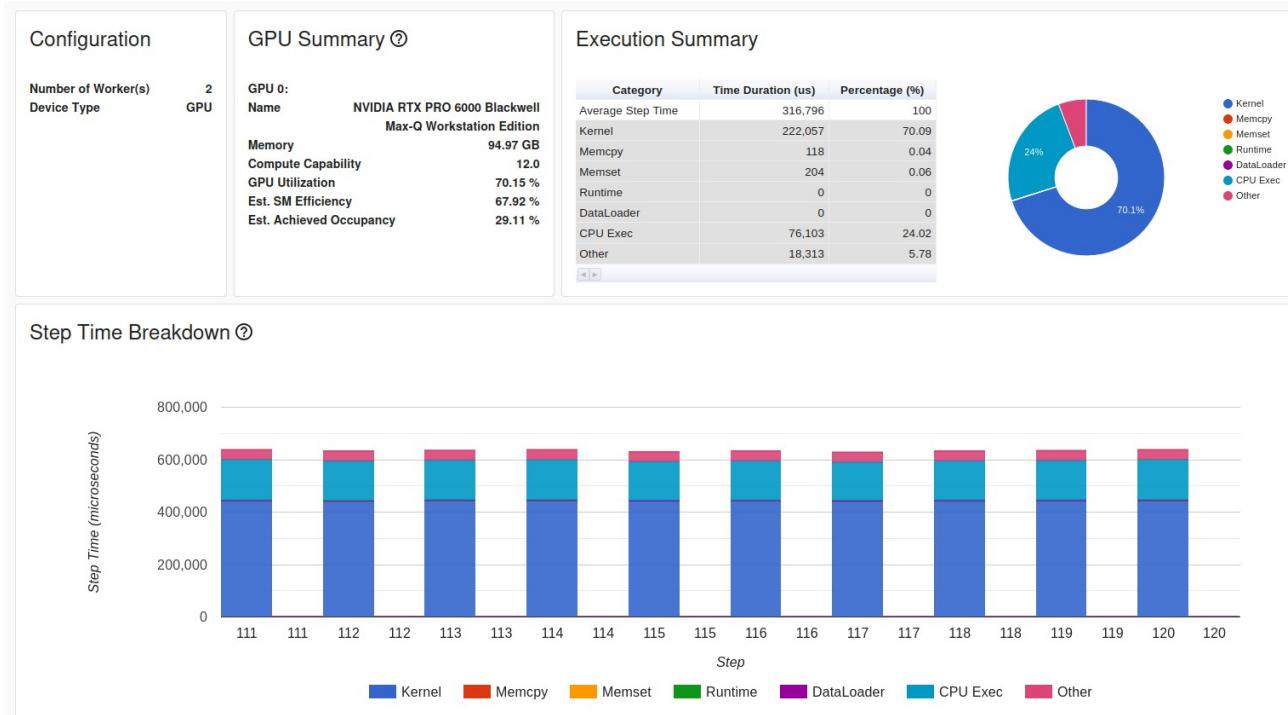
IS MAXIMUM BATCH SIZE WORTH IT?

- Batch size of 8
- Sequence length of 4096
- 6.5M tokens

Results:

- ~~44,942~~ 45,005 tok/s
- Time to 10B tokens: ~~61.80 hours~~ **61.72 hrs**

IS MAXIMUM BATCH SIZE WORTH IT?



LET'S TRY A DIFFERENT ATTENTION MECHANISM

```
1     model = AutoModelForCausalLM.from_config(  
2         AutoConfig.from_pretrained("./config.json", use_cache=False),  
3         torch_dtype=torch.bfloat16,  
4         attnImplementation="flex_attention"  
5     )
```

Results:

- ~~45,005~~ 59,037 tok/s
- Time to 10B tokens: ~~61.80 hours~~ 47.5 hrs

COULD IT BE THE DATA?



SIMPLE TEST: PIN_MEMORY=TRUE



WHAT IF WE MINIMIZE THE CPU WHATSOEVER?



WHAT IF WE MINIMIZE THE CPU WHATSOEVER?

Pin memory:

- ~~59,037~~ 59,207 tok/s
- Time to 10B tokens: ~~47.5 hours~~ **46.91 hrs**

WHAT IF WE MINIMIZE THE CPU WHATSOEVER?

