Department of Artificial Intelligence

CLARIN-PL

Wrocław University of Science and Technology

Large-Scale Machine Learning on Supercomputers: Challenges and Opportunities

Piotr Bielak

Jakub Binkowski

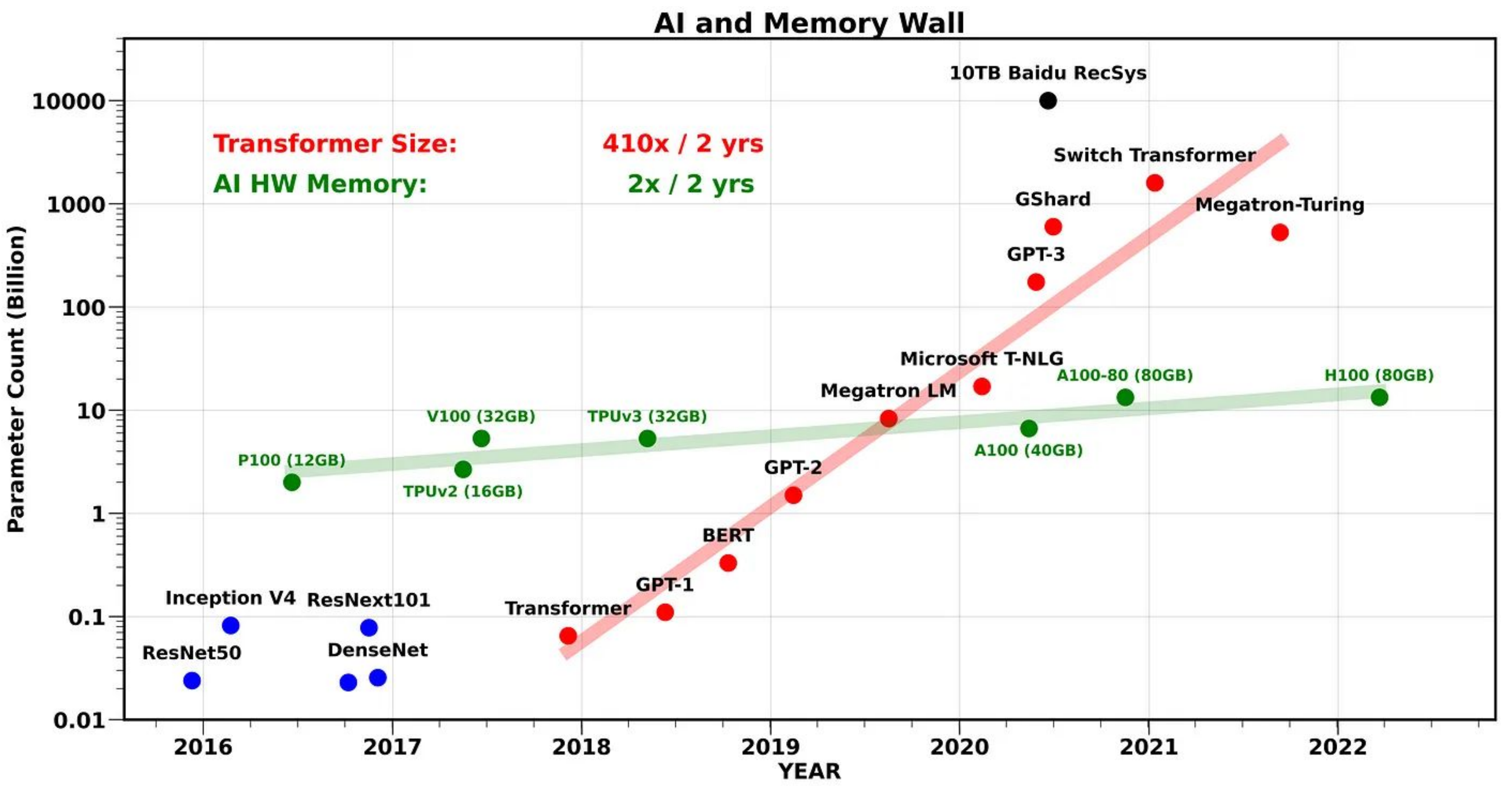
Denis Janiak

Mateusz Gniewkowski

Challenges

Memory Constraints

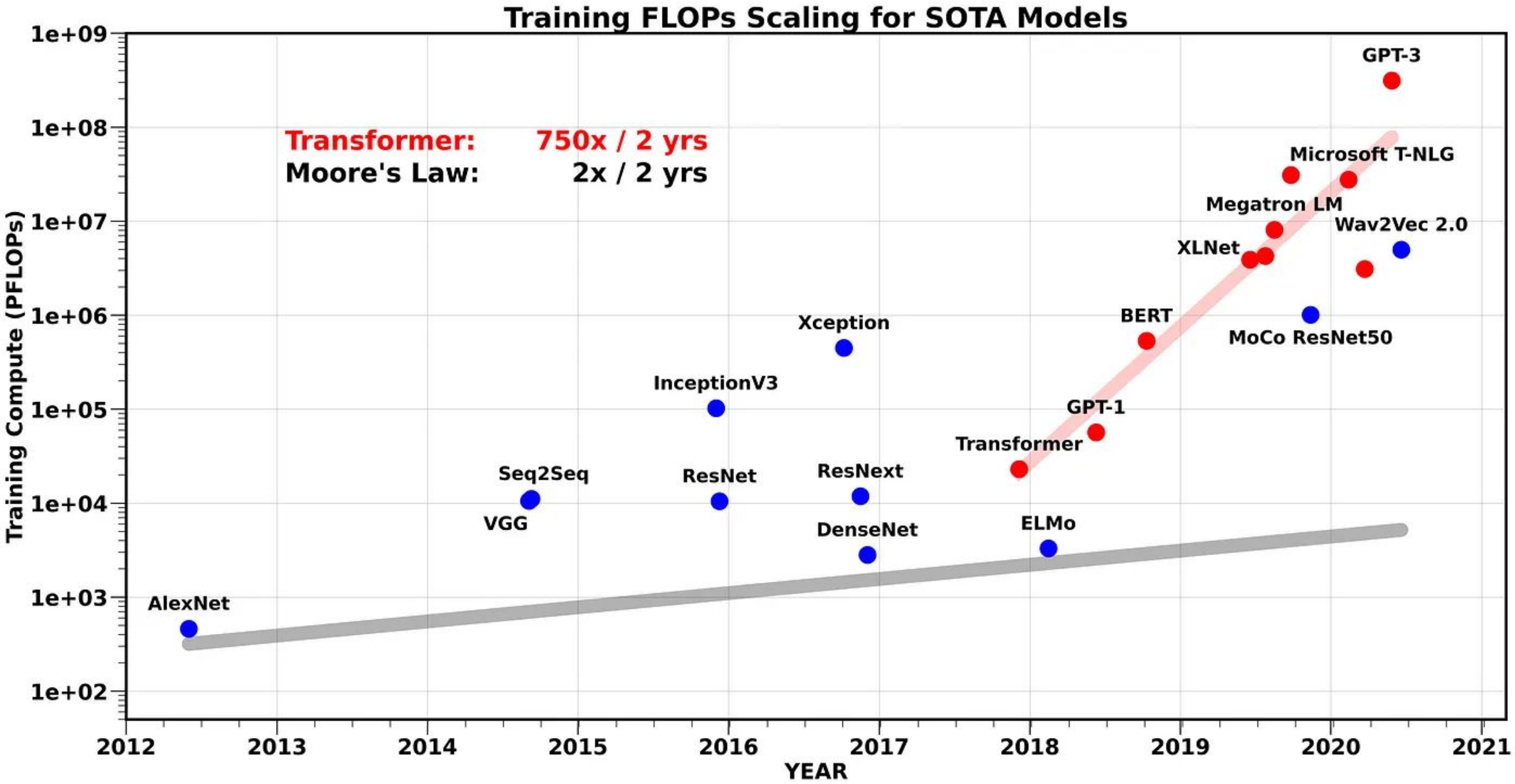
Large language models require massive amounts of memory for both storing the model parameters and processing large batches of data during training. On a single machine, memory constraints can become a bottleneck, especially for models with billions or trillions of parameters.



