Performance and Power: Systematic Evaluation of AI Workloads on Accelerators with CARAML

Chelsea Maria John, Stepan Nassyr, Carolin Penke, Andreas Herten
Jülich Supercomputing Centre
Forschungszentrum Jülich
Jülich, Germany

***Abstract*—The rapid advancement of machine learning (ML) technologies has driven the development of specialized hardware accelerators designed to facilitate more efficient model training. This paper introduces the CARAML benchmark suite, which is employed to assess performance and energy consumption during the training of transformer-based large language models and computer vision models on a range of hardware accelerators, including systems from NVIDIA, AMD, and Graphcore. CARAML provides a compact, automated, extensible, and reproducible framework for assessing the performance and energy of ML workloads across various novel hardware architectures. The design and implementation of CARAML, along with a custom power measurement tool called jpwr, are discussed in detail.**

***Index Terms*—Machine Learning, Energy, NLP, Computer Vision, AI, Performance Measurement, Benchmark, GPU, IPU, Accelerators**

I. Introduction

Fueled by the growing interest in training ever larger deep neural networks, such as large language models and other foundation models, the demands for hardware specialized on these workloads have grown immensely. Graphics processing units (GPUs) have evolved from their origins in computer graphics to become the primary computational engines of the AI revolution. While the central processing unit (CPU) controls a program's execution flow, it offloads compute-intensive highly-parallel tasks to the GPU (the *accelerator*). Evolving from a pioneering company, NVIDIA has emerged as the dominant player in the market as of 2024, spearheading current hardware developments. Other vendors, such as AMD and Intel, also provide GPUs aiming to accelerate model training and inference.

Another promising class of AI accelerators is based on the idea of distributed local per-compute-unit memory together with on-chip message passing, in contrast to a shared memory hierarchy, typical to classical CPUs and GPUs. Vendors following this *dataflow* approach include Graphcore, Cerebras, Groq, SambaNova, and Tenstorrent.

Performance characteristics not only vary between generations and vendors, but depend on the node or cluster configuration in which the accelerator is embedded, including CPU, memory, and interconnect. When evaluating and comparing these heterogeneous hardware options, e.g. for purchase decisions in an academic or industrial setting, it is not sufficient to compare hardware characteristics such as number of cores, thermal design power (TDP), theoretic bandwidth, or peak performance in FLOP/s. Their effect on workload performance is not straightforward, and the accelerator architectures might barely be comparable. Performance data reflecting the actual intended workloads, collected on various competing systems independently of vendor interests, offer highly valuable information. Power consumption is one such important metric in this regard.

In particular within the field of machine learning, having a structured, automatic benchmarking tool to investigate the effect of hyperparameters, such as learning rate and batch size, and to identify optimal settings is important. When training large models across multiple cluster nodes, additional hyperparameters are necessary to define the parallelization layout, leveraging various forms of parallelism. Moreover, hardware configurations, such as those related to processor affinity or network communication, must be systematically explored.

As a framework to collect this data, we present the CARAML benchmark suite, a **C**ompact, **A**utomated, **R**eproducible **A**ssessment of **M**achine-**L**earning workloads on novel accelerators. Further, we contribute performance data on various accelerators, collected with CARAML, including energy measurements, as well as first-hand experiences with encountered challenges. The machine learning workloads include the training of a Generative Pretrained Transformer (GPT)-based large language model using `PyTorch`, as well as training of a `ResNet50` model implemented in `TensorFlow`. Energy measurements are facilitated by our jpwr tool, which we also contribute with this paper.

The remaining paper is structured as follows: in section II we provide background information on the state-of-the-art AI workloads included in CARAML in the fields of natural language processing and image recognition. Details on the investigated hardware configurations. More details on the implementation of the benchmark, including a description of used tools, requirements, and execution instructions, are presented in section III. Performance results, in terms of throughput (images/s, tokens/s) for various batch sizes, are reported in section IV. Additional energy metrics such as images/Wh and tokens/Wh provide another layer of insight. In section V, we briefly discuss the difficulties encountered in the benchmark development in terms of hardware and software compatibility and comparability, and how they are resolved. Final conclusions, take-away messages, and future plans are given in section VI.

II. Background

Deep learning constitutes an important workload of high performance computing (HPC) clusters, only increasing in significance in recent years. A neural network aims to generalize an input-output relation observed in its training data. A chain of matrix products and activation functions, reflecting a neural network architecture, is applied to the batched input data in a forward pass. The backward pass, also called back-propagation, updates the matrix elements (the "weights") using further matrix products, This operation reflects a step of (stochastic) gradient descent to

minimize the loss function representing the difference between computed and true output of the sample.

Although specialized network architectures like transformers and convolutional neural networks have been developed for specific tasks, they rely on the fundamental building block of matrix multiplication. Since matrix multiplications are inherently parallel, many-core hardware architectures, such as GPUs and other accelerators optimized for this process, are crucial.

A. Natural Language Processing

The advent of large language models based on transformer architectures [1] has revolutionized the field of natural language processing (NLP). These models have enabled significant advancements across a wide range of tasks, including speech recognition, text classification, natural language understanding, and generation. By leveraging shared foundational models, these tasks can now be effectively addressed with additional fine-tuning or in-context learning, allowing for greater flexibility and efficiency in various domains.

Language models use large text corpora as training data to predict the next piece of text (a *token*) given the preceding context. The original transformer architecture consists of an encoder and a decoder connected via a cross-attention mechanism. Attention is a key operation in this architecture, characterized by its quadratic complexity in the sequence length. It involves matrix-matrix products of learned token representations, allowing the model to capture relationships between tokens while accounting for their relative positions.

Scaling up transformer models is achieved by stacking multiple transformer layers, each built around an attention mechanism, along with feed-forward layers, residual connections, and normalization. This increases the number of learnable parameters, making larger networks more capable, especially when trained on sufficiently large datasets. To handle the computational demands of such large models, parallelization techniques are essential. State-of-the-art methods include 3D parallelism [2], [3] (combining data, tensor, and pipeline parallelism), sequence parallelism [4], activation recomputation [4], and optimizations like flash attention [5].

The NLP benchmark task in CARAML utilizes the Megatron-LM framework [2], [6], a robust, research-focused software platform developed by NVIDIA using `PyTorch`. Megatron-LM has been instrumental in advancing large-scale language model training, incorporating and pioneering the previously mentioned features. The BigCode Project fork of Megatron-LM with ROCm adaptations is used for AMD and forked version of Graphcore application examples is used on Graphcore.

B. Computer Vision

Computer vision (CV), particularly its core task of image classification, has been a significant driver of deep learning advancements due to its numerous real-world applications. The architecture of convolutional neural networks (CNNs) effectively addresses challenges such as vanishing and exploding gradients by employing successive layers of convolutional filters, defined by learned parameters. It became evident that the massively parallel architecture of GPUs is well-suited for the computational demands of CNNs, leading to substantial improvements in training efficiency and model performance [7].

Residual connections were introduced by the ResNet model family [8] and solved the problem of *degradation*, i.e. the counterintuitive observation of higher training errors in deeper networks with more parameters.

- **NVIDIA A100 GPU (SXM4)**:
  - 108 SM (each: 64 CUDA cores, 4 Tensor Cores)
  - Peak performance FP16: 312 TFLOP/s
  - Memory: 40 GB HBM2e
- **NVIDIA H100 GPU (PCIe)** :
  - 114 SM (each: 128 CUDA cores, 4 Tensor Cores)
  - Peak performance FP16: 756 TFLOP/s
  - Memory: 80 GB HBM2e
- **NVIDIA H100 GPU (SXM5)**
  - 132 SM (each: 128 CUDA cores, 4 Tensor Cores)
  - Peak performance FP16: 990 TFLOP/s
  - Memory: 94 GB HBM2e
- **NVIDIA GH200 Superchip**:
  - CPU: NVIDIA Grace (Arm Neoverse-V2), 72 cores
  - GPU: NVIDIA Hopper H100, 132 SM (each: 128 CUDA cores, 4 Tensor Cores)
  - Peak performance FP16 : 990 TFLOP/s
  - Memory: 96 GB HBM3 (GPU) at 4 TB/s, up to 480 GB LPDDR5X (CPU) at up to 512 GB/s (8532 MHz)
- **AMD MI250 GPU**:
  - GPU built from two Graphics Compute Dies (GCD) as Multi-Chip Module (MCM). The numbers below refer to one MCM, i.e. two logical GPUs as seen by the operating system.
  - 2× 104 CU (each: 64 stream processors, 4 Matrix Cores)
  - Peak performance FP16: 362.1 TFLOP/s
  - Memory: 128 GB HBM2e
- **Graphcore GC200 IPU** :
  - 1472 processor cores (*IPU cores*)
  - Peak performance FP16: 250 TFLOP/s
  - Memory: 900 MB, distributed to cores

Fig. 1. List of evaluated accelerators. NVIDIA's *Streaming Multiprocessors* are abbreviated with *SM*. AMD's *Compute Units* are abbreviated with *CU*. Peak performance is given without sparsity.