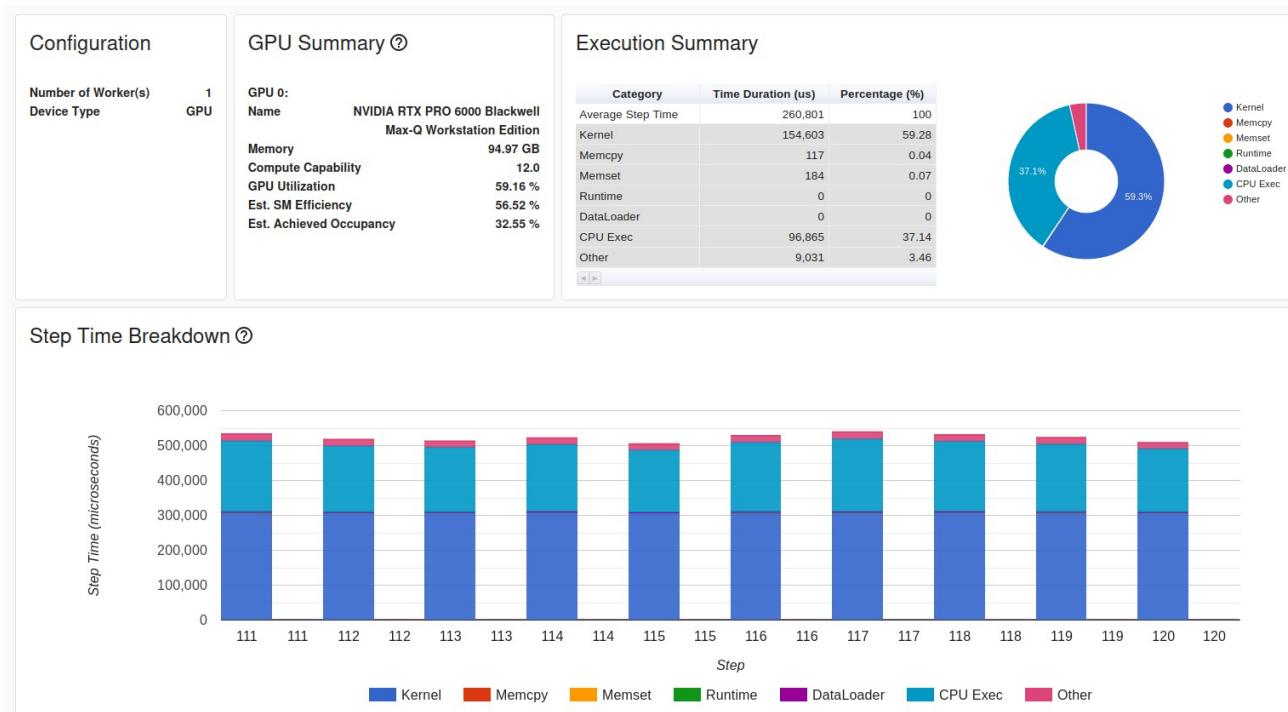
Pin memory:

- ~~59,037~~ 59,207 tok/s
- Time to 10B tokens: ~~47.5 hours~~ **46.91 hrs**

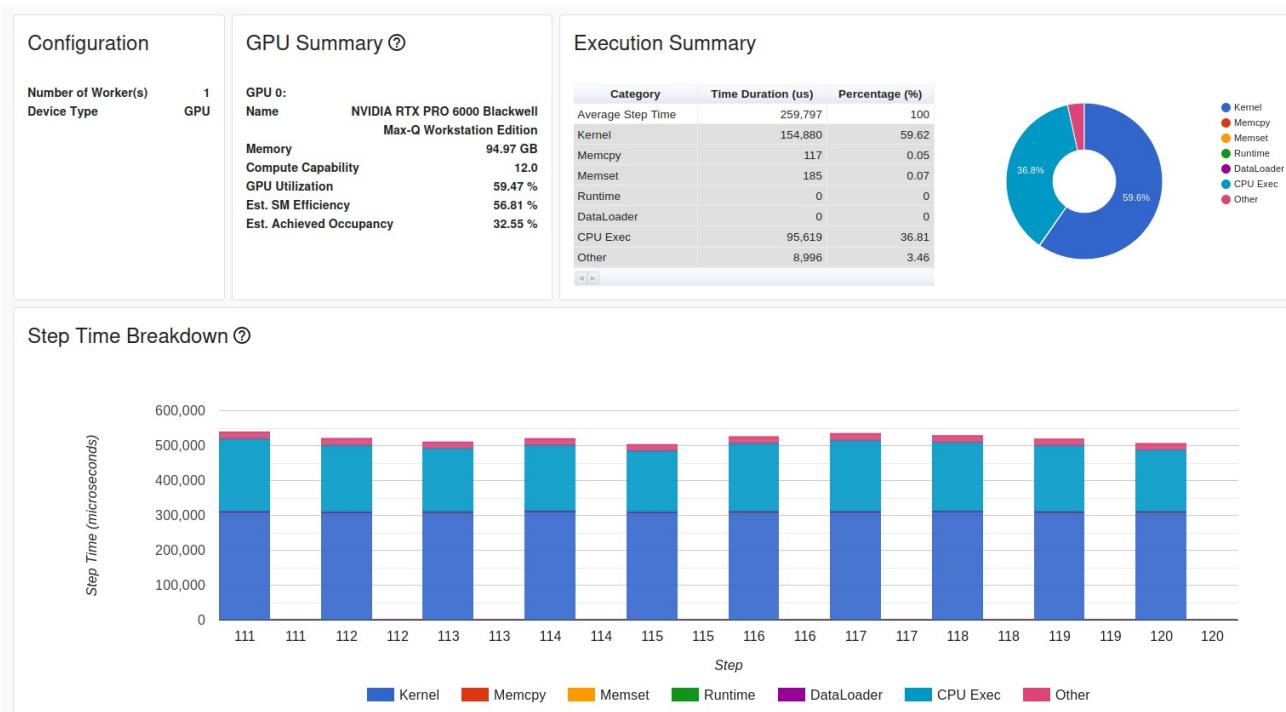
Pre-tokenize:

- ~~59,037~~ 62,840 tok/s
- Time to 10B tokens: ~~46.91 hours~~ **44.2 hrs**

LET'S PLAY WITH THE DATA MORE: CUDA BUFFERS (BEFORE)



LET'S PLAY WITH THE DATA MORE: CUDA BUFFERS (AFTER)



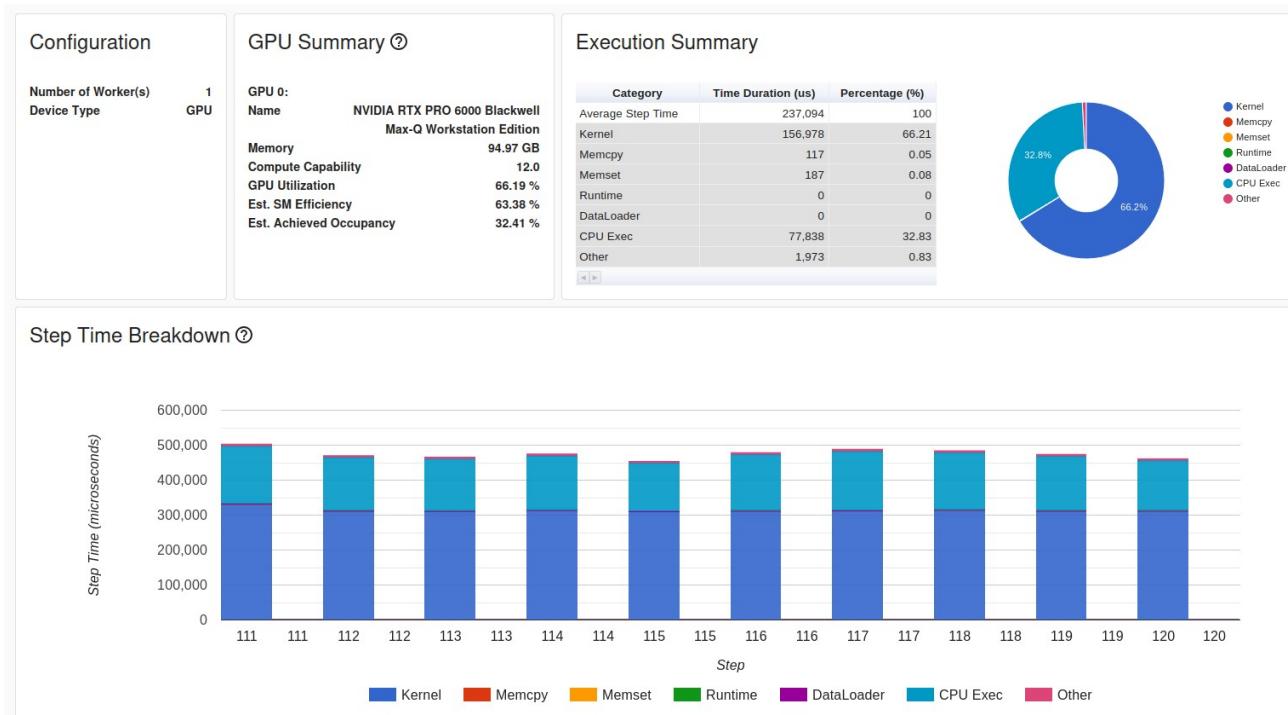
LET'S PLAY WITH THE DATA MORE: CUDA BUFFERS

```
1 for step, batch in enumerate(dataloader):
2     if step >= total_num_steps:
3         break
4     with torch.cuda.stream(prefetch_stream):
5         next_batch = {
6             k: v.to(
7                 device=accelerator.device, non_blocking=True
8             )
9             for k, v in batch.items()
10            }
10    torch.cuda.current_stream().wait_stream(prefetch_stream)
11    batch = next_batch
12    outputs = model(**batch)
13    loss = outputs.loss
```