AI and Memory Wall Gholami, Amir and Yao, Zhewei and Kim, Sehoon and Mahoney, Michael W, and Keutzer, Kurt

Computational Power

Training large language models is computationally intensive, requiring significant processing power for tasks such as gradient computation and optimization. Single machines may not have sufficient computational resources to efficiently train these models within a reasonable timeframe.



NVIDIA A100 delivers 312 TFLOPS, MI250X - 383 TFLOPS

AI and Memory Wall Gholami, Amir and Yao, Zhewei and Kim, Sehoon and Mahoney, Michael W, and Keutzer, Kurt

How to solve the above?

SUPERCOMPUTER!

But there are new challenges when using supercomputer…

Parallelization Efficiency

Supercomputers typically consist of multiple interconnected nodes, each with its own processors and memory. Efficiently parallelizing the training of LLMs across these nodes while minimizing communication overhead and ensuring load balancing is a significant challenge. Scaling training to thousands of nodes without sacrificing efficiency is non-trivial.

Data parallelism - datasets are broken into subsets which are processed in batches on different GPUs using the same model. The results are then combined and averaged in one version of the model.

Distributed data parallelism - enables you to perform data parallelism across GPUs and physical machines and can be combined with model parallelism.

Model parallelism - a single model is broken into segments with each segment run on different GPUs. The results from the segments are then combined to produce a completed model.

Main bottleneck: communication and synchronization overhead

Resource Management

Supercomputers are shared resources used by multiple users and projects simultaneously. Managing resources such as compute nodes, memory, and interconnect bandwidth becomes more complex when training large language models. Allocation policies, job scheduling, and optimizing resource utilization are crucial for maximizing throughput and minimizing wait times for users.

Tldr; memory, storage, queues

Opportunities

Training large-scale language models

OLMo: Accelerating the Science of Language Models

Dirk Groeneveld[α] Iz Beltagy[α]
Pete Walsh[α] Akshita Bhagia[α] Rodney Kinney[α] Oyvind Tafjord[α]
Ananya Harsh Jha[α] Hamish Ivison[αβ] Ian Magnusson[α] Yizhong Wang[αβ]
Shane Arora[α] David Atkinson[α] Russell Authur[α] Khyathi Raghavi Chandu[α]
Arman Cohan[γα] Jennifer Dumas[α] Yanai Elazar[αβ] Yuling Gu[α]
Jack Hessel[α] Tushar Khot[α] William Merrill[δ] Jacob Morrison[α]
Niklas Muennighoff Aakanksha Naik[α] Crys
Valentina Pyatkin[αβ] Abhilasha Ravichander[α] D
Will Smith[α] Emma Strubell[αμ] Nishant Subı
Pradeep Dasigi[α] Nathan Lambert[α]
Luke Zettlemoyer[β] Jesse Dodge[α] Kyl
Noah A. Smith[αβ] Hannaneh
[α]Allen Institute for Artificial
[β]University of Washington [γ]
[δ]New York University [μ]Carnegie
olmo@allenai.org

Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research

Luca Soldaini[♥α] Rodney Kinney[♥α] Akshita Bhagia[♥α] Dustin Schwenk[♥α]
David Atkinson[α] Russell Authur[α] Ben Bogin[αω] Khyathi Chandu[α]
Jennifer Dumas[α] Yanai Elazar[αω] Valentin Hofmann[α] Ananya Harsh Jha[α]
ıxi Lyu[ω] Nathan Lambert[α] Ian Magnusson[α]
ıennighoff Aakanksha Naik[α] Crystal Nam[α]
Ravichander[α] Kyle Richardson[α] Zejiang Shen[τ]
ubramani[χα] Oyvind Tafjord[α] Pete Walsh[α]
ah A. Smith[αω] Hannaneh Hajishirzi[αω]
irk Groeneveld[α] Jesse Dodge[α]
Kyle Lo[♥α]
of California, Berkeley [χ]Carnegie Mellon University
titute of Technology [ω]University of Washington
{lucas,kylel}@allenai.org