While the transformer architecture has also found its way into image recognition [9], convolutional neural networks are extremely mature and widely used in production environments, and represent an important benchmark case. Together with transformers, they cover a wide portion of currently relevant deep learning paradigms.

In CARAML, a benchmark to train a `ResNet50` model, i.e. a ResNet model with 50 convolutional layers from scratch, is curated from a forked version of the official TensorFlow CNN benchmark. The benchmark employs data-parallelism with `Horovod` to use multiple GPUs. In data-parallelism, each device holds a model copy but performs a backpropagation for a different batch of input data, combining the gradients after each step using an all-reduce collective operation.

C. Accelerators

GPUs have become the standard hardware for accelerating neural network training. However, while processing power has advanced rapidly in recent years, memory bandwidth has not kept pace, leading to potential bottlenecks. Accelerators based on data-flow architectures, such as Graphcore IPUs [10], offer a promising alternative by addressing these limitations. Unlike traditional architectures that rely on a shared memory hierarchy, IPUs leverage distributed per-core memory, which allows for

TABLE I
SYSTEMS ANALYZED WITH CARAML. CPU CORES ARE GIVEN AS *72 c* FOR 72 CORES.

Platform	**GH200** JEDI	**GH200** JURECA	**H100** JURECA	**H100** WestAI	**MI200** JURECA	**IPU-M2000** JURECA	**A100** JURECA
Accelerator	4× NVIDIA GH200-120GB (1× 72 c Grace, 1× H100)	1× NVIDIA GH200-480GB (1× 72 c Grace, 1× H100)	4× NVIDIA H100 GPU (PCIe)	4× NVIDIA H100 GPU (SXM5)	4× AMD MI250 GPU (OAM)	4× Graphcore GC200 IPU	4× NVIDIA A100 GPU (SXM4)
CPU			2× 72 c Intel Xeon Platinum 8452Y	2× 32 c Intel Xeon Platinum 8462Y	2× 48 c AMD EPYC 7443	2× 48 c AMD EPYC 7413	2× 64 c AMD EPYC 7742
CPU–Acc. Connect (intra-node)	NVLink-C2C 900 GB/s		PCIe Gen 5 128 GB/s		PCIe Gen 4 64 GB/s		
Acc.–Acc. Connect (intra-node)[1]	NVLink4 900 GB/s	-	NVLink4[2] 600 GB/s	NVLink4 900 GB/s	Infinity Fabric 500 GB/s	IPU-Link[3] 256 GB/s	NVLink3 600 GB/s
Interconnect internode[4]	4× IB NDR (4×200 Gbit/s)	-	-	2× IB NDR (2×400 Gbit/s)	2× IB HDR (2×200 Gbit/s)	-	2× IB HDR (2×200 Gbit/s)
Memory	4× 120 GB LPDDR5X (CPU), 4× 96 GB HBM3 (GPU)	480 GB LPDDR5X (CPU), 96 GB HBM3 (GPU)	512 GB DDR5-4800 (CPU), 80 GB HBM2e (GPU)	512 GB DDR5-4800 (CPU), 94 GB HBM2e (GPU)	512 GB DDR4-3200 (CPU), 128 GB HBM2e (GPU)	512 GB DDR4-3200 (CPU)	512 GB DDR4-3200 (CPU), 4× 40 GB HBM2e (GPU)
TDP / device	680 W†	700 W†	350 W	700 W	560 W	300 W	400 W
JUBE Tag	`JEDI`	`GH200`	`H100`	`WAIH100`	`MI250`	`GC200`	`A100`

[1] Bidirectional bandwidths per device.
[2] GPU0 and GPU1 and GPU2 and GPU3 are connected through NVLink bridges, each with 12 NVLink4 connections (each 25 GB/s).
[3] Each IPU in a node is connected to other IPUs in- and out-of-node with 10 *IPU-Links*. Intra-node, an IPU connects to two other IPUs with 2 links, and with one IPU with 4 links. At 32 GB/s bidirectional bandwidth per link, an IPU has hence an accumulated intra-node connection bandwidth of 256 GB/s.
[4] NVIDIA InfiniBand is abbreviated to *IB*.
† The TDP for the GH200 superchips is for the full package, i.e. including the CPU and GPU devices.

faster data loading due to the physical proximity of memory to each core. This design enables all cores to operate independently, making the processing of irregular or sparse neural network architectures more efficient. In terms of Flynn's taxonomy [11], this can be considered a MIMD (multiple instruction streams, multiple data streams) architecture, while GPUs follow a SIMD (single instruction stream, multiple data streams) approach.

Figure 1 provides a list of accelerators that have been explored in this work utilizing the CARAML benchmarks. The complete node configurations, including CPU, memory and interconnect, are documented in Table I.

We examine two generations of NVIDIA GPUs (A100 and H100) in various configurations. The A100 node is part of the JURECA DC [12] cluster at Jülich Supercomputing Centre, while the other nodes are part of the JURECA evaluation platform [13] or the WestAI [14] cluster. The two H100 node variants mainly differ in the CPU model, intra-node bandwidth and the amount of GPU memory.

The NVIDIA GH200 superchip is built up from a Grace CPU and a Hopper GPU that are connected on chip by a high-bandwidth, low-latency interconnect. The GPU can directly access the CPU memory without explicit transfers, potentially accelerating hybrid workloads. The two GH200 nodes we investigate differ in their configuration: A node of the JEDI system contains 4 NVIDIA GH200s, while the GH200 node of the JURECA evaluation platform only has one GH200.The amount of per-node memory is the same in both cases.

The AMD MI200 node from the JURECA evaluation platform contains four MI250 GPUs. Similar to the GH200, it can be seen as combining multiple devices on a single chip. Each MI250 contains two Graphics Compute Dies, that are seen as a GPU by the operating system. From that viewpoint, each node would contain 8 GPUs. Each pair of devices may have a different transfer bandwidth [15].

There are many programming paradigms with different compatibility characteristics [16]. `CUDA` is well-established and serves as a blueprint for other programming models (HIP), or as a backend for portability layers such as `OpenACC` or `SYCL`. While the software ecosystems for data-flow architectures are growing, they are not yet as mature and supported by third-party software as GPU programming models. A clear leader or common standard among the competitors has not yet been established. Graphcore systems can be instructed using the `Poplar` software development kit. At a higher level, machine learning frameworks such as `PyTorch` and `TensorFlow` can be seen as portability layers, serving various platforms via vendor-specific backends. This approach of using a common codebase for various architectures is followed by the benchmarks in this paper wherever possible. The limits of this approach are described in section V.

D. Related Work

Since the advent of modern computing technologies, benchmarks have been an important tool to assess the performance of hardware systems, spanning from consumer devices [17] and

accelerators [18], to HPC clusters as major research facilities [19]. Synthetic benchmarks, which concentrate on specific yet commonly used compute patterns, are valuable in this context [20]. However, the performance metrics they provide can be difficult to apply to more complex, real-world applications.

Benchmarks that incorporate the workloads of real applications, or their variations, are thus extremely valuable for evaluating hardware capabilities. They also offer a way to assess the effects of parameter choices or code optimizations.

In the field of machine learning, the MLPerf set of benchmark suites [21] is an established industry standard supported by all major vendors. Various MLPerf suites focus on training at device or cluster level [22], [23], or on performing inference across various devices [24]. Results are collected as a coordinated effort in a yearly industry-wide competition. Based on the premise of lacking (performance) portability, vendors are expected to port and optimize a reference code for their architecture, showcasing its capabilities. Established competition rules act as clearly defined guardrails and make a comparison possible. In this competitive context, the choice of time-to-solution as a benchmark metric over throughput-based metrics makes sense, as the latter ones could be optimized at the expense of the first one. The downside of the time-to-solution metric, which here refers to the runtime until a specified accuracy is achieved, is its high computational cost.

Similar to MLPerf, the SPEC benchmarks [18] are a consortium-driven effort to benchmark the performance of hardware systems in terms of general-purpose algorithms. It's closed source code is not freely available and does not contain ML specific workloads.

