Energy Efficiency in Machine Learning Models

Greta Ru-Mei Zu - 261099681

August 2025

1 Introduction

In recent years, machine learning has emerged as one of the most transformative technologies across industries, powering applications ranging from natural language processing and image recognition to scientific discovery and autonomous systems. At its core, machine learning involves designing models that learn patterns from data, a process that requires significant resources in the form of computational power, time, and human expertise. The process of training these models includes adjusting millions or even billions of parameters to minimize error, which can take days or even weeks on large-scale clusters of GPUs or TPUs. While the field continues to push the boundaries of model performance and scalability, another increasingly important consideration has emerged: the environmental and energy costs associated with training and deploying large-scale machine learning models. As discussed in many papers, such as the MLPerf Power benchmarking report by Tschand et al [7], the ecological impact of ML systems, ranging from microwatts in embedded systems to megawatts in training cutting-edge models, has become a topic of growing concern within both academic and industrial communities.

There is an expanding community working with the aim of quantifying and reducing the energy footprint of machine learning systems. Tools such as CodeCarbon, by Schmidt et al [6], provide estimates of carbon emissions and energy consumption associated with model training, allowing researchers and practitioners to better understand and mitigate their environmental impact. Studies have explored energy and carbon considerations in specific ML workflows, such as the fine-tuning of large language models like BERT by Wang et al [8], highlighting the trade-offs between performance and sustainability. On the systems side, distributed training frameworks and architectural innovations like Mixture-of-Experts (MoE) models have been developed to increase scalability while lowering computational costs. For instance, one of the frameworks: DeepSpeed-MoE, as presented by Rajbhandari et al [4], demonstrates how sparsely activated models can significantly reduce training latency and hardware requirements without compromising model quality.

This report investigates the energy consumption characteristics of training large-scale Mixture-of-Experts models by implementing Qwen3-MoE using DeepSpeed and the FastMoE (FMoE) library, with energy and carbon impact measured via CodeCarbon. The work combines an overview of the relevant theoretical foundations, and experimental runs are conducted under controlled settings to quantify the relationship between model configuration, computational efficiency, and environmental footprint. The findings provide insights into the trade-offs between performance and energy consumption, contributing to sustainable practices in large-scale model training.

2 Background

In order to measure and analyze the energy efficiency of modern machine learning models, the focus was placed on Mixture of Experts (MoE) applied to two transformer-based architectures: Switch-Transformers and Qwen3-MoE. These models are trained using two different distributed frameworks, DeepSpeed and FastMoE (FMoE), and evaluated in terms of their energy consumption and carbon emission using the CodeCarbon library. This section provides a high-level overview of these models, training frameworks, and the energy tracking tool used in the experiments.

2.1 Machine Learning Models

Mixture of Experts (MoE) Traditional machine learning models are often implemented as dense networks, where every parameter is involved in every forward pass. While this approach is straightforward and general-purpose, it becomes computationally and memory-intensive as model sizes scale, in particular

during pre-training. Mixture of Experts (MoE) introduces sparsity by activating only a subset of the model's parameters for each input, reducing the amount of computation per inference or training step. MoE architectures typically use a gating or routing mechanism that selects a small number of experts, usually small feed-forward neural networks, to process each token or input. This enables the construction of models with significantly more parameters while keeping the computational budget constant. In distributed machine learning settings, MoE models can scale efficiently by assigning different experts to different devices or nodes, making them particularly well-suited for large-scale training on multi-GPU clusters [5].

Switch-Transformers The Switch-Transformers [1] is a scalable MoE architecture that replaces every other dense feed-forward layers in a Transformer model with a sparse mixture of expert layers. Each token is routed to only one expert (top-1 routing), which simplifies computation and reduces communication overhead compared to earlier MoE models. This design leads to models with a much larger parameter count but only marginally increased training and inference time. Switch-Transformers can be configured in encoder-only, decoder-only, or encoder-decoder formats. In the implementation of the project, a sparse encoder-decoder architecture where the dense MLP blocks are replaced with Switch MoE layers is used. Each expert is a small feed-forward network, and the model uses a learned routing mechanism to assign tokens to experts dynamically, as can be seen in Figure 1 [1]. This allows a retention of a high model capacity with relatively low computational cost, which is ideal for distributed training and energy efficiency studies.

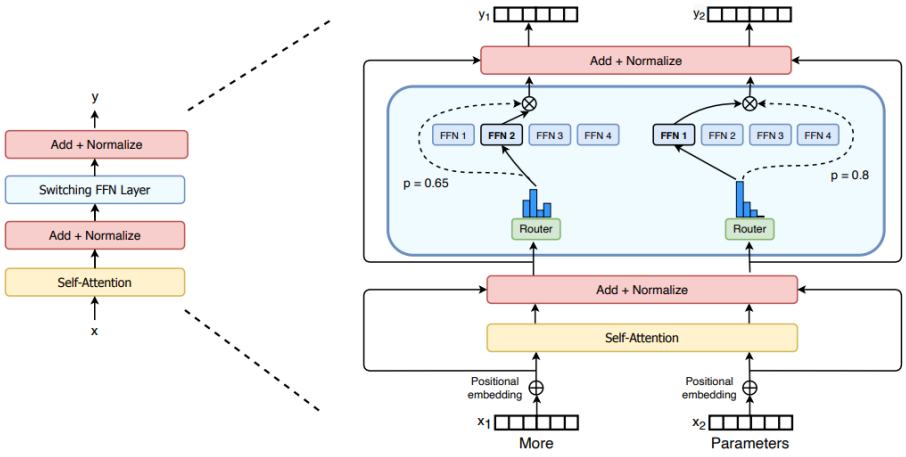


Figure 1: Switch-Transformers MoE structure

Qwen3-MoE Qwen3-MoE is a decoder-only transformer model designed for language generation tasks. It employs a sparse MoE structure at each decoder layer, where a token is routed to a subset of small MLP-based experts based on a gating function. The structure of Qwen3-MoE uses expert blocks in every decoder layer, creating much denser expert layers. The architecture consists of:

- **Embedding Layer:** A lookup table maps token IDs to 2048-dimensional vectors.
- **Decoder Layers:** Each of the 24 decoder layers includes a self-attention block, followed by a sparse MoE block. In the project implementation, this was changed to 8 decoder layers.
- **Sparse MoE Block:** Implements top-k expert routing, enabling dynamic expert selection per token while maintaining efficiency.
- **Rotary Embedding:** Positional information is added using rotary embeddings.
- **RMSNorm:** A normalization technique that ensures training stability using root mean square statistics.

2.2 Training Frameworks (Trainers)

DeepSpeed DeepSpeed is an open-source deep learning optimization library designed to scale model training to extremely large sizes efficiently and affordably [4]. It provides memory optimizations, communication primitives, and support for Mixture-of-Experts via DeepSpeed-MoE. When training MoE models, DeepSpeed replicates the non-expert layers across devices while distributing expert layers among available GPUs. This allows the system to parallelize the expensive computations without duplicating the entire model, thus enabling efficient large-scale training, as shown in Figure 2. In the experiments, DeepSpeed was used to train both Switch-Transformers and Qwen3-MoE by integrating the expert layers with DeepSpeed's MoE module, which handles token dispatch, expert routing, and gradient aggregation efficiently.

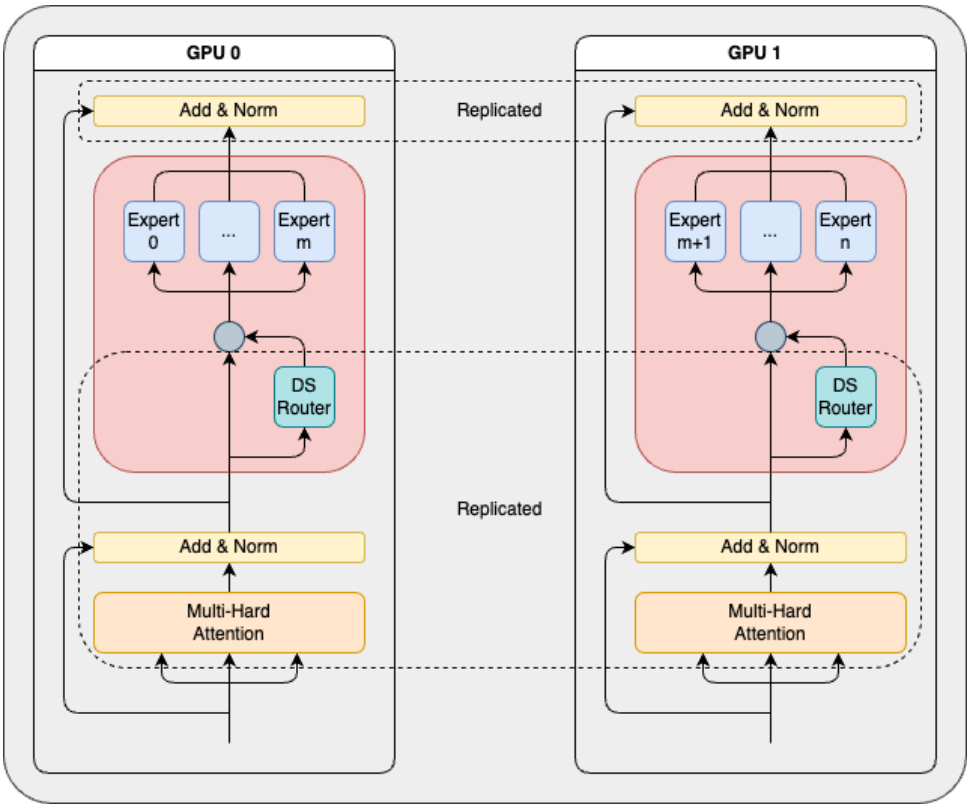


Figure 2: DeepSpeed MoE structure

FastMoE Fast Mixture-of-Expert (FMoE) is another distributed training framework optimized for sparse models. FMoE differs from DeepSpeed primarily in its handling of padding and expert dispatching. It avoids padding tokens to fixed sizes, which can waste computation, and instead relies on more dynamic expert grouping. FMoE was used to train Switch-Transformers and Qwen3-MoE models with a similar number of experts and routing configurations as in DeepSpeed to compare the training efficiency and energy use under different system designs.

Simple Training Framework As a baseline, a non-distributed version of each model using a single-GPU training framework was also implemented. This setup allows the evaluation of the overhead introduced by communication and expert dispatch in distributed settings. To ensure a fair comparison, the number of experts was configured to match the total number used in the distributed setups (e.g., 16 experts on a single GPU vs. 128 experts across 8 GPUs - 16 experts per GPU).