FinGPT: Large Generative Models for a Small Language

Risto Luukkonen[†*] Ville Komulainen[†] Jouni Luoma[†] Anni Eskelinen[†]
Jenna Kanerva[†] Hanna-Mari Kupari[†] Filip Ginter[†] Veronika Laippala[†]
Niklas Muennighoff[‡] Aleksandra Piktus[‡] Thomas Wang[‡] Nouamane Tazi[‡]
Teven Le Scao[‡] Thomas Wolf[‡] Osma Suominen[◇] Samuli Sairanen[◇]
Mikko Merioksa[◇] Jyrki Heinonen[◇] Aija Vahtola[◇] Samuel Antao[○]
Sampo Pyysalo[†*]
[†] TurkuNLP Group, University of Turku [‡] Hugging Face
[◇] National Library of Finland [○] AMD
*risto.m.luukkonen@utu.fi, sampo.pyysalo@utu.fi

Benchmarking and new methods

CrIBo: Self-Supervised Learning via Cross-Image Object-Level Bootstrapping

Tim Lebailly[1*] Thomas Stegmüller[2*] Behzad Bozorgtabar[2,3]
Jean-Philippe Thiran[2,3] Tinne Tuytelaars[1]
[1]KU Leuven [2]EPFL [3]CHUV
[1]{firstname}.{lastname}@esat.kuleuven.be [2]{firstname}.{lastname}@epfl.ch

Monolingual or Multilingual Instruction Tuning: Which Makes a Better Alpaca

Pinzhen Chen[1,*] Shaoxiong Ji[2,*] Nikolay Bogoychev[1]
Andrey Kutuzov[3] Barry Haddow[1] Kenneth Heafield[1]
rsity of Helsinki [3]University of Oslo
haoxiong.ji@helsinki.fi

Energy Concerns with

Roblex Nana
Mines Paris - PSL,
Centre de Recherche en Informatique (CRI)
Fontainebleau, France
roblex.nana_tchakoute@minesparis.psl.eu

Petr Dokladal
Mines Paris - PSL,
Centre de Morphologie Mathématique (CMM
Fontainebeau, France
petr.dokladal@minesparis.psl.eu

Paloma: A Benchmark for Evaluating Language Model Fit

Ian Magnusson♠ Akshita Bhagia♠ Valentin Hofmann♠ Luca Soldaini♠
Ananya Harsh Jha♠ Oyvind Tafjord♠ Dustin Schwenk♠ Evan Pete Walsh♠
Yanai Elazar♠◇ Kyle Lo♠ Dirk Groeneveld♠ Iz Beltagy♠ Hannaneh Hajishirzi♠◇
Noah A. Smith♠◇ Kyle Richardson♠ Jesse Dodge♠
♠Allen Institute for Artificial Intelligence
◇Paul G. Allen School of Computer Science & Engineering, University of Washington
{ianm, jessed}@allenai.org

youssef.mesri@minesparis.psl.eu

Large-scale computations vs. sustainability

Energy Concerns with HPC Systems and Applications

Roblex Nana
Mines Paris - PSL,
Centre de Recherche en Informatique (CRI)
Fontainebleau, France
roblex.nana_tchakoute@minesparis.psl.eu

Claude Tadonki
Mines Paris - PSL,
Centre de Recherche en Informatique (CRI)
Fontainebleau, France
claude.tadonki@minesparis.psl.eu

Petr Dokladal
Mines Paris - PSL,
Centre de Morphologie Mathématique (CMM)
Fontainebeau, France
petr.dokladal@minesparis.psl.eu

Youssef Mesri
Mines Paris - PSL,
Centre de Mise en Forme de Matériaux (CEMEF)
Sophia Antipolis, France
youssef.mesri@minesparis.psl.eu

Wayback Machine
(Hackathon LUMI and our experiences)

Benchmark of LLMs for Polish language: NVIDIA vs AMD https://github.com/CLARIN-PL/klajster

1. Building an image

```
#!/usr/bin/bash
echo -n "username:"
read USERNAME

BASEIMAGE_NAME=lumi-pytorch-rocm-5.5.1-python-3.10-pytorch-v2.0.1.sif

if [ ! -f $BASEIMAGE_NAME ]
then
scp -C $USERNAME@lumi.csc.fi:/appl/local/containers/sif-images/$BASEIMAGE_NAME .
fi

singularity build --fakeroot klajster.sif klajster.def
scp -C klajster.sif $USERNAME@lumi.csc.fi:/project/project_465000858/
```

Remarks:

1. There should be possibility to build images using LUMI server
2. It is probably possible to create a job for a cluster to do exactly that

2. Running jobs

1. Running jobs is handled via **DVC (Data Version Control)**
2. The code on the right generates X jobs with different configurations
3. Most of the magic behind multiple nodes and multiple GPUs is handled using **PyTorch-Lightning**

```yaml
run_amd_benchmark:
  matrix:
    dataset: [ "polemo2" ]
    embedding: [
      "allegro__herbert-base-cased",
    ]
    device_cfg: [
      {"num_nodes": 1, "num_gpus": 1},
      {"num_nodes": 1, "num_gpus": 2},
      {"num_nodes": 1, "num_gpus": 4},
      {"num_nodes": 1, "num_gpus": 8},
      {"num_nodes": 2, "num_gpus": 8},
    ]
  cmd: >-
    MYMASKS="0xfe000000000000,0xfe00000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000";
    export SINGULARITY_BIND=;
    srun
    -t 24:00:00
    -N ${item.device_cfg.num_nodes}
    --account=project_465000858
    --partition=standard-g
    --cpu-bind=mask_cpu:$MYMASKS
    --ntasks-per-node ${item.device_cfg.num_gpus}
    --gpus $((${item.device_cfg.num_gpus}*${item.device_cfg.num_nodes}))
    singularity exec
    -B $(pwd):/myrun
    /project/project_465000858/klajster.sif
    bash -c "
    \$WITH_CONDA;
    cd /myrun;
    source setup_envs.sh;
    PYTHONPATH=. python experiments/scripts/evaluate_lightning_classification.py
    --ds ${item.dataset}
    --embedding-path ${item.embedding}
    --pipeline-params-path experiments/configs/eval/${item.dataset}/${item.embedding}.yaml
    --output-path data/benchmark/amd_${item.embedding}_${item.dataset}_num-nodes-${item.device_cfg.num_nodes}_num-gpus-${item.device_cfg.num_gpus}
    --num-nodes ${item.device_cfg.num_nodes}
    --devices ${item.device_cfg.num_gpus}
    --gpu-type amd
    --accelerator gpu
    --retrains 1
    "
```