CARAML on the other hand focuses on the user rather than the vendor perspective. As a free and open source framework under a permissive MIT license, it empowers users to evaluate the out-of-the-box performance of accelerators with minimal code adaptions. It relies on two widely used machine learning frameworks (`PyTorch` and `TensorFlow`) as portability layers. Focusing on throughput and performance in images or tokens per second allows for quick evaluation without the need to perform full training runs, even with limited computational resources. This resource-efficiency and immediate feedback, together with CARAML's high level of automation, allows its user to rapidly explore an architecture's (hyper-) parameter space or to perform parameter ablation studies.

Recent years have seen growing efforts to minimize environmental impact of HPC systems, motivated by an ongoing climate crisis and a changing energy landscape. To this end, measuring energy consumption of HPC hardware and workloads has come more into focus. Efforts to assess energy efficiency on a cluster level [25] are now accompanied by the development of tools for a more fine-grained assessment [26]. AI workloads such as large language models in particular have come under scrutiny due to the significant energy footprint required for their training [27], [28]. The jpwr tool was developed as a compact prototype to fulfil the requirements of incorporating energy measurement in the CARAML benchmark suite.

III. The CARAML benchmark

The CARAML codebase is accessible at https://github.com/FZJ-JSC/CARAML. The repository features a clean and straightforward structure, consisting of a `README.md` file and two main directories: `llm_training` and `resnet50`. By focusing on these two representative benchmark cases, the repository is easy to navigate and deploy, enabling users to quickly gather relevant metrics without unnecessary complexity.

Our approach is to maximize automation in order to facilitate ease of use, reproducibility, and compactness of the benchmarks. To this end, the benchmark codes themselves are not part of the repository. The repository contains the scaffolding code, that automatically downloads codebases, packages, and execution containers and sets up the required compute environment. To achieve the outlined level of automation, CARAML relies heavily on the JUBE [29], [30] automation and benchmarking framework. The reported energy measurements are extracted from hardware counters using the self-developed `jpwr` framework.

The used Docker container images are provided by the hardware vendors, containing the respective machine learning frameworks, with additional steps to make them usable for CARAML benchmarks. More details can be found in section V.

A. Benchmark Details

1) LLM Training: For the LLM training benchmark in CARAML, a GPT decoder model is trained from scratch using a subset of the OSCAR data that is preprocessed using GPT-2 tokenizers. The benchmark for NVIDIA GPUs is curated from a specific commit version of Megatron-LM[1] to make it compatible with all NVIDIA GPU generations. For AMD devices, the BigCode Project fork[2] is used, which contains adaptations to utilize `ROCm` instead of `CUDA`. In the case of Graphcore, a forked version of a vendor-provided application example[3] is used. All benchmarks employ jpwr to provide a power measurement feature. This necessitates performing a `git patch` to Megatron-LM after cloning the repository. All these steps are automated by JUBE. The patch further contains fixes to streamline the benchmark's automated execution.

The sizing of the specific network architectures, e.g. in terms of number of layers and parallelization configuration, are performed with the aim to fully utilize the hardware system's capabilities. This means that not all accelerators train the exact same model. Due to the different programming paradigm and having only 4 GC200 IPUs available during creation of the suite, only a 117M parameter GPT decoder LLM was trained on the Graphcore device, instead of the 800M parameter GPT decoder model that is used on NVIDIA and AMD hardware. Further JUBE configurations for models containing 13B and 175B parameters are provided in the suite. They can be executed when necessary resources are available, and were tested on NVIDIA GH200 devices.

Megatron-LM leverages several optimization features, including flash attention, distributed optimizers, activation recomputation, mixed precision, and rotary positional embeddings, in conjunction with various parallelization strategies such as data, tensor, pipeline, and sequence parallelism. For models with 800M parameters, which fit within a single device on both AMD and NVIDIA hardware, only data parallelism is utilized. For the larger model configurations with 13B and 175B parameters, tensor, pipeline, and sequence parallelism are also enabled. The parallel implementation is done using *PyTorch Distributed*. The benchmark's throughput is measured

[1]https://github.com/NVIDIA/Megatron-LM
[2]https://github.com/bigcode-project/Megatron-LM
[3]https://github.com/chelseajohn/examples

in terms of tokens/second which is calculated by dividing global_batch_size × sequence_length with elapsed_time_per_iteration. The benchmark uses all the possible optimization features like flash attention, rotary positional embeddings, distributed optimizers and mixed precision and is terminated based on the value of the --exit-duration-in-mins command line argument in Megatron-LM.

To work around the limited available memory of the Graphcore IPU, we chose a smaller GPT model size (117M), and further employ pipeline parallelism to distribute the model's layers (including the embedding layer) to four devices, using Poplar [31]. This decreases the memory demand per device. Our evaluated system (IPU-POD4 with four IPU, see Table I) contains four GC200 IPUs, which means that we use a single replica and a single instance (i.e. no data parallelism). Scaling to more nodes can be done by employing more instances using PopDist and Horovod. It is possible to use synthetic data with the benchmark instead of OSCAR data. The benchmark is executed for one epoch and the throughput is measured again in terms of tokens/second, but calculated by dividing global_batch_size with elapsed_time_per_iteration, as the global_batch_size is given in number of tokens and not number of samples.

2) ResNet50 Training: The CARAML ResNet50 benchmark for NVIDIA and AMD is curated from the forked version of official TensorFlow benchmarks[4]. The main addition in the forked version is the power measurement using jpwr (see section subsubsection III-A4). The benchmark uses the ResNet50 model, but other models like inception3, vgg16, and alexnet can also be utilized. The benchmark trains a ResNet50 model from scratch for 100 iterations and outputs the throughput in images/second computed by dividing the global_batch_size by elapsed_time_per_iteration. Training data can be passed as an argument to the benchmark, or else synthetic data is used. The benchmark is scaled to multiple GPUs using data parallelism implemented with Horovod. The benchmark uses mixed precision and the openXLA [32] compiler for accelerating training.

When targeting Graphcore devices, CARAML uses a forked version of vendor-provided application examples, similar to LLM training, that incorporated power measurement using jpwr (see section subsubsection III-A4. The benchmark uses the ResNet50 model, but ResNet18 and ResNet34 models can also be executed with modified configuration files. Similar to the TensorFlow benchmark, used for NVIDIA and AMD devices, a ResNet50 model is trained from scratch for one epoch and images/second is used as the throughput metric. Using the Poplar library, provided by Graphcore, the benchmark can be scaled to multiple IPUs using data parallelism when using a single instance; for multiple instances, PopDist and Horovod are used. It is possible to pass training data as an argument or use synthetic data generated either on the host CPU and transferred to the IPU or generated directly on the IPU. Mixed precision training and other custom device optimizations like memory and device mapping, 8-bit transfers, and fused preprocessing are used to make the training efficient.

[4]https://github.com/chelseajohn/tf_cnn_benchmarks

3) Automation with JUBE: The JUBE [29], [30] workflow environment facilitates reproducibility and ease of use of the provided benchmarks. Each of the two benchmarks is fully characterized by configuration files, called *JUBE scripts*, where hyperparameters and execution steps are defined.

A JUBE script can be in XML or YAML format. For illustrative reasons, we provide the scripts for LLM training in YAML (llm_training/llm_benchmark_nvidia_amd.yaml and llm_training/llm_benchmark_ipu.yaml) and the script for training the image classification model in XML (resnet50/resnet50_benchmark.xml).

The execution steps include downloads, compilation, training, and verification. Different systems and steps are executed by supplying the required *tags*. The JUBE runtime interprets the script, resolves dependencies and submits jobs to the Slurm batch system. The job templates are populated from a system-specific configuration file, platform.xml, making the approach system-agnostic. JUBE presents the benchmark results, including a throughput figure-of-merit (images/second and tokens/second) along with energy consumed per device in Watt hour (Wh) during the course of the model training in the benchmark, in compact tabular form after execution.

The JUBE scripts can be utilized to define a set of experiments aimed at exploring the impact of various parameters on performance, such as batch size, optimizers, and learning rate. JUBE simplifies the process of conducting model layout and scaling experiments by automatically generating job scripts with different parameter permutations. Beyond machine learning hyperparameters, this exploration can be extended to system-level configurations, including number of CPU cores or threads, CPU binding strategies and accelerator affinity in terms of NUMA domains.

4) Power Measurements with jpwr: jpwr is a modular tool for measuring power and energy of different compute devices, currently supporting methods for querying AMD and NVIDIA GPUs, as well as specific methods for getting system power measurements from NVIDIA Grace-Hopper chips and Graphcore IPUs. The code is available at https://github.com/FZJ-JSC/jpwr/ under an AGPL-3.0 license. It can be used either as a command-line tool jpwr, or within Python code as a context manager get_power. The command line tool wraps other applications, specifying the method to extract power measurements via command line switch, determined by the examined hardware.