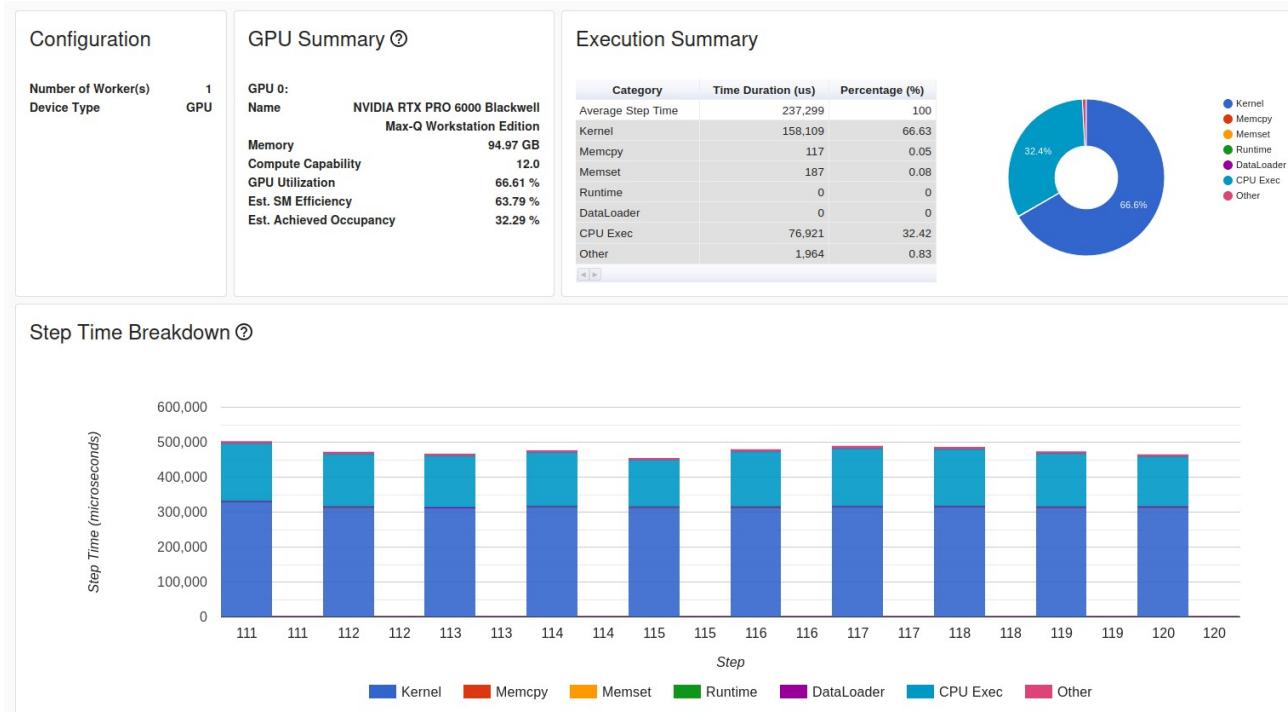
Results (not helpful):

- ~~62,840~~ 57,003 tok/s
- Time to 10B tokens: ~~44.2 hrs~~ **48.73 hrs**

OUT OF DATA IDEAS, THE OPTIMIZER!



LET'S FUSE A KERNEL (THANKS TO CLAUDE)



LET'S FUSE A KERNEL (THANKS TO CLAUDE)

```
1 class Qwen3MoeMLP(nn.Module):
2     def __init__(self, config, intermediate_size=None):
3         ...
4
5     def forward(self, x):
6         out = self.act_fn(self.gate_proj(x))
7         out *= self.up_proj(x)
8         down_proj = self.down_proj(out)
9
10    return down_proj
```

LET'S FUSE A KERNEL (THANKS TO CLAUDE)

```
1 class Qwen3MoeMLP(nn.Module):
2     def __init__(self, config, intermediate_size=None):
3         ...
4
5     def forward(self, x):
6         # single GEMM → splits into gate and up
7         gate_up = self.gate_up_proj(x)
8         gate, up = gate_up.chunk(2, dim=-1)
9         out = self.down_proj(self.act_fn(gate) * up)
10        return out
```

WE'VE GRADUATED TO MULTI-GPU!