2. Running jobs

```
run_amd_benchmark:
  matrix:
    dataset: [ "polemo2" ]
    embedding: [
      "allegro__herbert-base-cased",
    ]
    device_cfg: [
      {"num_nodes": 1, "num_gpus": 1},
      {"num_nodes": 1, "num_gpus": 2},
      {"num_nodes": 1, "num_gpus": 4},
      {"num_nodes": 1, "num_gpus": 8},
      {"num_nodes": 2, "num_gpus": 8},
    ]
  cmd: >-
    MYMASKS="0xfe000000000000,0xfe00000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000";
    export SINGULARITY_BIND=;
    srun
    -t 24:00:00
    -N ${item.device_cfg.num_nodes}
    --account=project_465000858
    --partition=standard-g
    --cpu-bind=mask_cpu:$MYMASKS
    --ntasks-per-node ${item.device_cfg.num_gpus}
    --gpus $((${item.device_cfg.num_gpus}*${item.device_cfg.num_nodes}))
    singularity exec
    -B $(pwd):/myrun
    /project/project_465000858/klajster.sif
    bash -c "
    \$WITH_CONDA;
    cd /myrun;
    source setup_envs.sh;
    PYTHONPATH=. python experiments/scripts/evaluate_lightning_classification.py
    --ds ${item.dataset}
    --embedding-path ${item.embedding}
    --pipeline-params-path experiments/configs/eval/${item.dataset}/${item.embedding}.yaml
    --output-path data/benchmark/amd_${item.embedding}_${item.dataset}_num-nodes-${item.device_cfg.num_nodes}_num-gpus-${item.device_cfg.num_gpus}
    --num-nodes ${item.device_cfg.num_nodes}
    --devices ${item.device_cfg.num_gpus}
    --gpu-type amd
    --accelerator gpu
    --retrains 1
    "
```

Remarks – Cool, but several issues:

1. Using python3.9 from a container causes SINGULARITY_BIND variable to be set inside any job called from python (therefore “export SINGULARITY_BIND=;”)

The problem does not occur with natively installed Python3.6, but its version is too old ;(

2. Running jobs

```
run_amd_benchmark:
  matrix:
    dataset: [ "polemo2" ]
    embedding: [
      "allegro__herbert-base-cased",
    ]
    device_cfg: [
      {"num_nodes": 1, "num_gpus": 1},
      {"num_nodes": 1, "num_gpus": 2},
      {"num_nodes": 1, "num_gpus": 4},
      {"num_nodes": 1, "num_gpus": 8},
      {"num_nodes": 2, "num_gpus": 8},
    ]
  cmd: >-
    MYMASKS="0xfe000000000000,0xfe00000000000000,0xfe0000,0xfe000000,0xfe,0xfe00,0xfe00000000,0xfe0000000000";
    export SINGULARITY_BIND=;
    srun
    -t 24:00:00
    -N ${item.device_cfg.num_nodes}
    --account=project_465000858
    --partition=standard-g
    --cpu-bind=mask_cpu:$MYMASKS
    --ntasks-per-node ${item.device_cfg.num_gpus}
    --gpus $((${item.device_cfg.num_gpus}*${item.device_cfg.num_nodes}))
    singularity exec
    -B $(pwd):/myrun
    /project/project_465000858/klajster.sif
    bash -c "
    \$WITH_CONDA;
    cd /myrun;
    source setup_envs.sh;
    PYTHONPATH=. python experiments/scripts/evaluate_lightning_classification.py
    --ds ${item.dataset}
    --embedding-path ${item.embedding}
    --pipeline-params-path experiments/configs/eval/${item.dataset}/${item.embedding}.yaml
    --output-path data/benchmark/amd_${item.embedding}_${item.dataset}_num-nodes-${item.device_cfg.num_nodes}_num-gpus-${item.device_cfg.num_gpus}
    --num-nodes ${item.device_cfg.num_nodes}
    --devices ${item.device_cfg.num_gpus}
    --gpu-type amd
    --accelerator gpu
    --retrains 1
    "
```

2. There are few problems when using slower storage with PyTorch (even after setting MIOPEN_USER_DB_PATH and MIOPEN_CUSTOM_CACHE_DIR) – dataloaders often time-out. Using SSD drives solved the issue.

We run the code from the `/flash/project_465000858/` directory!

Let’s sum up our work

Which of your goals did you accomplish?

- Add RetNet to LEPISZCZE
  - RetNet with 300M parameters trained
  - No full integration with LEPISZCZE
- Check model scalability by parallelization on a different number of GPUs:
  - Run the benchmark on NVIDIA
  - Run the benchmark on AMD
- Analyze the results
- Profile the models to identify bottlenecks
  - Basic profiling
  - In-depth investigation