The following example shows how jpwr is used to get energy measurements for an application call stress-ng --gpu 8 -t 5 on an AMD GPU supporting ROCm, writing the results to a CSV file:

```
jpwr --methods rocm --df-out energy_meas --df-filetype
↪  csv stress-ng --gpu 8 -t 5
```

The next example shows how the context manager can be invoked for GH200 GPU and system measurements, to save gathered metrics in the object measured_scope:

```
from jpwr.gpu.pynvml import power
from jpwr.sys.gh import power as gh_power
from jpwr.ctxmgr import get_power
[...]
met_list=[power(),gh_power()]
with get_power(met_list, 100) as measured_scope:
    application_call()
print(measured_scope.df)
```

The context manager initiates a power-measurement loop in a separate thread, which periodically queries power consumption

using device-specific interfaces, saving data points along with their timestamps. At the end of the operation, these data points are used to calculate the total amount of energy consumed. The device-specific interfaces, referred to as "methods", are implemented as individual modules that can be passed to the context manager.

As backends, vendor-provided libraries and a `sysfs` interface are employed to extract hardware counters. NVIDIA GPUs use `pynvml` [33], which provides bindings for the NVIDIA Management Library, which is also used by the popular NVIDIA System Management Interface (`nvidia-smi`). For AMD GPUs, we use the Python module `rsmiBindings` [34], which is shipped with the ROCm System Management Interface (`rocm-smi`). Graphcore IPUs are queried using the Graphcore IPU Info library (`gcipuinfo` [35]), which is also available as a Python module. To also include CPU metrics for GH200 CPU/GPU superchips, the Linux kernel's `sysfs` interface is used by reading data from device files under the path `/sys/class/hwmon/` [36] (called `gh` in the tool). Multiple backends can be used at the same time, which is useful for GH200, where both `pynvml` and `sysfs` methods can be used, or in exotic systems with multiple types of accelerator. The modular structure of these methods ensures they are easily maintained and allows for the seamless addition of further interfaces.

jpwr saves the measured data as Pandas DataFrames internally and this data can be exported. For the command-line tool, specifying `--df-out` and `--df-filetype` arguments sets the output directory and filetype for the DataFrames accordingly. The tool will save all available power and energy data in the specified directory using the specified filetype (HDF5's `.h5` or `.csv`). For the context manager, the `measured_scope.df` DataFrame contains the power measurement data, and `energy_df, additional_data = measured_scope.energy()` returns an energy DataFrame derived from the measurement data in `energy_df` and a dictionary of additional DataFrames in `additional_data`.

The tool works per-node, i.e. for an MPI or other types of multi-node applications, writing the result files would result in a race condition. To combat this, the tool allows adding a suffix to all result files with the `--df-suffix` option. Furthermore, the suffix string can contain a `%q{VARIABLE}` statement, which will interpret the `VARIABLE` environment variable at runtime. i.e. for a job submitted with Slurm, `--df-suffix "%q{SLURM_PROCID}"` can be used to add the MPI rank as a suffix to the file names.

B. Benchmark Execution

After cloning the CARAML repository, each benchmark can be executed with just a few JUBE commands, providing the desired benchmark's JUBE script and a tag to define the target architecture. The system tags can be found in the overview of considered systems in Table I.

For the LLM training benchmark, the required system and model parameters are to be set in `llm_training/llm_benchmark_nvidia_amd.yaml` (for NVIDIA and AMD systems) or `llm_training/llm_benchmark_ipu.yaml` (for Graphcore).

For the ResNet50 benchmark, the required system and model parameters and the path to the downloaded ImageNet data need to be set in `resnet50_benchmark.xml`.

More details can be found in Appendix A.

IV. Results

In the following, we report throughput measurements obtained from the hardware systems described in Table I alongside the corresponding energy consumption data collected during the execution of the CARAML benchmarks. The benchmarks were conducted with careful consideration of CPU binding, MPI threading, and GPU affinity to ensure optimal conditions on the examined machines.

A. LLM Training

Figure 2 provides throughput results in `tokens/second` per GPU for NVIDIA systems of various generations and the AMD MI250 GPU, for global batch sizes from 16 to 4096. All experiments train a decoder-only transformer model with 800M parameters using a subset of the OSCAR dataset preprocessed using GPT-2 tokenizers. Since the 800M model fits on a single device, data parallelism can be employed to scale the model across multiple accelerators within a node. The model was trained on an entire node for each system, utilizing data parallelism and micro-batch-size of 4, when multiple accelerators were available. All systems contained 4 GPUs, except the GH200 node in JURECA which has only one (see Table I).

For the AMD node, we report two sets of results to draw a complete picture and make a nuanced comparison possible in the context of Multi-Chip Modules (see details in Table I). The first set of results (*AMD MI250:GCD*) uses 4 GCDs (2 GPUs) with data parallelism of 4, while the second set (*AMD MI250:GPU*) uses all 8 available GCDs (4 GPUs) with data parallelism of 8. When using data parallelism of 8 the global batch size of 16 is not possible since it is not divisible by micro-batch-size times data parallel. In each case, the data is normalized per data parallel (i.e. by 4 and 8, respectively). Additionally, we present the average total energy consumed per GPU during one hour of model training, along with an energy efficiency metric, calculated as the number of tokens processed per unit of energy consumed.

In general, one can see the performance improvements in more recent GPU hardware generations, with GH200 nodes yielding a throughput of up to $47\,505$ Tokens/s/GPU, $2.45\times$ higher than throughput achieved on A100 GPU nodes. This can be alluded to having more cores and SMs, faster CPU-to-GPU-NVLink connection, TDP, and fast memory. It is evident, that choosing a larger batch size is beneficial for throughput. However, when training a neural network in a production setting, this increased GPU utilization must be balanced against the potential drawback of slower convergence, which could impact the overall training efficiency of the neural network.

Different variants of accelerators were examined, namely the H100 incorporated in the JURECA evaluation platform (referred to here as JRDC) and the H100 in the WestAI cluster, as well as the GH200 in JEDI and the GH200 in JRDC (see Table I), and different results can be inferred.

When comparing the two GH200 configurations, we see that a device on a single-accelerator node (GH200 (JRDC)) yields a $20\,\%$ higher performance than a device on a multi-accelerator node (GH200 JEDI)), accompanied by a $20\,\%$ higher energy consumption. Hence, the `tokens/energy` efficiency per device is similar; even slightly better for the less performant JEDI case. On JEDI, all devices engage in data-parallel model training, and the additional communication overhead, together with the lower