2.3 Energy Tracking

CodeCarbon To quantify the energy consumption and environmental impact of each training configuration, the software tool CodeCarbon [6] was used. It is a Python package that estimates power usage

and carbon emissions associated with a given script. CodeCarbon monitors system components, including the CPU, GPU, and RAM, and estimates carbon emissions using regional carbon intensity factors based on the geographical location of the machine. In the implementation, CodeCarbon is instrumented at three levels:

- **Full Training Session:** Measures the total energy consumption for an entire training run.
- **Per Step:** Measures energy used per training iteration.
- **Substeps:** Tracks energy for forward pass, backward pass, and optimizer step separately.

This granularity allows us to break down which parts of the training pipeline contribute most to energy use, and to compare the efficiency of different models and training framework setups in detail.

3 Description

3.1 Overview

This project investigates the energy efficiency of sparsely activated transformer models - namely, Switch-Transformers and Qwen3-MoE - by tracking their training processes using the CodeCarbon library. These models are trained on multi-GPU systems using two popular distributed frameworks: DeepSpeed and FMoE. By tracking energy consumption and carbon emissions throughout the training process, the goal is to gain insights into the ecological cost of training large-scale Mixture of Experts (MoE) models and explore solutions for how to improve the resource consumption without losing model quality.

The primary goal of the project is to compare different configurations of these models and training frameworks to identify key factors that influence energy consumption. The aim is to highlight opportunities for energy-efficient training without degrading model performance. Parameters such as the choice of training framework, batch size, number of experts, and learning rate are varied in a controlled manner to better understand their effects on energy and computational efficiency.

3.2 Implementation

Qwen3-MoE Initially, both Qwen2-MoE and Qwen3-MoE were considered for experimentation. The decision leaned toward Qwen3-MoE after encountering out-of-memory (OOM) issues when attempting to load Qwen2-MoE. Qwen3-MoE introduced several architectural changes over Qwen2-MoE, such as removing the "shared expert" layer and increasing the default number of experts from 60 to 128, resulting in a more standardized and comparable design.

However, running Qwen3-MoE out of the box was still not feasible on the server hardware, which has only 32 GB of memory per GPU. To mitigate this and have a runnable Qwen model with DeepSpeed on the server, the model complexity was reduced: the number of hidden transformer layers was lowered from the standard 24 to 8, the number of experts was cut from 128 to 64, and the batch size was reduced to 4. These modifications reduced memory usage enough to run on the setup, but they also mean that the model used in our experiments is significantly smaller and less expressive than the original.

Qwen3-MoE and DeepSpeed To integrate and train Qwen3-MoE with DeepSpeed, the model architecture was modified by replacing its feed-forward MLP layers with sparsely activated MoE layers provided by DeepSpeed's MoE module. Each MoE layer is constructed using a wrapper that integrates DeepSpeed's MoE module into the existing Qwen3-MoE layer structure so that expert routing and auxiliary loss computation are handled by DeepSpeed's optimized MoE implementation.

In a hypothetical system with significantly more GPU memory, the unmodified Qwen3-MoE could be run in its full 24-layer, 128-expert configuration and use larger batch sizes. This would likely improve both the model's representational capacity and training stability, as well as improve the training speed, enabling it to capture more complex patterns in the data and achieve higher accuracy and better generalization.

Codecarbon CodeCarbon is used to track energy consumption and estimate carbon emissions at multiple levels of training granularity: the full training session, each training step, and substeps within a step (forward, backward, optimizer). It does so by monitoring the energy usage of the CPU, GPU(s), and RAM throughout execution, then converting this into emissions using the carbon intensity of electricity in the specified location. For the experiment runs, the `OfflineEmissionsTracker` was configured with the region set to "Quebec," where hydroelectric power yields a comparatively low carbon intensity. Tracking hooks are integrated via the `TrainerStats` interface, allowing the training loop to remain agnostic to the specific tracker implementation.

Due to CodeCarbon being originally designed to track long-running executions, where any overhead would be proportionally small enough to be insignificant, some of the libraries used carry a bit of CPU overhead. However, in the experimental runs for this project, the runs involve very short measurement intervals. Thus, certain CPU-measurement libraries with high overhead had to be disabled, which triggers CodeCarbon to skip these time-intensive libraries and fall back on less accurate but more rapid estimations, accepting a small loss in accuracy since GPU usage dominates the training energy profile. All tracker outputs are then saved in a structured folder hierarchy by run, model, and GPU rank, simplifying later analysis and plotting across configurations.

Learning Rate Experiments Multiple learning rates were experimented with to find suitable values that produced stable training loss curves without sudden drops, plateaus, or divergence. The following learning rates were tested:

- **Switch-Transformers:** 4e-8, **1e-7**, 7e-7, 1e-6, 1.4e-6, 1.8e-6, 2e-6, 2.5e-6, 3e-6, 4.5e-6
- **Qwen3-MoE:** 7e-7, **1e-6**, 8e-6, 1e-5, 1.8e-5, 2.5e-5, 5e-5

Loss Tracking Loss values were tracked throughout training and stored alongside CodeCarbon outputs for correlation analysis. Each step's loss was saved to an array, and at the end of training, all loss values were exported to a .csv file using a Python Pandas dataframe with the GPU rank, iteration number and loss value. The loss is computed as the sum of the model's primary task loss (eg. Cross-entropy) and an additional MoE router loss. This MoE loss is obtained by summing the auxiliary (AUX) balancing losses from all MoE layers in the model and scaling this sum by a fixed coefficient (0.1). The combined loss drives the gradient computation, although only the primary task loss is recorded for tracking and analysis.