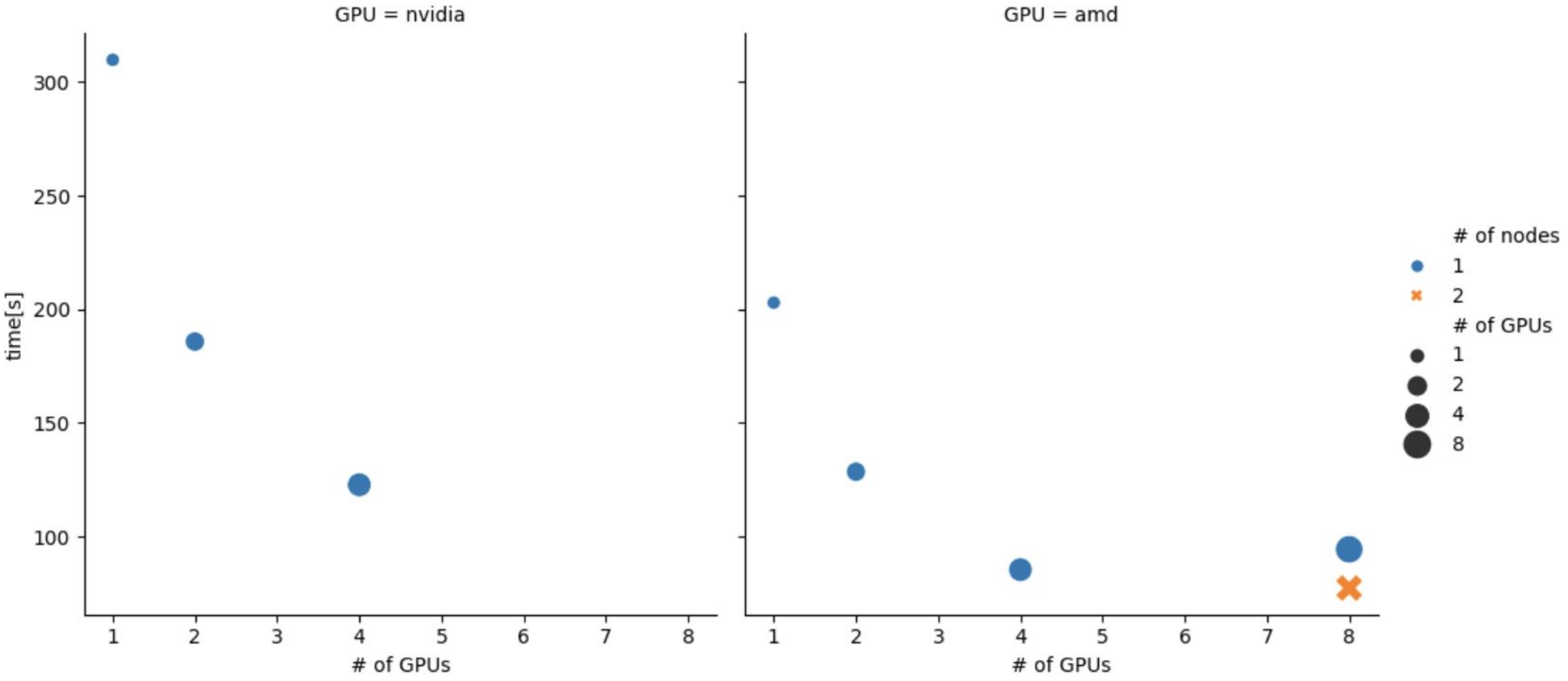
What is left to do?

1. Exploration of grid search for hyperparameters to refine public-facing models.
2. Using a larger dataset for RetNet.
   a. PoC 1% of the **oscar** dataset
   b. 1 % = 200,000 texts
   c. https://huggingface.co/datasets/oscar/viewer/unshuffled_original_pl
3. In-depth review of profiling results
   a. Try to eliminate model bottlenecks

What was the most important change implemented during that week?

- Implementation of **multi-node multi-gpu** training in the **embeddings** library
  - LEPISZCZE □ embeddings □ PyTorch-Lightning □ PyTorch
  - PyTorch-Lightning should support such training out-of-the-box…
  - …but the pipeline implementation in “embeddings” was not ready :(
  - Our change allows to scale the LEPISZCZE benchmark to utilize larger compute clusters!
- **Training RetNet**
  - 128 AMD GPUs
  - First time trained using only Polish corpora
  - RetNet allows for O(1) inference
  - Further downstream evaluation of the trained model should reveal its potential for Polish NLP!

(How) did your performance improve?

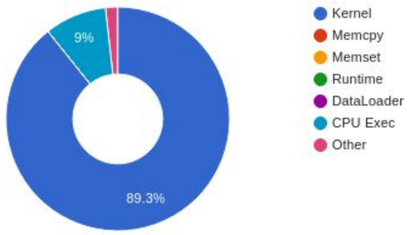


Configuration

Number of Worker(s)	**3**
Device Type	**GPU**

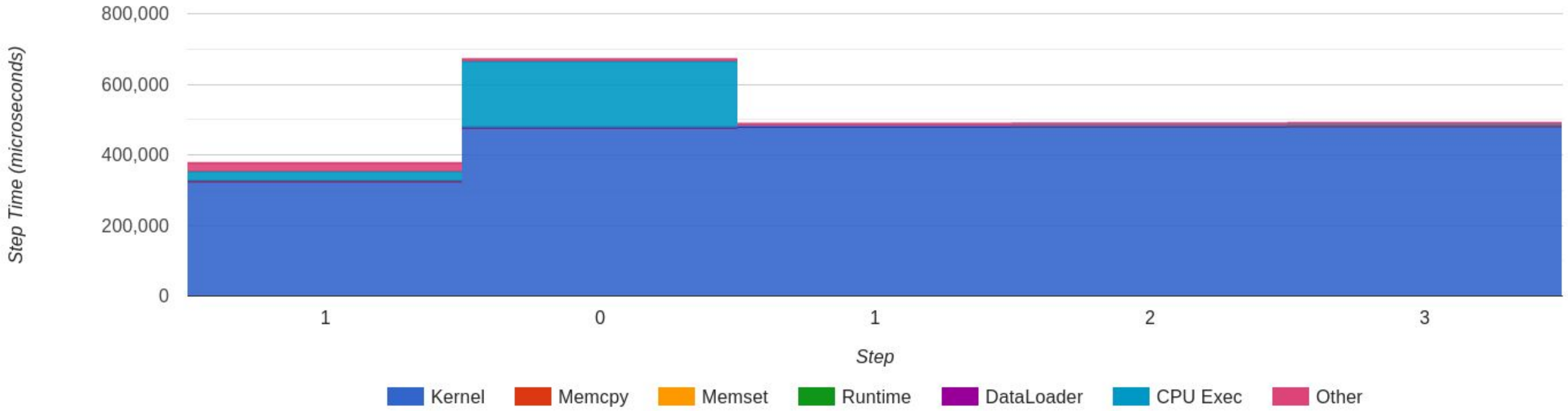
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	502,881	100
Kernel	448,876	89.26
Memcpy	0	0
Memset	0	0
Runtime	0	0
DataLoader	0	0
CPU Exec	45,121	8.97
Other	8,884	1.77



Step Time Breakdown

Training AMD (1 node, 1 gpu)



Configuration

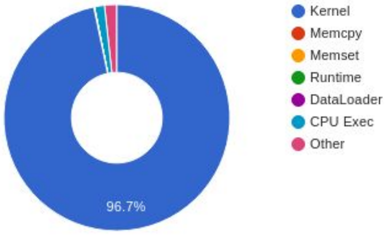
Number of Worker(s)	**3**
Device Type	**GPU**