DISTRIBUTED DATA PARALLEL

- See a linear-ish speedup
- 1 -> 4 GPUs, keeping local batch size
- 19.6M tokens seen across 200 batches

Results (vs baseline):

- ~~45,005~~ 118,168 tok/s
- Time to 10B tokens: ~~61.72 hrs~~ 23.5 hrs

APPLYING ALL OUR EARLIER TRICKS (BAR MODELING TRICKS)

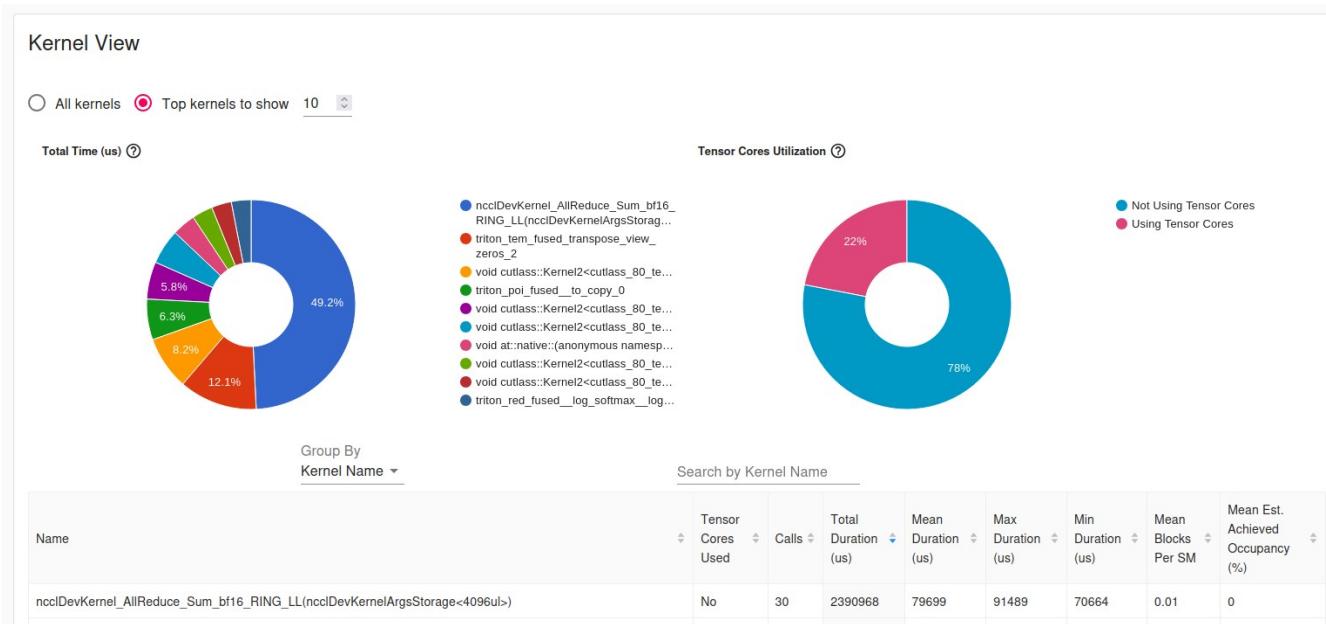
Results:

- ~~118,168~~ 154,870 tok/s
- Time to 10B tokens: ~~23.5 hrs~~ **17.94 hrs**

Now we're doing **really good**

What's our bottleneck?

APPLYING ALL OUR EARLIER TRICKS (BAR MODELING TRICKS)



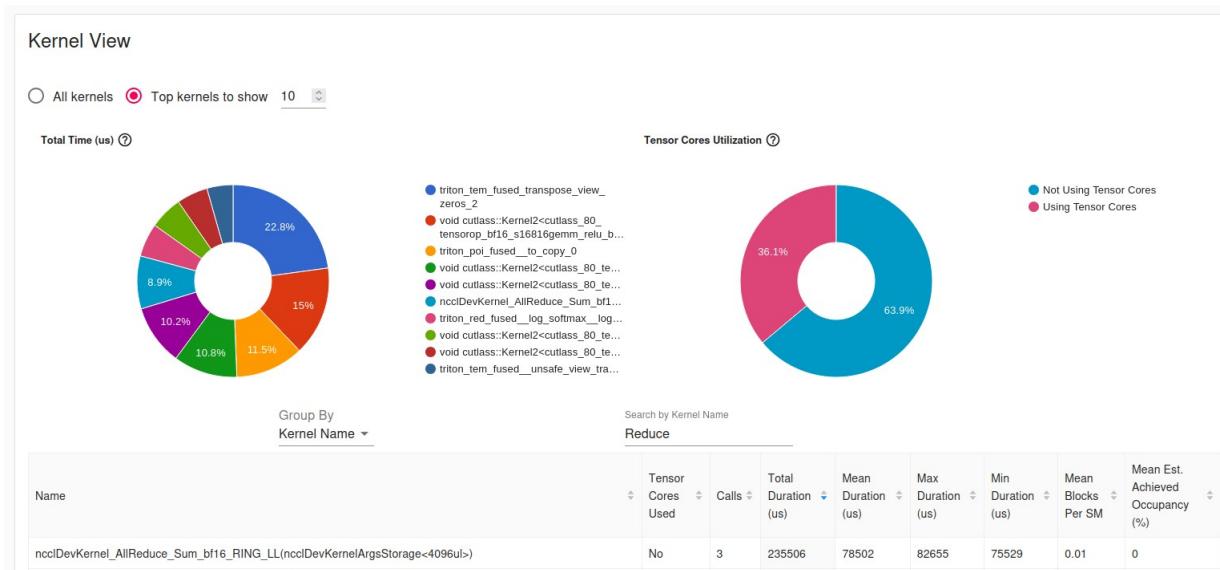
APPLYING ALL OUR EARLIER TRICKS (BAR MODELING TRICKS)