Data Collection and Automation Data was collected using a set of Bash scripts that automated training under various parameter configurations. Scripts were designed to control the number of GPUs, batch sizes, learning rates, and iterations, and to run both distributed and single-GPU versions of the models. A data collection script collects all the CodeCarbon data files, and a separate cleanup script was created to merge the energy and loss logs for each run, organize output folders, and prepare aggregated files for plotting and analysis. Further scripts and Python classes were created for plotting purposes, including loss, duration, emissions, energy and energy vs. loss plotting with different combinations of parameters.

3.3 Chosen Parameters

Through preliminary experiments, parameters were determined by choosing the ones that had the most significant impact on energy efficiency and training stability, as well as the ones that made more sense to be compared with each other. For example, comparing different model types (e.g., Qwen vs. Switch-Transformers) introduced too many uncontrolled variables, and Deepspeed-ZeRO offloading techniques, while relevant to memory optimization, were outside the scope of the focus on energy consumption for the project. Ultimately, the focus of the comparisons was decided following key parameters:

- Batch size (BS)
- Number of GPUs
- Number of Experts
- Number of Iterations

- Learning Rates (LR)

It was determined that batch size should be evaluated in terms of the number of samples seen rather than a fixed number of iterations, where $samples = num\_gpus * batch\_size * iterations$. The number of samples directly reflects how much training data the model has actually processed, whereas batch size alone can be misleading. Increasing the batch size means that more samples are seen per iteration, and keeping the iteration count fixed would result in more total training, thus a biased comparison. By fixing the total number of samples instead, we ensure a fair comparison, meaning that each experimental configuration is exposed to the same amount of training data, and any performance differences can be attributed to the parameter being varied rather than differences in total training. This shows the comparison of energy consumption between faster training (more samples seen at each iteration with less iterations) and slower training (more iterations with less samples each time). This definition also extends naturally to experiments where the number of GPUs changes, since more GPUs increase throughput (more samples per iteration). In such cases, fixing the number of samples means that higher-throughput setups simply complete the training in fewer iterations, preserving comparability across GPU counts.

4 Experiments

4.1 Set up

The following experiments were all performed using the given setup:

- Hardware:
  - DGX system, dual Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz CPUs, with 252GB per CPU, and 8 Nvidia V100 SXM2 32GB GPUs
- OS:
  - Ubuntu 20.04.6 LTS

4.2 Data

The **Colossal Cleaned Crawled Corpus (C4)** [3] data set was used, downloaded from HuggingFace [2]. It is a colossal, cleaned version of Common Crawl's web crawl corpus "https://commoncrawl.org". While the full English split contains over 750GB of text, only a single shard was used: `multilingual/c4-en.tfrecord-00000-of-11264.json.gz`, which contains 273,356 samples. This file was selected specifically since the experiment of the project consists of pre-training, which needs cleaned and massive in scale datasets. Since the focus of the experimentation was also on language tasks, this dataset suited the needs very well.

Given the limited computational time and early experimental nature of the work, this sample size is sufficient to observe trends across several parameter settings.

4.3 Parameters

The key parameters used during the experiments are summarized below. These were chosen based on a combination of prior literature, hardware constraints, and empirical tuning. The values are the final ones chosen after prior experimentation were ran, as mentioned in section 4.4.

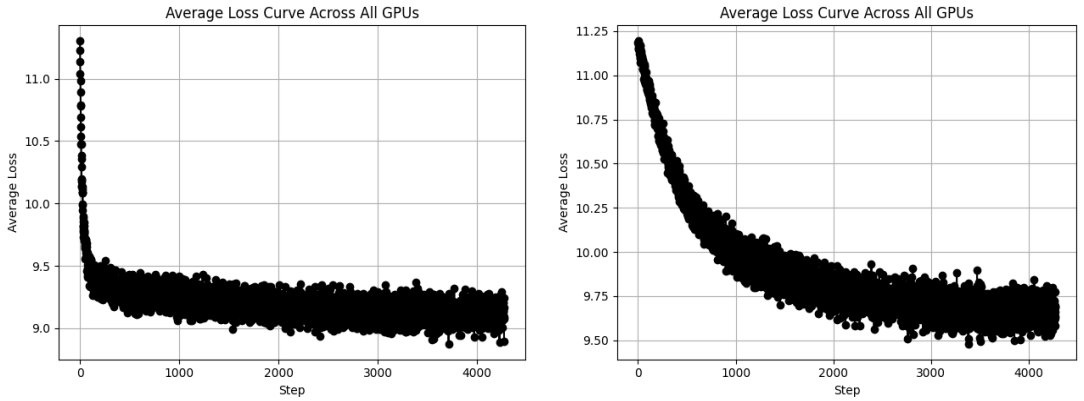
Parameter	**Switch-Transformers Value**	**Qwen3-MoE Value**
Training framework	Deepspeed / FMoE	Deepspeed / FMoE
Num GPUs	8	8
Batch Size	2 & 8	2 & 4
Num Experts	128	64
Dataset Split	$num\_gpus * batch\_size * iterations$	$num\_gpus * batch\_size * iterations$
Learning Rate	1e-7	1e-6

Table 1: Training configuration used for Switch-Transformers and Qwen3-MoE models

4.4 Loss and Energy Trends

4.4.1 Loss Trends

Below are some training loss curves and energy consumption plots for representative runs. These provide early indicators of convergence behaviour and efficiency tradeoffs. From these runs, the final learning rate - as shown in the parameters table - was chosen.