GPU Summary

GPU 0:	
Name	**NVIDIA A40**
Memory	**44.35 GB**
Compute Capability	**8.6**
GPU Utilization	**96.72 %**
Est. SM Efficiency	**96.44 %**
Est. Achieved Occupancy	**54.4 %**

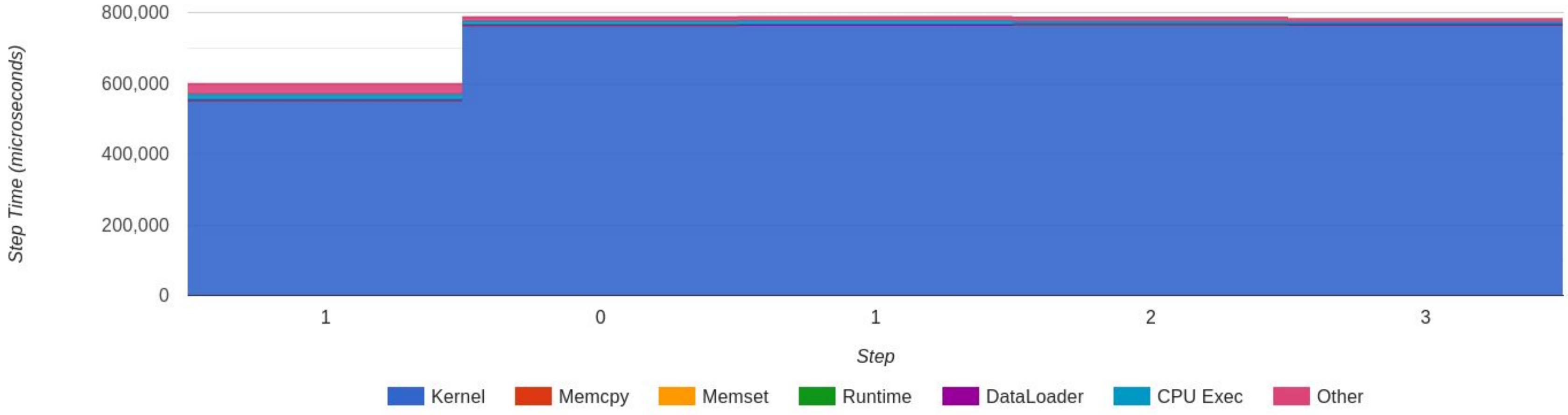
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	747,560	100
Kernel	723,056	96.72
Memcpy	415	0.06
Memset	205	0.03
Runtime	0	0
DataLoader	0	0
CPU Exec	10,749	1.44
Other	13,136	1.76



Step Time Breakdown

Training NVIDIA (1 node, 1 gpu)

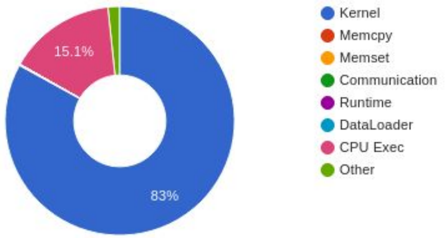


Configuration

Number of Worker(s)	**12**
Device Type	**GPU**

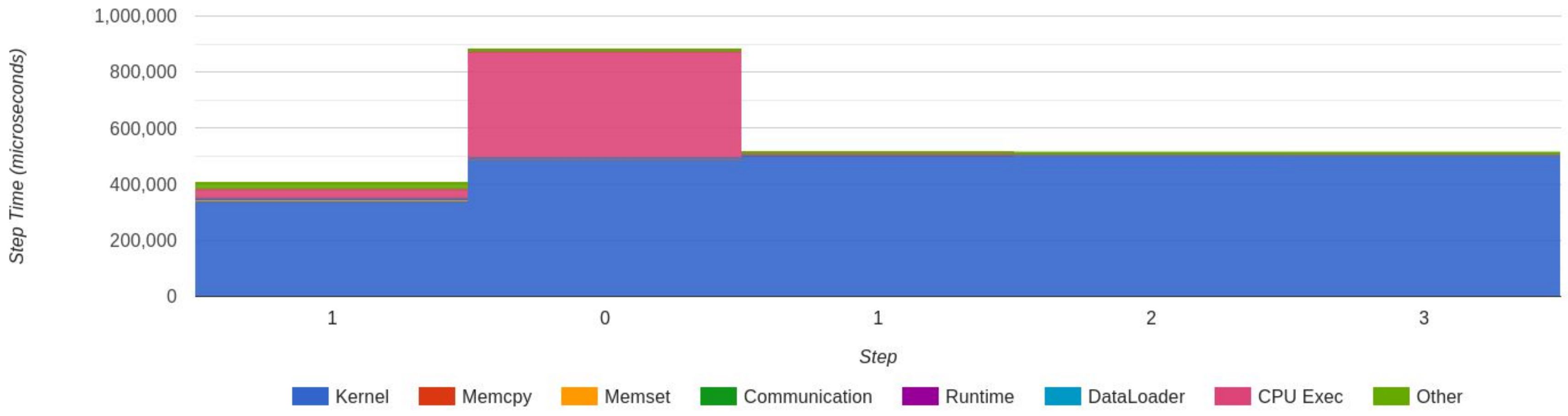
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	564,770	100
Kernel	468,972	83.04
Memcpy	0	0
Memset	0	0
Communication	1,046	0.19
Runtime	0	0
DataLoader	0	0
CPU Exec	85,340	15.11
Other	9,412	1.67



Step Time Breakdown

Training AMD (1 node, 4 gpu)



Configuration

Number of Worker(s)	12
Device Type	GPU

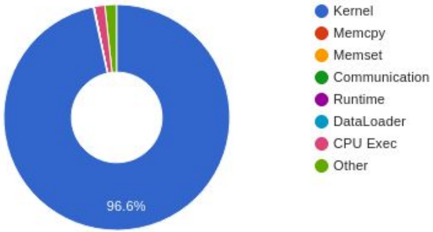
GPU Summary

GPU 0:

Name	NVIDIA A40
Memory	44.35 GB
Compute Capability	8.6
GPU Utilization	96.62 %
Est. SM Efficiency	94.49 %
Est. Achieved Occupancy	52.23 %