The 0.08s/all_reduce (on PCIe4, 0.04s on PCIe5)

- $10B / (6 * 4 * 4096) = 98,304 \text{ tok/batch}$
- $10B / 98,304 = 101,725 \text{ batches}$
- $0.08 * 101,725 = 2.26 \text{ hours purely on AllReduce}$

ACHIEVING THE TARGET BATCH SIZE

- Recommended to have a batch token amount of 1M
- Equates to ~grad accum of 10



ACHIEVING THE TARGET BATCH SIZE

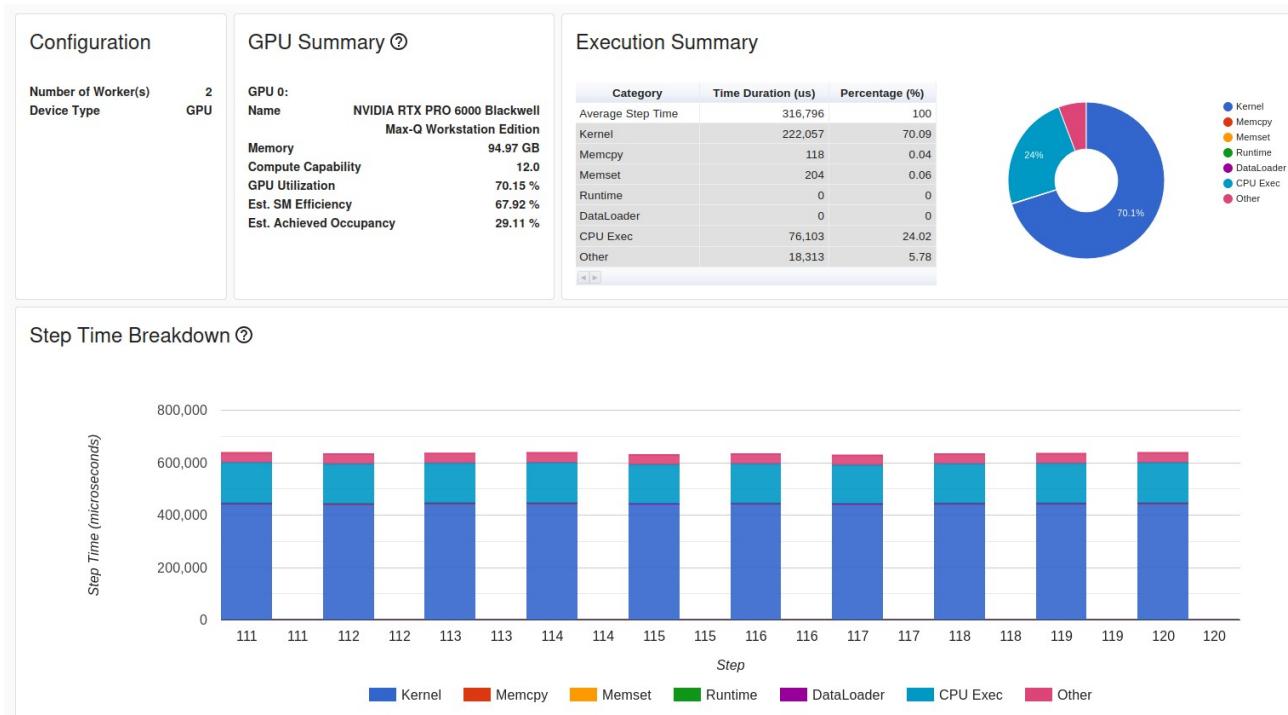
- Recommended to have a batch token amount of 1M
- Equates to ~grad accum of 10
- ~~154,870~~ 210,377 tok/s
- Time to 10B tokens: ~~17.94 hrs~~ **13.2 hrs**