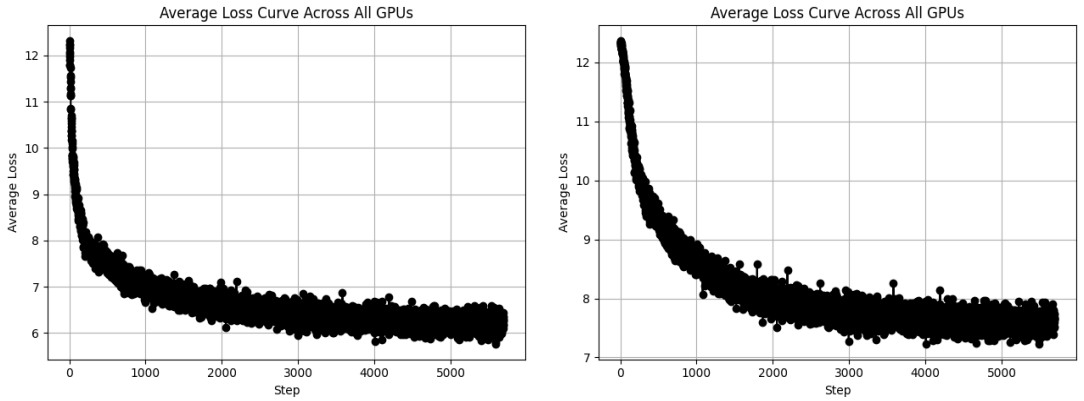


(a) 128 experts, FMoE training framework, BS=8, LR=3e−6

(b) 128 experts, FMoE training framework, BS=8, LR=1e−7

Figure 3: Loss Curves For Switch-Transformers with Batch Size (BS) and Learning Rate (LR)

Figure 3a shows a too quick drop in the loss curve while Figure 3b shows a steadier decline in loss.



(a) 64 experts, FMoE training framework, BS=6, LR=8e−6

(b) 128 experts, FMoE training framework, BS=6, LR=1e−6

Figure 4: Loss Curves For Qwen3-MoE with Batch Size (BS) and Learning Rate (LR)

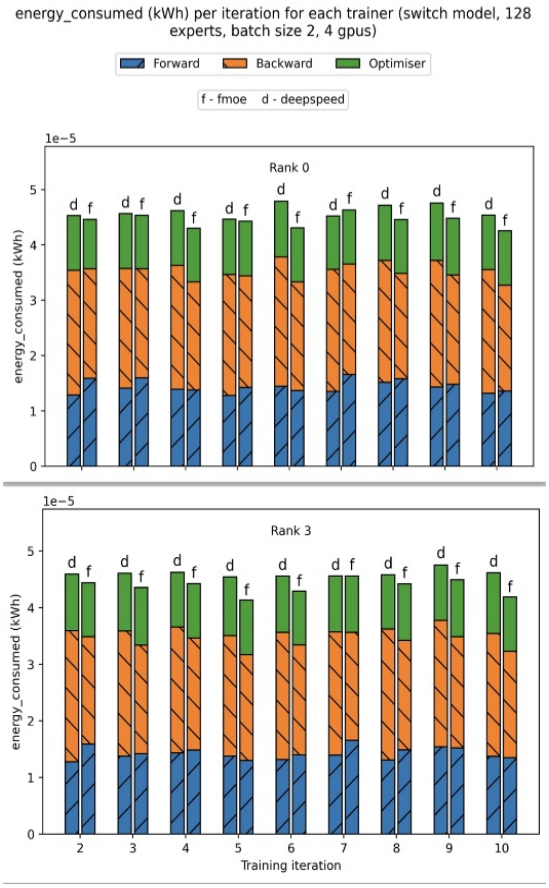
Qwen3-MoE, on the other hand, showed relatively stable loss decline, thus the curve with a slightly slower decrease in loss was chosen: Figure 4b shows a loss that dips down to around 7.5 in a slower drop especially at the beginning compared to Figure 4a shows a loss of around 5.8 to 6 at the lowest but with a quicker drop in the first 500 steps.

4.4.2 Energy Consumption Trends

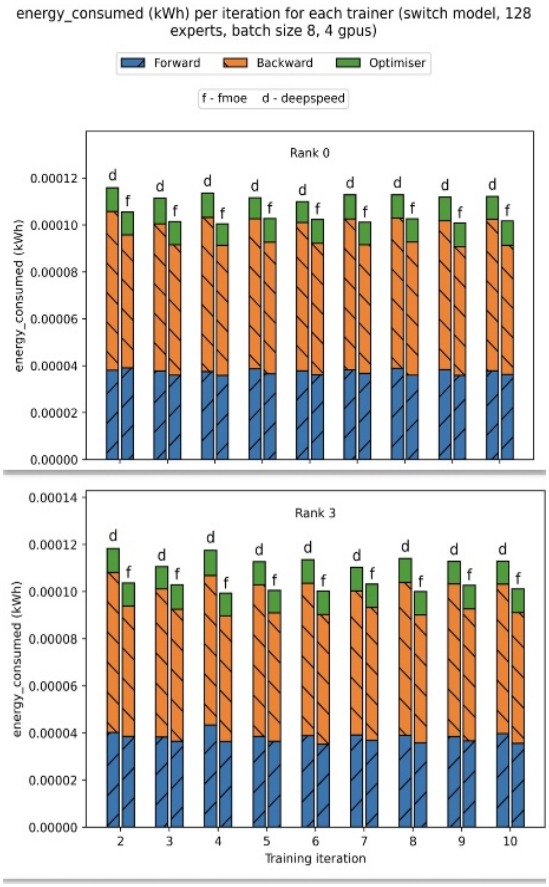
Note: Figures are cropped to show highlights and key details. See the Appendix for the full image. All GPU ranks exhibit similar trends in energy consumption and loss curves.