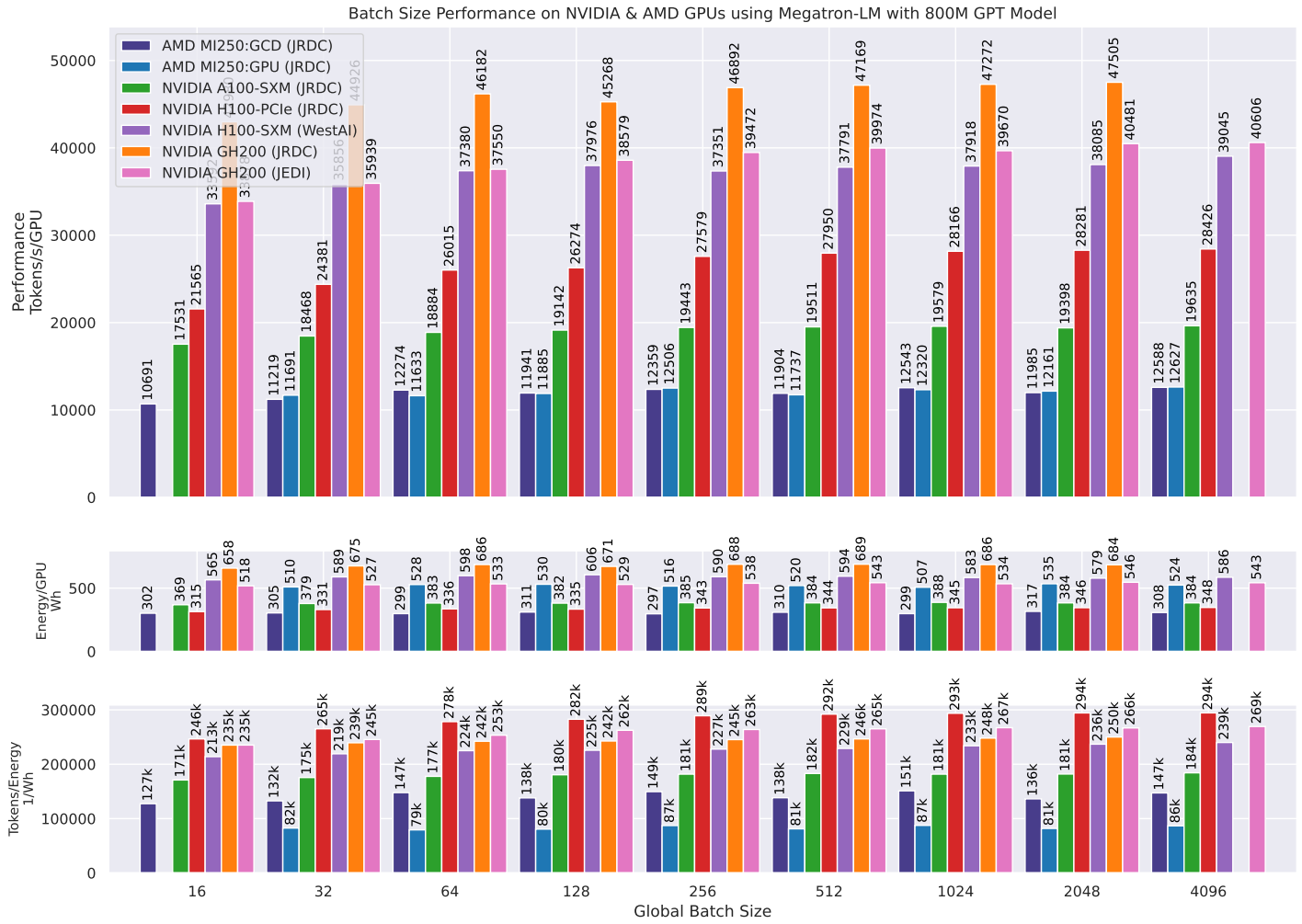


Fig. 2. Throughput and energy efficiency for LLM training on NVIDIA and AMD systems using a 800M GPT model.

amount of CPU memory available per device, could be the reason for the lower performance per device.

Another large throughput difference can be observed between the two H100 systems, with the WestAI variant processing 1.3× as many tokens as the JRDC variant. This could be due to the higher available bandwidth from NVLink connections between GPUs and the SXM GPU form factor, which comes with a higher power envelope (TDP).

For AMD, using 4 GCDs (2 GPUs) performs slightly better per device than using 8 GCDs (4 GPUs), again representing the overhead of higher parallelization. This overhead leads to a higher energy consumption per device and lower energy efficiency when using 8 GCDs (4 GPUs).

In terms of energy efficiency, the results indicate that the H100-PCIe (JRDC) outperforms all other devices by up to 25 %, even against the newer technology of GH200 chips, which provide a throughput twice as high. This is likely related to the limited power budget of the PCIe card, moving its operation mode to a more power-efficient spot. Another factor could be that the other H100 variants, especially the GH200s, are not yet completely saturated in the examined benchmark scenario, as they have higher SM counts.

Table II provides performance and energy efficiency results for the Graphcore machine. Here, the vendor benchmark specifies IPU POD16 as the minimum requirement for GPT-2 PyTorch model training [37]. As we only have access to an IPU POD4, a smaller GPT model with 117M parameters is used to benchmark the hardware with energy measurements for global batch sizes from 64 to 16 384. The larger batch sizes may not be practical for model convergence, but were investigated to understand the limitations of the system. The model layers are split across 4 IPUs and trained for one epoch (global batch size samples) using synthetic data. We see in Table II that the throughput (`tokens/second`) increases with the batch sizes, saturating the accelerator, and uses a maximum of 33 W h. The performance is very low compared to GPUs but can partially be explained by the required pipeline parallelism. This form of parallelism introduces a pipeline bubble [2] and is not as efficient as data parallelism.

B. *ResNet50 Training*

The ResNet50 training benchmark was performed on all available systems with global batch sizes 16 to 2048.

Figure 3 reports the throughput of the ResNet50 training process in images per second on a single device on various systems (see Table I), as well as consumed energy for the whole epoch (processing all images of the input dataset once), and energy efficiency in images/Wh. ImageNet data was used as input, containing 1 281 167 images.

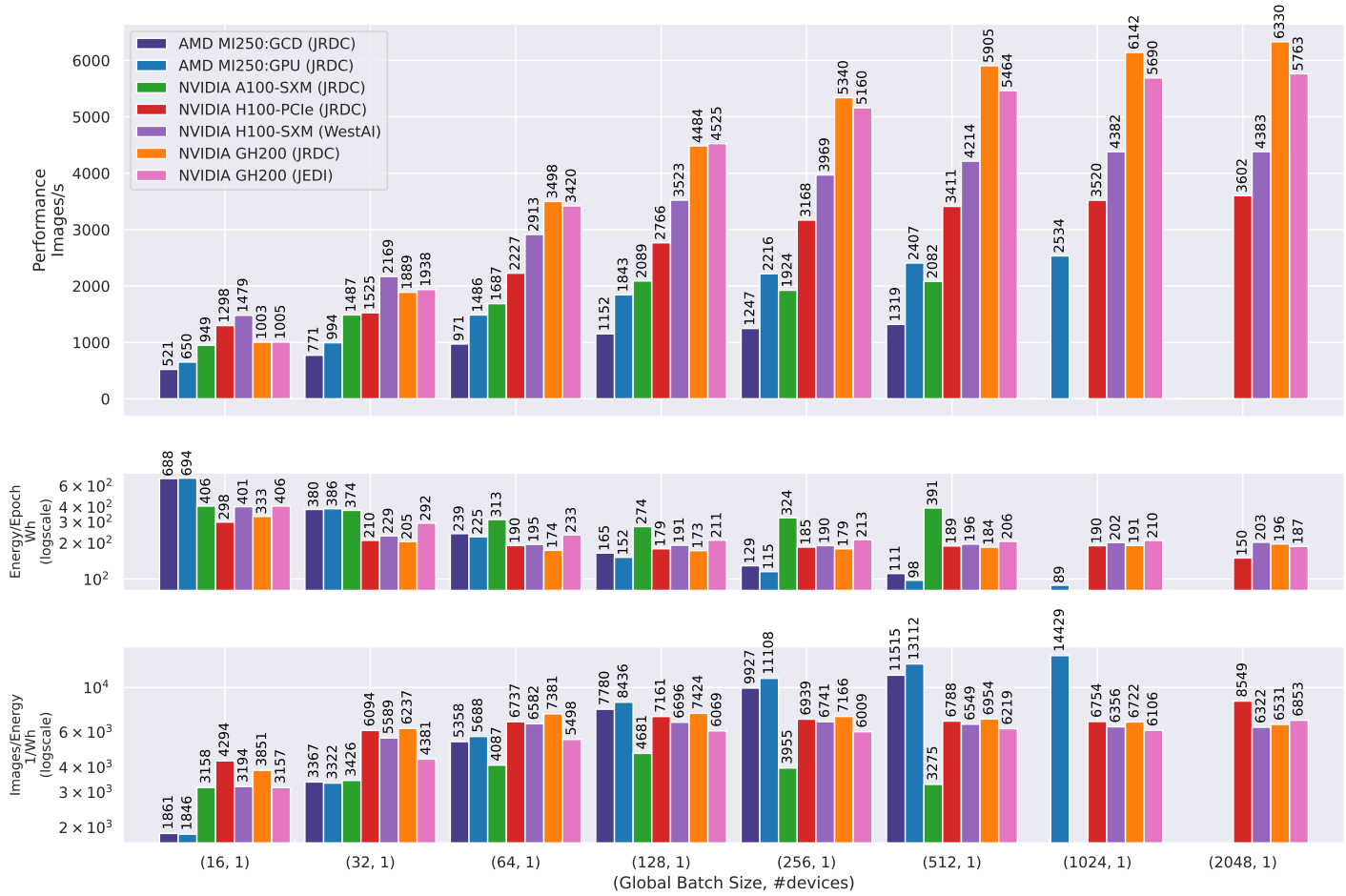


Fig. 3. Throughput and energy consumption for `ResNet50` model training on a single device of NVIDIA and AMD systems.

TABLE II
PERFORMANCE AND ENERGY CONSUMPTION DATA FOR TRAINING A 117M GPT MODEL FOR ONE EPOCH ON IPU GC200 IN M2000 POD4. UNITS FOR THE ENTRIES ARE GIVEN IN THE SECOND ROW.