LET'S RECAP

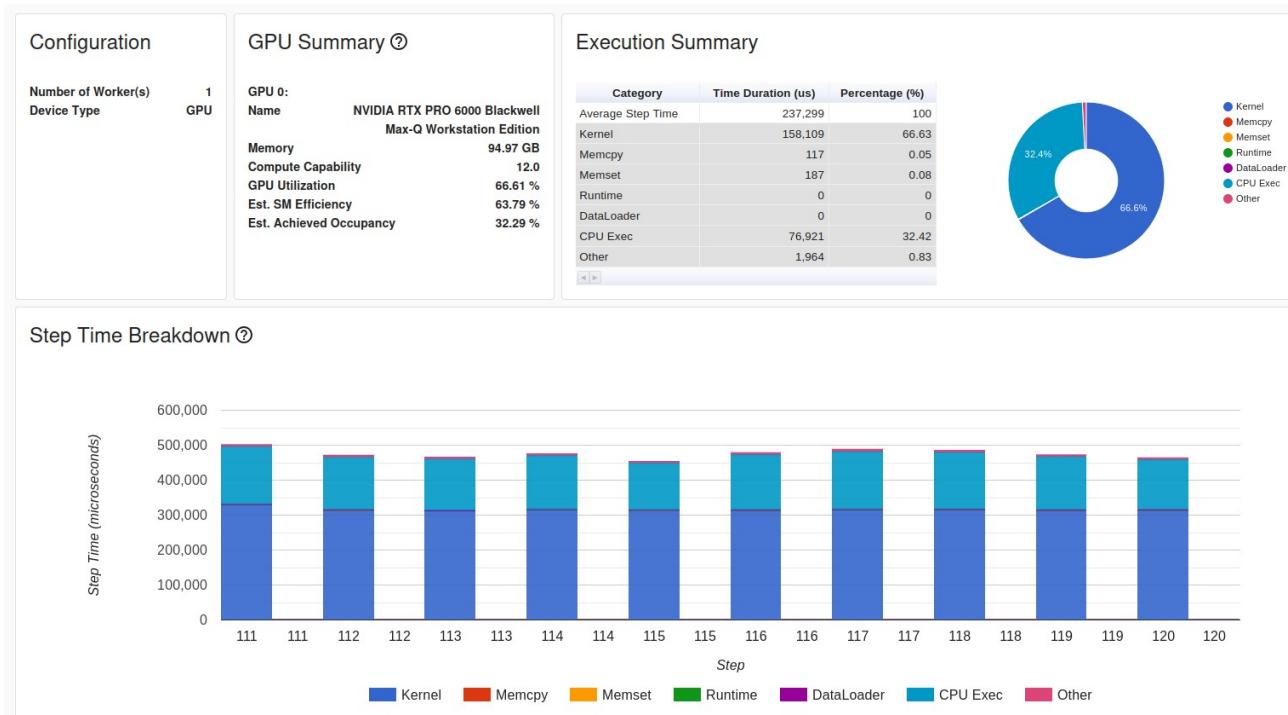
STEP 1: OPTIMIZE SINGLE GPU

- Optimal batch size
- `pin_memory=True`
- Offline data
- CUDA Buffers (showed improvement in multi-GPU)
- `AdamW(fused=True)`
- Fused GEMM
- 44,942 -> 63,627 tok/s (41.57% speedup)
- Time to 10B tokens: 61.80 hours -> **43.67 hours**