Early energy plots were also created, with much fewer iterations (10) and unstable parameters, to compare the energy consumption of the substeps (forward, backwards, and optimizer steps) in each

training framework and model. These plots were also the deciding factor for the decisions made in parameter choice.



(a) Switch-Transformers, BS=2, GPU=4



(b) Switch-Transformers, BS=8, GPU=4

Figure 5: Switch-Transformers: Deepspeed (d) vs. FMoE (f) training frameworks, Batch size 2 and 8 comparison

In all configurations, it can be seen that the backwards pass is the phase that consumes the most energy and that varies between training frameworks. For example, in Figure 5, while the difference between Deepspeed and FMoE is small, it is obvious that it lies in the backward pass. This is most probably caused by the way FMoE is designed to have a faster and non-padded approach.

4.5 Experiment Runs

Note: Figures are cropped to show highlights and key details. See the Appendix for the full image. All GPU ranks exhibit similar trends in energy consumption and loss curves.

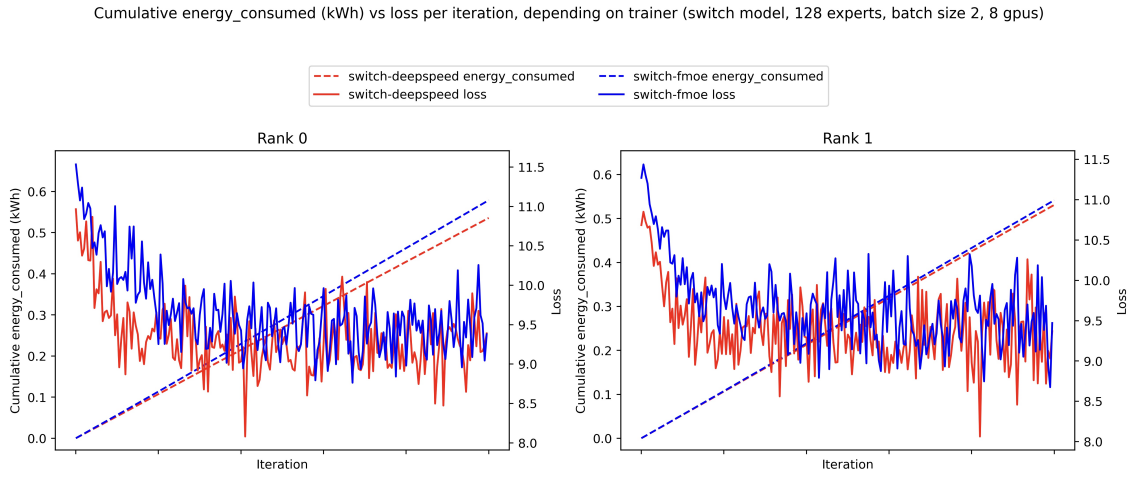
4.5.1 Switch-Transformers

For the project experiment, the workload run for Switch-Transformers was pre-training using masked language modelling (MLM). The model was evaluated across different batch sizes and two training frameworks: DeepSpeed and FMoE. The goal is to observe both energy consumption and training loss behaviour, and to analyze how batch size and training framework framework impact these metrics.

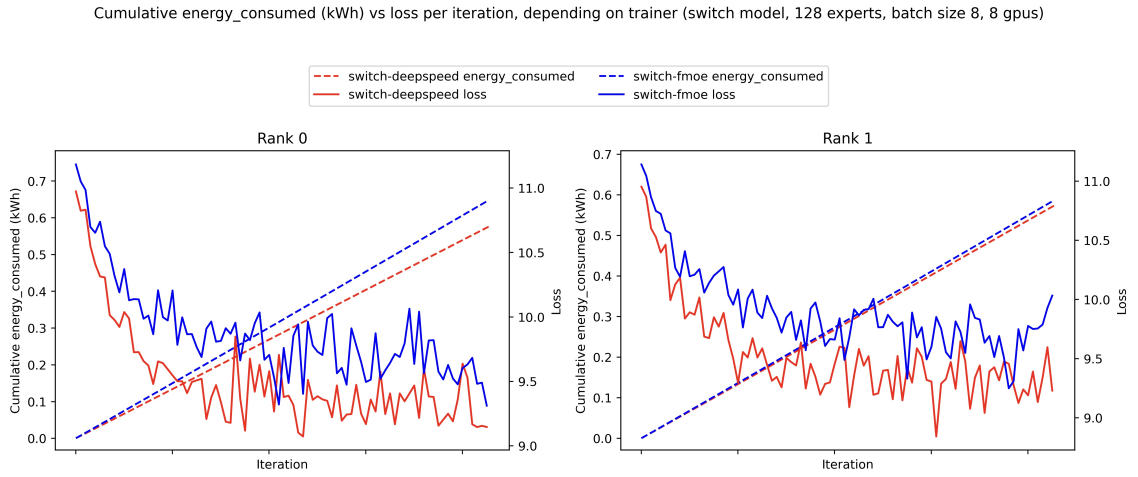
As shown in Figure 6a, at a batch size of 2, energy consumption remains relatively close between the two frameworks. DeepSpeed consumes approximately 0.54 kWh per GPU, while FMoE shows a slightly higher consumption of around 0.58 kWh on rank 0 and 0.56 kWh on other ranks. This minor difference might stem from additional communication overhead in the FMoE routing mechanism. Both curves grow linearly, consistent with the cumulative nature of energy consumption. The corresponding loss curve is noisy and shows frequent fluctuations, suggesting possible instability from the high learning