Batch Size	**Tokens/Time** 1/s	**Energy/Epoch/IPU** Wh	**Tokens/Energy** 1/Wh
64	64.99	15.68	4.08
128	97.21	18.20	7.03
256	129.96	18.37	13.93
512	155.72	18.56	27.60
1024	172.94	19.07	53.71
2048	183.37	20.05	102.13
4096	188.88	21.88	187.22
8192	191.86	25.47	321.34
16384	193.41	33.00	496.43

As expected, the performance increases from older to newer GPU generations. Similar to the LLM training benchmark (see Figure 2), we see the WestAI H100 node, with higher-TDP SXM form factor, outperforming the PCIe variant when considering performance per node, with similar energy footprints. Again, GH200 (JRDC) performs better than GH200 (JEDI). This holds true especially for larger batch sizes, which can likely benefit from $4\times$ as much available CPU memory per GPU, allowing for faster data loading. As seen for `ResNet50` before, the PCIe-variant of the H100 (JRDC), appears to have the best energy efficiency amongst the NVIDIA GPUs, closely followed by the GH200 (JRDC). The reasons seem to stem from a combination of TDP, memory capacity, and bandwidth.

To provide data for more nuanced comparisons in the context of MCMs, as already explained for the LLM results, also for `ResNet50`, two benchmark runs on the AMD MI250 node were conducted. One run (AMD MI250:GPU) utilized 1 GPU (2 GCDs), requiring data parallelism of 2, and another run (AMD MI250:GCD) utilized only 1 GCD, without parallelism. Using 2 GCDs naturally leads to a higher throughput, and the device is used more efficiently. This leads to slightly lower amounts of energy needed to process the whole dataset, and a slightly higher energy efficiency.

The AMD MI250 gives the best efficiency in terms of images per unit of energy for higher batch sizes, while for smaller batches the H100 and GH200 (JRDC) devices are more energy efficient.

In the case of Graphcore, the vendor-based TensorFlow `ResNet50` model training benchmark contains optimizations catering to the IPU execution strategy. When running the benchmark for a single epoch, the IPU first compiles an optimized

TABLE III
PERFORMANCE AND ENERGY DATA FOR TRAINING A RESNET50 MODEL FOR ONE EPOCH ON A SINGLE IPU GC200 IN M2000 POD4. UNITS FOR THE ENTRIES ARE GIVEN IN THE SECOND ROW.

Batch Size	Images/Time	Energy/Epoch	Images/Energy
	1/s	Wh	1/Wh
16	1827.72	32.09	39925.87
32	1857.90	31.73	40382.19
64	1879.29	31.75	40346.18
128	1888.11	31.67	40452.50
256	1887.23	31.58	40563.65
512	1891.74	31.49	40689.85
1024	1893.07	31.50	40668.79
2048	1889.87	31.53	40636.28
4096	1891.58	31.51	40660.14

model graph, which takes close to an hour. It is excluded from the timings presented here. The compiled model graph upon execution is able to complete an epoch with 1 281 167 samples in 10 to 15 minutes.

Table III provides results on throughput, energy consumption, and energy efficiency for the `ResNet50` benchmark on a Graphcore GC200 IPU. The model performance does not scale on increasing the global batch size. This is likely related to the limitation of not being able to process a micro-batch-size of more than 16 due to limited on-chip RAM (SRAM) and having to execute multiple sequential calls to fetch data from the chip-external memory (DRAM). The energy efficiency compared to classical GPUs looks very promising.

The heatmaps in Figure 4 (4a to 4g) explore the impact of data-parallelism and global batch size, giving throughput results for all examined architectures. These extensive results for various configurations can be used to estimate the training time required to train a `ResNet50` model efficiently.

The heatmaps also contain multi-node results for systems where resources were available. The throughput increases with the global batch size as expected. In the case of Graphcore, the highest throughput was obtained using 2 IPUs for a global batch size of 16, this can be explained due to the batch size fitting into the on-chip RAM, and using fewer IPU links for data transfer.

Judging the results overall, it can be seen that GPUs tend to perform better for larger batch sizes and number of GPUs. In nearly all GPU cases, the best value achieved is for the largest batch size using most GPUs, indicating that the GPUs are not yet saturated, and the limiting factor is the available memory. For Graphcore IPUs performance behavior is relatively flat over a large range of the parameter space, and is best when the batch size fits into the on-chip RAM and multiple calls to DRAM is avoided.

V. TECHNICAL CHALLENGES

A. Software

In CARAML, benchmarks are curated to compare the accelerators with minimal parameter adjustments using open source codebases. For this purpose, NVIDIA and AMD GPUs are tested using the same baseline code. The distinct execution strategy of Graphcore IPUs necessitates a separate codebase, posing a challenge to ensure comparable hyperparameters between IPUs and GPUs.

The NVIDIA Megatron-LM code base incorporates hardware specific optimizations, with the current version focusing on the Hopper architecture, for example making use of its *Transformer Engine*. To ensure compatibility with the Ampere architecture, CARAML rolls back to a Megatron-LM commit that can be executed on all devices without losing other performance optimizations.

Typically, new optimizations, such as flash attention, are first made available on NVIDIA hardware, with AMD accelerators receiving support later. Currently, the `flash-attention2` implementation in `ROCm` is still under development and supports head dimensions only up to 128, whereas the CUDA implementation on NVIDIA GPUs already supports head dimensions up to 256 and `flash-attention3`.

B. Containers

Using containers for reproducible workloads promises to simplify the creation of reproducible environments, but setting them up in performance-sensitive HPC environments can be challenging. Issues include locating vendor-supported containers for specific software versions and managing conflicting package dependencies within the container, often with limited permissions.

Solving these challenges required multiple iterations of container testing, finally leading to the creation of custom containers, inheriting from vendor-provided containers. To manage the installation of additional packages within the container, we utilize `pip` with the `--prefix`, `--no-deps`, and `--ignore-installed` options, and manually adjust the `PYTHONPATH`. The container's isolation from the system environment necessitates defining custom bind paths and the development of wrapper scripts to export environment variables.

Utilizing the shared resources of HPC systems requires usage of job schedulers (Slurm) and message transport libraries (MPI, NCCL). As the employed containers need to bring their own MPI installation, some effort needs to be taken to align the out-of-container distribution setup with the in-container installation. For our setup, the involved PMIx configurations need to be explicitly made compatible by manually setting `PMIX_SECURITY_MODE=native` out-of-container, but within a Slurm-distributed job[5].