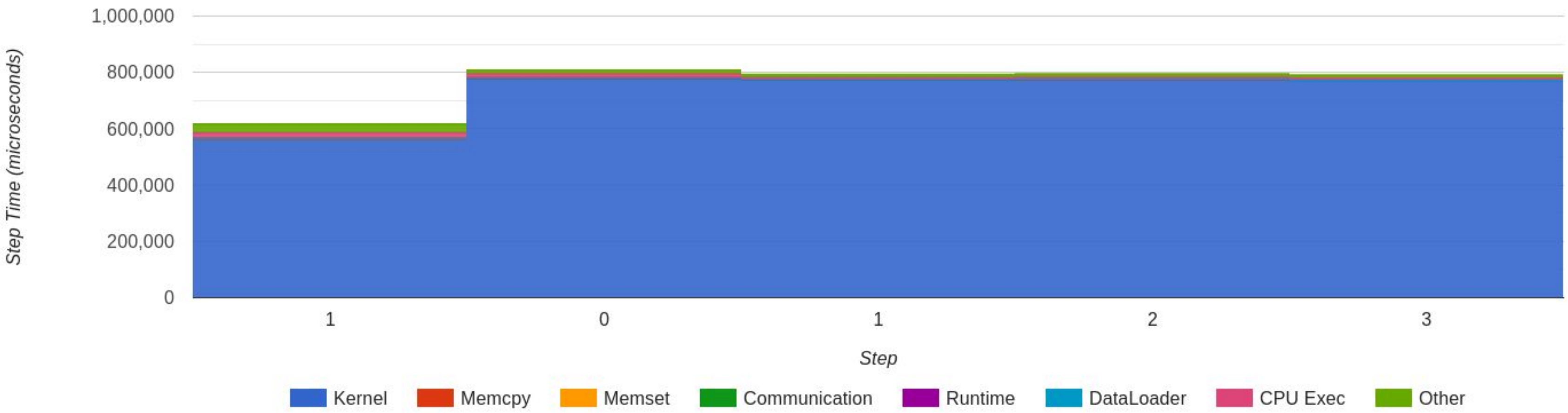
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	759,516	100
Kernel	733,888	96.63
Memcpy	436	0.06
Memset	201	0.03
Communication	365	0.05
Runtime	0	0
DataLoader	0	0
CPU Exec	11,786	1.55
Other	12,840	1.69



Step Time Breakdown

Training NVIDIA (1 node, 4 gpu)

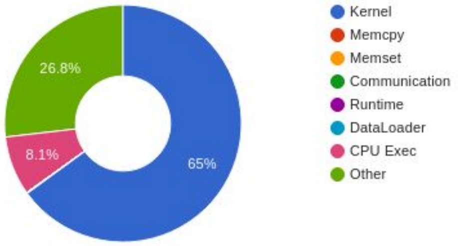


Configuration

Number of Worker(s)	**16**
Device Type	**GPU**

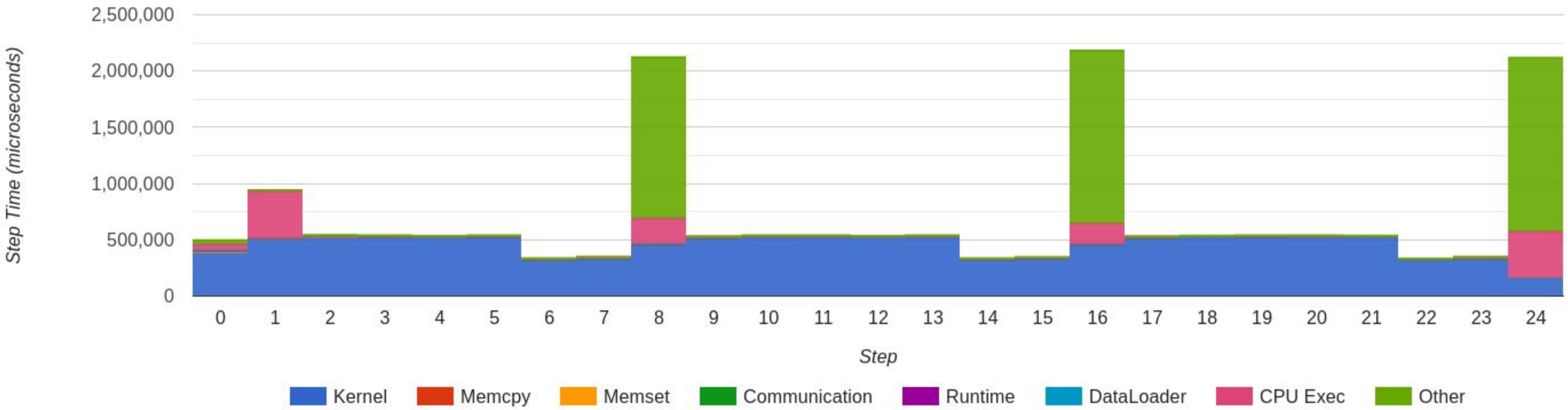
Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	696,563	100
Kernel	452,455	64.96
Memcpy	0	0
Memset	0	0
Communication	978	0.14
Runtime	0	0
DataLoader	0	0
CPU Exec	56,189	8.07
Other	186,940	26.84



Step Time Breakdown ?

Training AMD (2 node, 16 gpu)