rate or small batch size. This curve was also not smoothed, and data was plotted for every few points instead of all for a less cluttered picture, which will be evened out for future plotting. While DeepSpeed shows a slightly lower average loss compared to FMoE, both exhibit similar trends.

On the other hand, at batch size 8 as shown in Figure 6b, there is an observable improvement in loss stability for both training frameworks. The energy consumption values per iteration remain close to BS=2 (around 0.58 kWh for DeepSpeed and 0.62/0.6 kWh for FMoE on rank 0/others), but since batch size 8 processes 4x more samples per iteration, a fair comparison must adjust for total samples seen. When normalized, DeepSpeed uses approximately 0.2 kWh and FMoE about 0.22 kWh per GPU for the same number of samples, approximately 3x less energy with around the same loss values. Furthermore, the DeepSpeed loss curve is consistently lower than FMoE's at this batch size, which may indicate better optimization dynamics under larger batches.



(a) Switch-Transformers, BS=2, Energy consumption vs. loss per iteration



(b) Switch-Transformers, BS=8, Energy consumption vs. loss per iteration

Figure 6: Switch-Transformers Energy consumption vs. loss: Training framework and batch size (BS) comparison

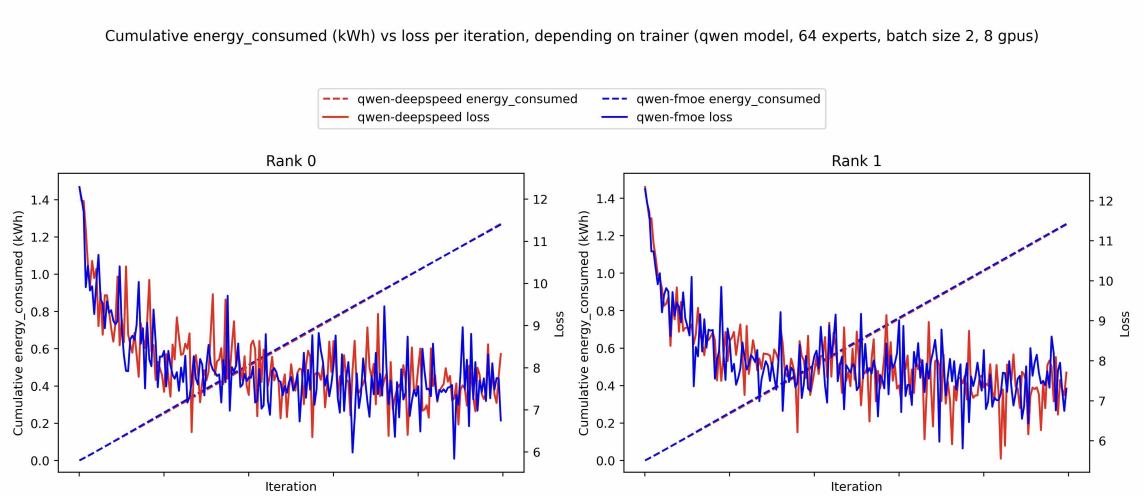
4.5.2 Qwen3-MoE

Similarly, the Qwen3-MoE model was run with the workload set for pre-training using causal language modelling instead since it does not support MLM. The model was evaluated with different batch sizes using DeepSpeed and FMoE. The larger top-k (8 for Qwen and 1 for Switch-Transformers) parameter and denser architecture of Qwen3-MoE models contribute to higher absolute energy use.

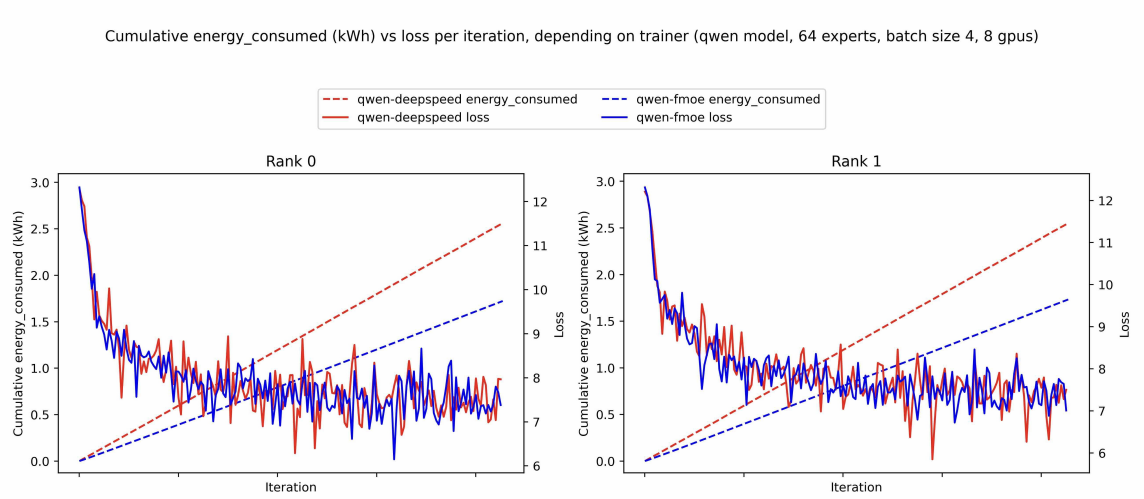
At batch size 2, energy consumption for DeepSpeed is around 1.3 kWh per GPU, while FMoE remains