STEP 1: OPTIMIZE SINGLE GPU



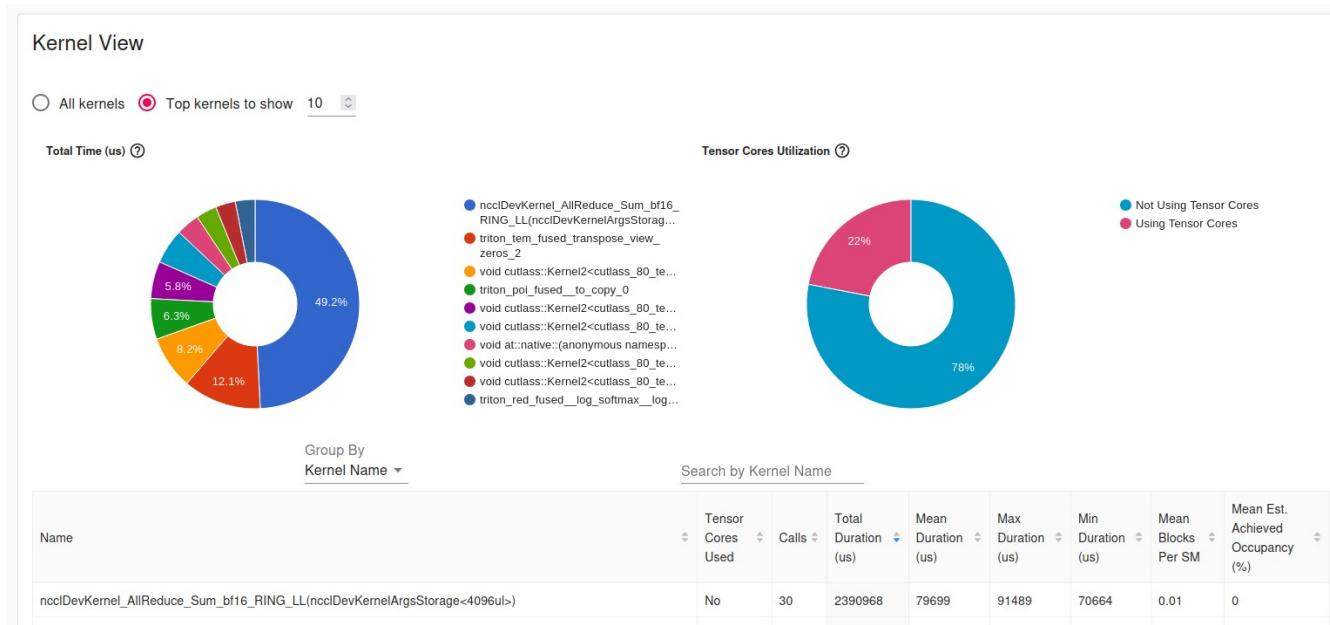
STEP 1: OPTIMIZE SINGLE GPU



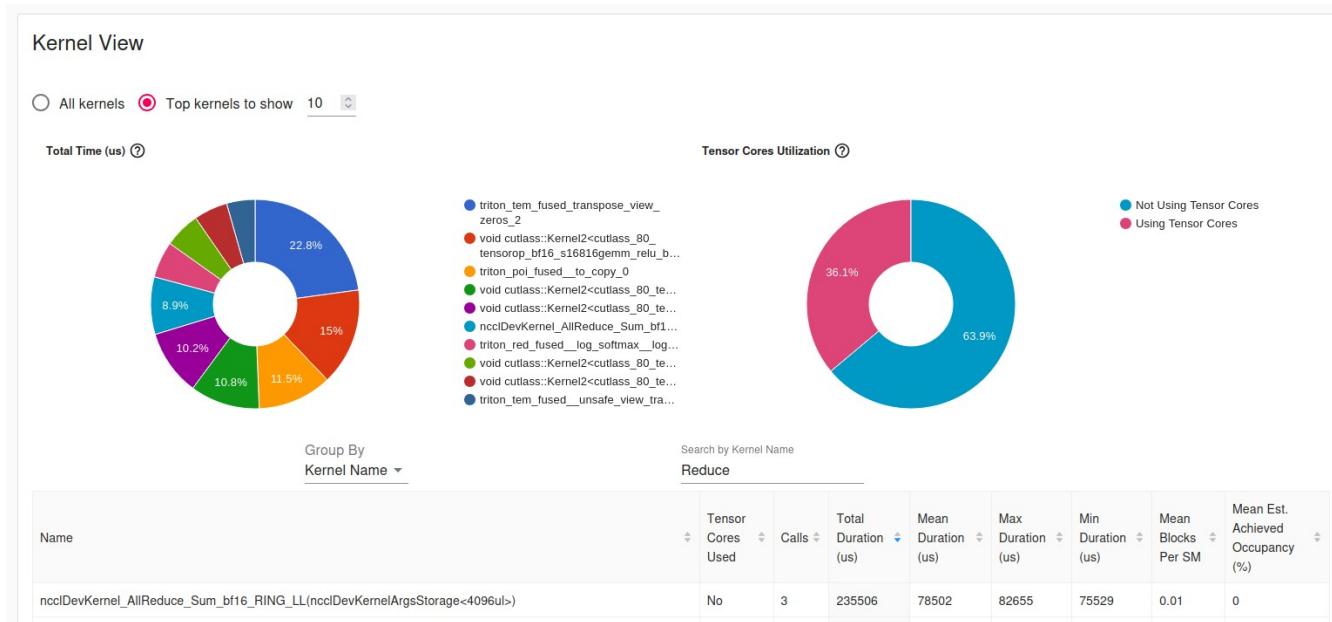
STEP 2: OPTIMIZE MULTI-GPU

- Verify single GPU results align at scale
- Minimize comms as much as possible
- Saturate ~1M tok/batch using gradient accumulation
- 63,627 -> 154,870 -> 210,377 tok/s (230% | 35.85%)
- 43.67 -> 17.94 -> **13.2 hours**

STEP 2: OPTIMIZE MULTI-GPU



STEP 2: OPTIMIZE MULTI-GPU



THANKS FOR COMING!