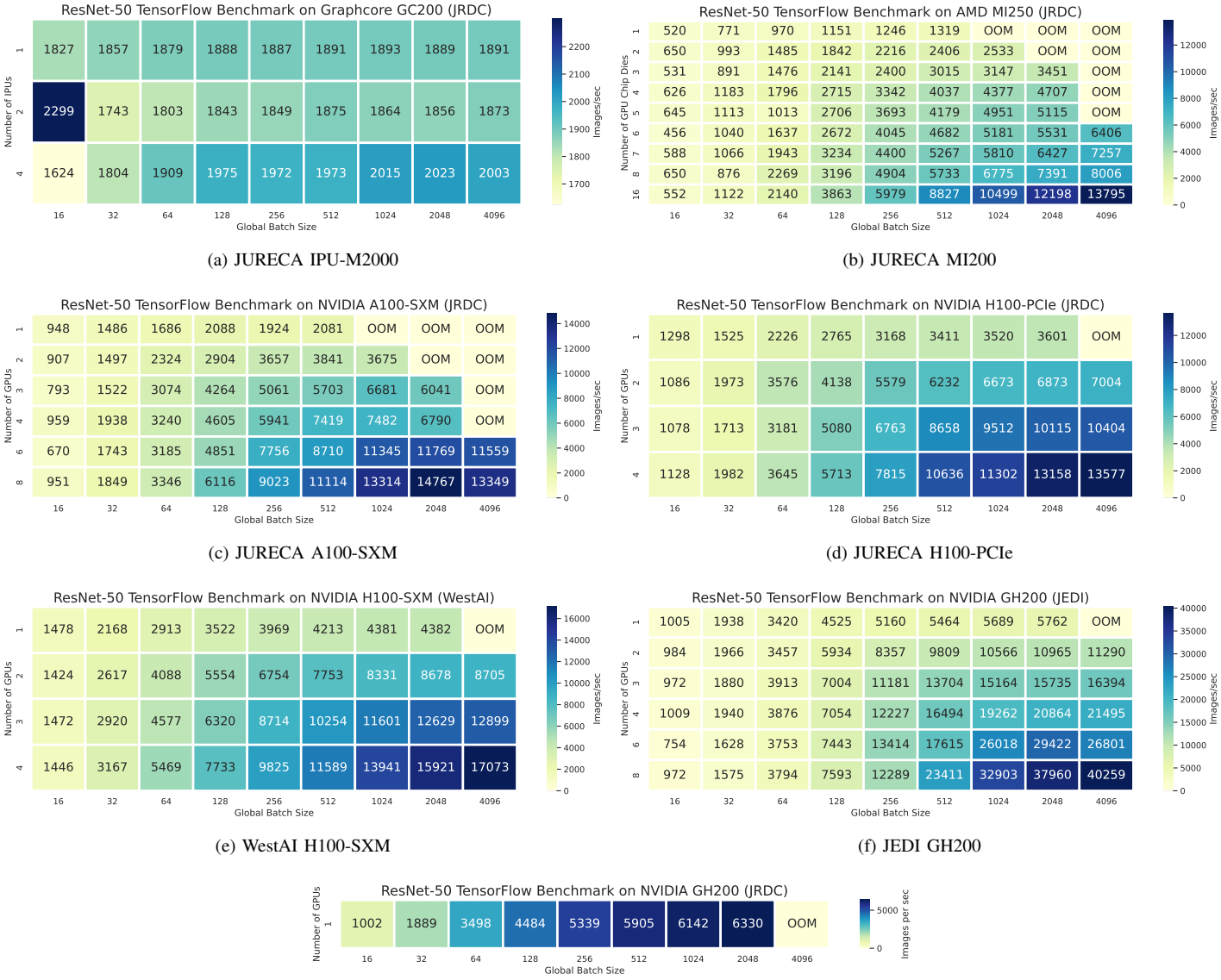
C. System

In order to ensure a smooth user experience, some non-trivial, HPC-related technical fixes have to be implemented. We document them here to highlight on system-specific issues, which can be helpful when adding more system support for future benchmarks.

As with many HPC systems, the systems at Jülich Supercomputing Centre feature a high-speed interconnect (InfiniBand) between the nodes. IP connectivity is only available using InfiniBand devices (IP over Infiniband, IPoIB). `PyTorch` needs to be made aware of the different format of the hostname, which contains an appended `i` to the `MASTER_ADDR` variable[6]. Further, a fixed `torchrun.py` script is required for execution on such a

[5] `srun env PMIX_SECURITY_MODE=native apptainer ...`

[6] Coincidentally, the `ib0` interface is ordered after the `en0` interface, such by default the wrong interface is picked. For Jülich Supercomputing systems, the hostnames of IPoIB network are equal to the `en0`-network hostnames with an appended `i`.



(a) JURECA IPU-M2000

(b) JURECA MI200

(c) JURECA A100-SXM

(d) JURECA H100-PCIe

(e) WestAI H100-SXM

(f) JEDI GH200

(g) JURECA GH200

Fig. 4. Throughput for `ResNet50` training depending on number of GPUs and global batch size on various systems. *OOM* stands for Out of Memory, i.e. the batch size is too large for the memory of the device.

system. While not yet being fixed in the main `PyTorch` codebase[7] a patched version is available in the benchmark repository (`llm_training/aux/fixed_torch_run.py`).

With AI-distributed training often depending on external software like `PyTorch` and `Horovod` [38] to spawn multiple processes, errors due to conflicting MPI ranks with Slurm are encountered at times and have to be fixed.

Further, during the benchmarking process, the critical impact of correct CPU binding, optimal number of threads, and GPU affinity on performance for each system was carefully studied. It was found that a GPU-centric approach to affinity is useful, creating one Slurm task per GPU and distributing them to CPU cores with affinity to respective GPUs. At the same time, it is important to create CPU masks that are open enough for NCCL to place its helper thread.

[7]The issue is open upstream since 2022: https://github.com/pytorch/pytorch/issues/73656; meanwhile a patched version is provided on PyPI for convenience, https://pypi.org/project/torchrun-jsc/.

JEDI, as one example, features four Grace-Hopper superchips, so that the Slurm options `--ntasks=4 --cpus-per-task=72 --gpus-per-task=1` give the proper affinity. JURECA A100 nodes, as another example, feature EPYC processors in which not all CPU chiplets have GPU affinity. Due to this, explicitly targeting the *proper* NUMA domains with `--cpu-bind` is a complex, but useful approach.

VI. Conclusions

As AI continues to experience rapid growth and the market is seeing a growing influx of AI accelerators, evaluating accelerator performance using real world applications is crucial. In this paper, we introduced CARAML, a benchmark suite designed to assess AI workloads on accelerators with energy measurements. CARAML uses the JUBE framework to create compact, automated benchmarks for both LLM and Computer Vision training. The benchmarks incorporate the modular jpwr tool to measure energy consumption. CARAML is further capable to perform ablation studies to identify hardware and model configurations

for optimal performance. The details of the framework and results obtained using CARAML on seven different accelerators systems from NVIDIA, AMD and Graphcore that differ either in generation or configuration were discussed in detail.

The results confirm that the latest accelerator generations yield a better performance, but the energy efficiency is influenced by more factors in the hardware and network configuration. The GH200 generally gives the best performance, related to the CPU-to-GPU-NVLink connection, TDP, and fast memory. The PCIe-flavor of the H100 usually gives the best energy-efficiency, a result of operation at an efficient power operating point.

While the surveyed Graphcore accelerator system could not yield a competitive performance to classical GPUs, the results on energy efficiency are very promising, outperforming GPUs in this regard for some benchmarks. This relies on code that is optimized for the execution on an IPU's data-flow architecture, which can yield performance improvements.

Several technical challenges were encountered while automating the CARAML benchmark setup. Solutions required a deep understanding of networking specifics, AI framework backends, and how containers interact with their environment.

As future work, we plan to further develop CARAML by incorporating continuous benchmarking capabilities and enhancing its usability. We also aim to expand the suite by including additional AI training and inference benchmarks.

ACKNOWLEDGMENT

Part of this work was funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the project OpenGPT-X (project no. 68GX21007D). Additional support was provided by the EuroHPC Joint Undertaking under Grant Agreement 955513, co-funded by the German Federal Ministry of Education and Research (BMBF) under funding reference 16HPC029 through the MAELSTROM project.

Work presented here made extensive use of JURECA-DC, the JURECA-DC Evaluation Platform, the WestAI infrastructure, and the JUPITER enablement platform JEDI, which we greatly acknowledge.

We would like to express our gratitude to Jan Ebert and Jan Robert Finkbeiner for their valuable insights and discussions on configuring neural network architectures.

REPRODUCIBILITY

The source codes of CARAML and jpwr are available at https://github.com/FZJ-JSC/CARAML and https://github.com/FZJ-JSC/jpwr. The results shown in the paper can be reproduced using CARAML by following the instruction entailed in subsection III-B and the corresponding readme files.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using Megatron-LM," 2021. [Online]. Available: https://arxiv.org/abs/2104.04473

[3] C. M. John, C. Penke, A. Herten, J. Ebert, and S. Kesselheim, "OpenGPT-X — training large language models on HPC systems," Poster presented at the International Supercomputing Conference (ISC) 2023, May 2023, Hamburg, Germany.

[4] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, pp. 341–353, 2023.

[5] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=mZn2Xyh9Ec

[6] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020. [Online]. Available: https://arxiv.org/abs/1909.08053

[7] K. Chellapilla, S. Puri, and P. Simard, "High Performance Convolutional Neural Networks for Document Processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, G. Lorette, Ed., Université de Rennes 1. La Baule (France): Suvisoft, Oct. 2006, http://www.suvisoft.com. [Online]. Available: https://inria.hal.science/inria-00112631

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[10] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza. Dissecting the Graphcore IPU Architecture via Microbenchmarking. [Online]. Available: http://arxiv.org/abs/1912.03413

[11] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[12] P. Thörnig, "JURECA: Data centric and booster modules implementing the modular supercomputing architecture at jülich supercomputing centre," *J. Large-scale Res. Facil. JLSRF*, vol. 7, no. A182, Oct. 2021. [Online]. Available: https://doi.org/10.17815/jlsrf-7-182

[13] JURECA Evaluation Platform Overview. [Online]. Available: https://apps.fz-juelich.de/jsc/hps/jureca/evaluation-platform-overview.html

[14] WestAI. [Online]. Available: https://westai.de/

[15] A. Herten, "First Benchmarks with AMD Instinct MI250 GPUs at JSC," 2022. [Online]. Available: https://juser.fz-juelich.de/record/916416

[16] ——, "Many cores, many models: GPU programming model vs. vendor compatibility overview," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1019–1026. [Online]. Available: https://doi.org/10.1145/3624062.3624178

[17] UserBenchmark. [Online]. Available: https://www.userbenchmark.com

[18] G. Juckeland, W. Brantley, S. Chandrasekaran, B. Chapman, S. Che, M. Colgrove, H. Feng, A. Grund, R. Henschel, W.-M. W. Hwu, H. Li, M. S. Müller, W. E. Nagel, M. Perminov, P. Shelepugin, K. Skadron, J. Stratton, A. Titov, K. Wang, M. van Waveren, B. Whitney, S. Wienke, R. Xu, and K. Kumaran, "SPEC ACCEL: A standard application suite for measuring hardware accelerator performance," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer International Publishing, 2015, pp. 46–67. [Online]. Available: https://doi.org/10.1007/978-3-319-17248-4_3

[19] A. Herten, S. Achilles, D. Alvarez, J. Badwaik, E. Behle, M. Bode, T. Breuer, D. Caviedes-Voullième, M. Cherti, A. Dabah, S. El Sayed, W. Frings, A. Gonzalez-Nicolas, E. B. Gregory, K. Haghighi Mood, T. Hater, J. Jitsev, C. John, J. H. Meinke, C. I. Meyer, P. Mezentsev, J.-O. Mirus, S. Nassyr, C. Penke, M. Römmer, U. Sinha, B. von St. Vieth, O. Stein, E. Suarez, D. Willsch, and I. Zhukov, "Application-driven exascale: The JUPITER benchmark suite," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '24. New York, NY, USA: Association for Computing Machinery, 2024, to appear.