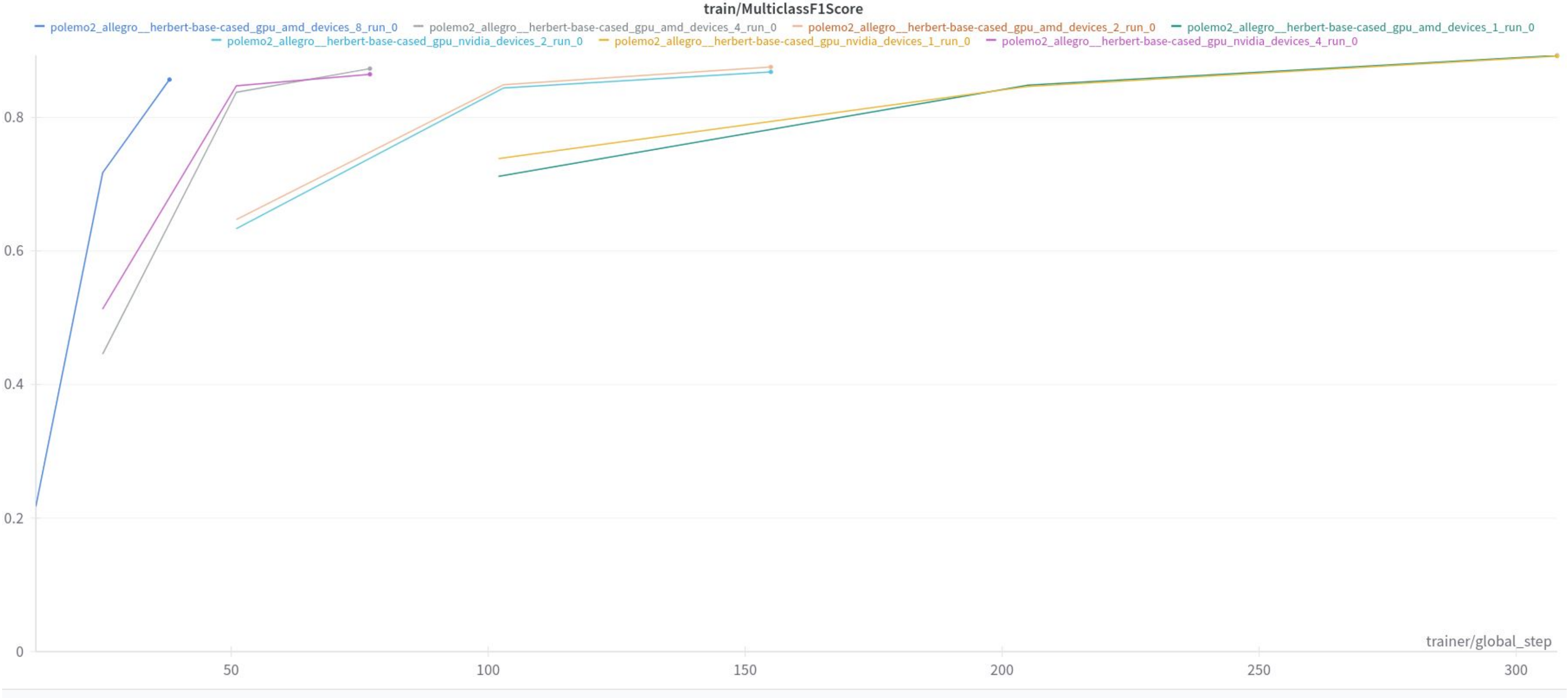


Comparison of number of lower level calls is possible

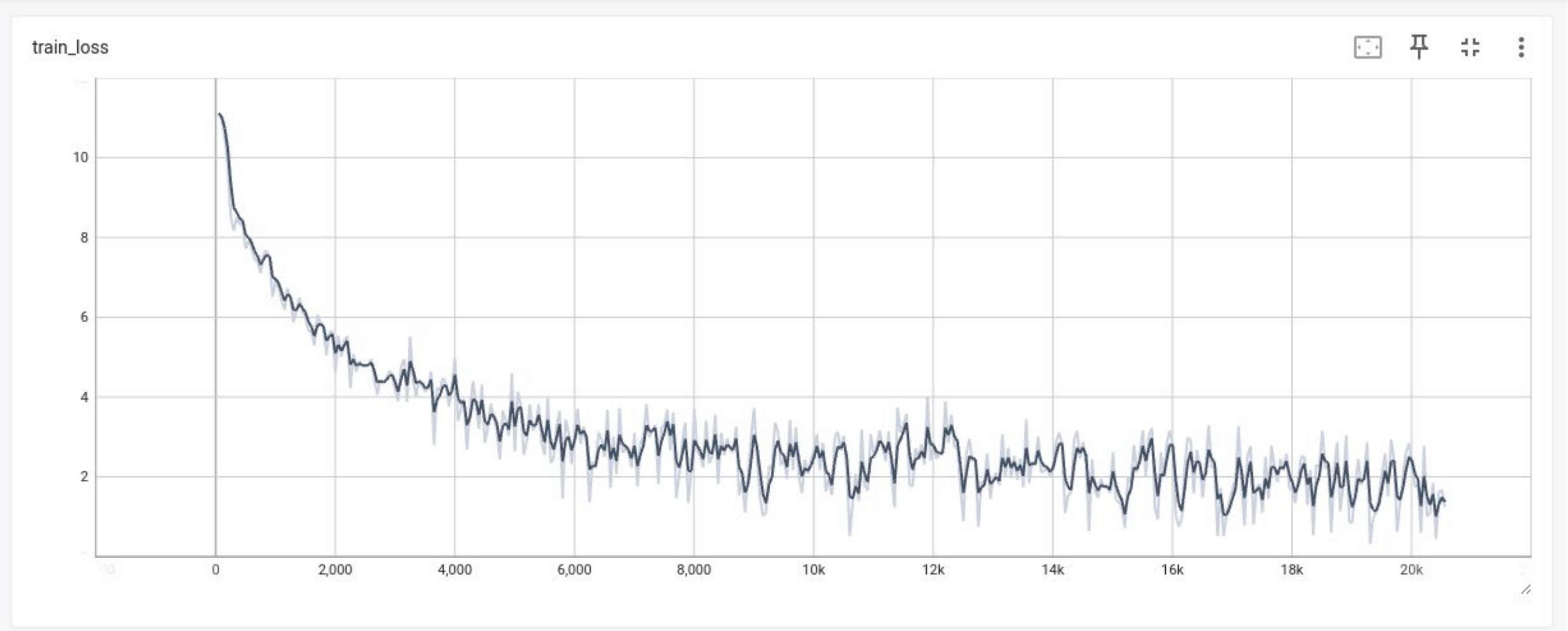
Host Duration ×

Operator	Baseline Calls	Exp Calls	Delta Calls	Delta Calls%	Baseline Host Duration (us)	Exp Host Duration (us)	Delta Host Duration (us)	Delta Host Duration%
aten::as_strided	2786	3147	361	12.96%	1901	1170	-731	-38.45%
aten::empty_strided	1517	1517	0	0.00%	42825	3727	-39098	-91.30%
aten::result_type	1402	1534	132	9.42%	213	70	-143	-67.14%
aten::empty	1293	934	-359	-27.76%	50375	3124	-47251	-93.80%

As expected, result quality is the same



RetNet – learning progress



Thank you for your attention!