[20] K. Asanović, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, no. 10, p. 56–67, oct 2009. [Online]. Available: https://doi.org/10.1145/1562764.1562783

[21] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G.-Y. Wei, and C.-J. Wu, "MLPerf: An industry standard benchmark suite for machine

learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020. [Online]. Available: https://doi.org/10.1109/MM.2020.2974843
[22] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. Janapa Reddi, T. Robie, T. St John, C.-J. Wu, L. Xu, C. Young, and M. Zaharia, "MLPerf training benchmark," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 336–349. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2020/file/411e39b117e885341f25efb8912945f7-Paper.pdf
[23] S. Farrell, M. Emani, J. Balma, L. Drescher, A. Drozd, A. Fink, G. Fox, D. Kanter, T. Kurth, P. Mattson, D. Mu, A. Ruhela, K. Sato, K. Shirahata, T. Tabaru, A. Tsaris, J. Balewski, B. Cumming, T. Danjo, J. Domke, T. Fukai, N. Fukumoto, T. Fukushi, B. Gerofi, T. Honda, T. Imamura, A. Kasagi, K. Kawakami, S. Kudo, A. Kuroda, M. Martinasso, S. Matsuoka, H. Mendonca, K. Minami, P. Ram, T. Sawada, M. Shankar, T. t. John, A. Tabuchi, V. Vishwanath, M. Wahib, M. Yamazaki, and J. Yin, "MLPerf™ HPC: A holistic benchmark suite for scientific machine learning on HPC systems," in *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. Los Alamitos, CA, USA: IEEE Computer Society, 11 2021, pp. 33–45. [Online]. Available: https://doi.org/10.1109/MLHPC54614.2021.000092
[24] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00045
[25] W. Feng and K. W. Cameron, "The Green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, 2007. [Online]. Available: https://doi.org/10.1109/MC.2007.445
[26] J. P. Gutiérrez Hermosillo Muriedas, K. Flügel, C. Debus, H. Obermaier, A. Streit, and M. Götz, "perun: Benchmarking energy consumption of high-performance computing applications," in *Euro-Par 2023: Parallel Processing*, J. Cano, M. D. Dikaiakos, G. A. Papadopoulos, M. Pericàs, and R. Sakellariou, Eds. Cham: Springer Nature Switzerland, 2023, pp. 17–31. [Online]. Available: https://doi.org/10.1007/978-3-031-39698-4_2
[27] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," 2021. [Online]. Available: https://arxiv.org/abs/2104.10350
[28] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of BLOOM, a 176B parameter language model," *J. Mach. Learn. Res.*, vol. 24, no. 1, mar 2024. [Online]. Available: https://www.jmlr.org/papers/volume24/23-0069/23-0069.pdf
[29] T. Breuer, J. Wellmann, F. Souza Mendes Guimarães, C. Himmels, and S. Luehrs, "JUBE," May 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11394333
[30] S. Lührs, D. Rohe, A. Schnurpfeil, K. Thust, and W. Frings, "Flexible and Generic Workflow Management," in *Parallel Computing: On the Road to Exascale*, ser. Advances in parallel computing, vol. 27, International Conference on Parallel Computing 2015, Edinburgh (United Kingdom), 1 Sep 2015 - 4 Sep 2015. Amsterdam: IOS Press, Sep 2016, pp. 431 – 438. [Online]. Available: https://doi.org/10.3233/978-1-61499-621-7-431
[31] poplibs. [Online]. Available: https://github.com/graphcore/poplibs
[32] xla. [Online]. Available: https://github.com/openxla/xla
[33] pynvml. [Online]. Available: https://pypi.org/project/pynvml/
[34] pyrsmi. [Online]. Available: https://github.com/ROCm/pyrsmi
[35] Graphcore IPU Info Library (gcipuinfo). [Online]. Available: https://docs.graphcore.ai/projects/gcipuinfo/en/latest/
[36] NVIDIA Grace Performance Tuning Guide. [Online]. Available: https://docs.nvidia.com/grace-performance-tuning-guide.pdf
[37] IPU GPT-2 README. [Online]. Available: https://github.com/graphcore/examples/tree/master/nlp/gpt2/pytorch#gpt-2
[38] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," 2018. [Online]. Available: https://arxiv.org/abs/1802.05799

APPENDIX

To execute the CARAML benchmark, clone the CARAML repository and use the corresponding JUBE script and a tag to identify the target architecture. The available system tags are listed in the overview of systems in Table I

a) LLM Training:

- Set the required system and model parameters in `llm_training/llm_benchmark_nvidia_amd.yaml`(for NVIDIA and AMD systems) or `llm_training/llm_benchmark_ipu.yaml`(for Graphcore)
- To pull the required container and build packages, use container tag as:
  - NVIDIA A100 and H100 GPUs

```
jube run
↪ llm_training/llm_benchmark_nvidia_amd.yam
↪ --tag container H100
```

  - NVIDIA GH200 and JEDI GPUs

```
jube run
↪ llm_training/llm_benchmark_nvidia_amd
↪ --tag container GH200
```

  - AMD MI250

```
jube run
↪ llm_training/llm_benchmark_nvidia_amd
↪ --tag container MI250
```

  - Graphcore GC200

```
jube run
↪ llm_training/llm_benchmark_ipu.yaml
↪ --tag container
```

- To run the benchmark with defined configurations for `800M` GPT model with tokenized OSCAR data provided with the repository do:

```
jube run
↪ llm_training/llm_benchmark_nvidia_amd.yaml
↪ --tag A100 800M
```

  A100 can be replaced with H100, WAIH100, GH200, JEDI and MI250 for the respective systems and `800M` can be replaced with `13B` and `175B` for systems with available node resources.
- To run the benchmark with defined configurations for `117M` GPT model on Graphcore with synthetic data do:

```
jube run
↪ llm_training/llm_benchmark_ipu.yaml
↪ --tag 117M synthetic
```

  If tag synthetic is not given, the benchmark will use the tokenized OSCAR data.
- To combine the energy data into a single CSV file and post-process results do:

```
jube continue
↪ llm_training/llm_benchmark_nvidia_amd_run
↪ -i last
```

  Or

```
jube continue
↪ llm_training/llm_benchmark_ipu_run
↪ -i last
```

- To get the final result in tabular form do:

```
jube result
↪ llm_training/llm_benchmark_nvidia_amd_run
↪ -i last
```

  Or

```
jube result
↪ llm_training/llm_benchmark_ipu_run
↪ -i last
```

b) ResNet50 Training:

- Set the required system and model parameters and the path to downloaded ImageNet data in `resnet50_benchmark.xml`
- To pull the required container, use container tag as:
  - NVIDIA A100 and H100 GPUs

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag container H100
```

  - NVIDIA GH200 and JEDI GPUs

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag container GH200
```

  - AMD MI250

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag container MI250
```

  - Graphcore GC200

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag container GC200
```

- To run the benchmark with defined configurations do:

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag A100
```

  Or with synthetic data

```
jube run
↪  resnet50/resnet50_benchmark.xml
↪  --tag A100 synthetic
```

  A100 can be replaced with H100, WAIH100, GH200, JEDI, MI250 and GC200 for the respective systems.
- To combine the energy data into a single CSV file and post-process the results do:

```
jube continue
↪  resnet50/resnet50_benchmark_run -i
↪  last
```

- To get the final result in tabular form do:

```
jube result
↪  resnet50/resnet50_benchmark_run -i
↪  last
```