slightly lower at 1.2 kWh, varying slightly between GPU ranks (with similar trends for all GPUs as the ones in Figure 7a). These results oppose the behaviour observed in Switch-Transformers, where FMoE showed higher energy per iteration in early stages. The training loss, however, is very similar across both frameworks, and both follow a consistent downward trend. Compared to Switch-Transformers, the Qwen3-MoE loss curves appear more stable, possibly due to the lower value and architectural robustness.

At batch size 4, the energy gap becomes more pronounced. FMoE consumes approximately 1.6 kWh over the same number of iterations, while DeepSpeed reaches 2.6 kWh. When normalized to the number of samples seen (halfway through the graph), the effective energy use becomes 1.0 kWh for FMoE and 1.6 kWh for DeepSpeed. One hypothesis is that the router mechanism or expert selection strategy used for the FMoE implementation with Qwen3-MoE is not the correct one for comparison, since the router loss is much simpler, with the AUX loss improperly computed. The AUX loss is usually used to reduce router imbalances and thus could be causing issues - an area for further investigation. Despite this discrepancy, the loss curves remain similar across both frameworks and again show better stability at the larger batch size.



(a) Qwen3-MoE, BS=2, Energy consumption vs. loss per iteration



(b) Qwen3-MoE, BS=4, Energy consumption vs. loss per iteration

Figure 7: Qwen3-MoE Energy consumption vs. loss: Training framework and batch size (BS) comparison

4.5.3 Overall Comparisons

Across both models, a few consistent trends were observed:

- **Batch size** significantly affects training stability, with larger batches reducing loss volatility.

- **Energy efficiency per sample** is generally better with higher batch sizes, despite higher per-iteration costs. The batch size affects the speed at which the model sees the same number of samples, reducing the duration of the training and leading to less energy consumption.
- **DeepSpeed vs. FMoE**: DeepSpeed tends to use more energy than FMoE at larger batch sizes for Qwen3-MoE. This may be due to differences in routing strategy or expert parallelism overhead. On the other hand, both training frameworks have very close energy consumption values for Switch-Transformers, with FMoE using slightly more on rank 0, possibly due to some communication overhead.
- **Model architecture**: Qwen3-MoE models consume more energy than Switch-Transformers under similar configurations, likely due to the higher default top-k parameter, the denser parameter layout and deeper layers.

While absolute numbers vary, the key insight is that MoE-based scaling provides a meaningful tradeoff space between training efficiency, energy consumption, and model performance. Future work should investigate router optimizations and energy-aware training schemes for deeper insights.

5 Future Work

Since the project is not yet completed, continued experiments are still in progress, including explorations on the following topics, as well as any others not mentioned yet:

- **Throttling and parallelizing**: Exploring the possibilities of throttling the GPUs in order to conserve energy in some stages of the training and communication steps. Also exploring the possibilities of running some things in parallel CUDA streams (Eg, communication).
- **Throttling vs. Speed** Comparing the energy consumption between throttling and simply running training faster. Exploring the tradeoff of bigger batch sizes with less training time and the accuracy of the training produced.
- **More iterations**: Running experiments with many more iterations since real cases of machine learning training usually run for much longer.
- **Custom Training framework**: Exploring the possibility of building a training framework that contains the optimizations from many other training frameworks.

6 Conclusion

In conclusion, the experiments with Switch-Transformers and Qwen3-MoE highlight important trade-offs in training efficiency and model design for sparsely activated models. Both models were observed to benefit from the Mixture-of-Experts (MoE) architecture in reducing per-token compute through expert routing. There are notable differences in energy consumption and training time, particularly during the backward pass, where the difference between the FastMoE training framework and the DeepSpeed implementations differ in their overhead.

Other configurations investigated, such as expert routing strategies, batch size, and GPU count, affect overall system performance. These factors are critical when scaling up models to larger sizes and training or deploying in real-world distributed settings.

Finally, as mentioned previously, much more work, scaling, and experimentation are required to draw definitive conclusions and unlock the full potential of energy-efficient distributed training. Future directions include benchmarking at larger model scales, exploring hybrid training framework mechanisms, and exploring different mechanisms like throttling to reduce communication and other overheads.

7 Code

Note: the code can be found in the following repository "https://github.com/zu-greta/energy-efficiency-ml" as well as the upstream repository it was forked from "https://github.com/OMichaud0/msc-research-exploration/tree/qwen"

References

[1] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. allenai/c4 dataset card. `https://huggingface.co/datasets/allenai/c4`.

[3] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[4] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale, 2022.

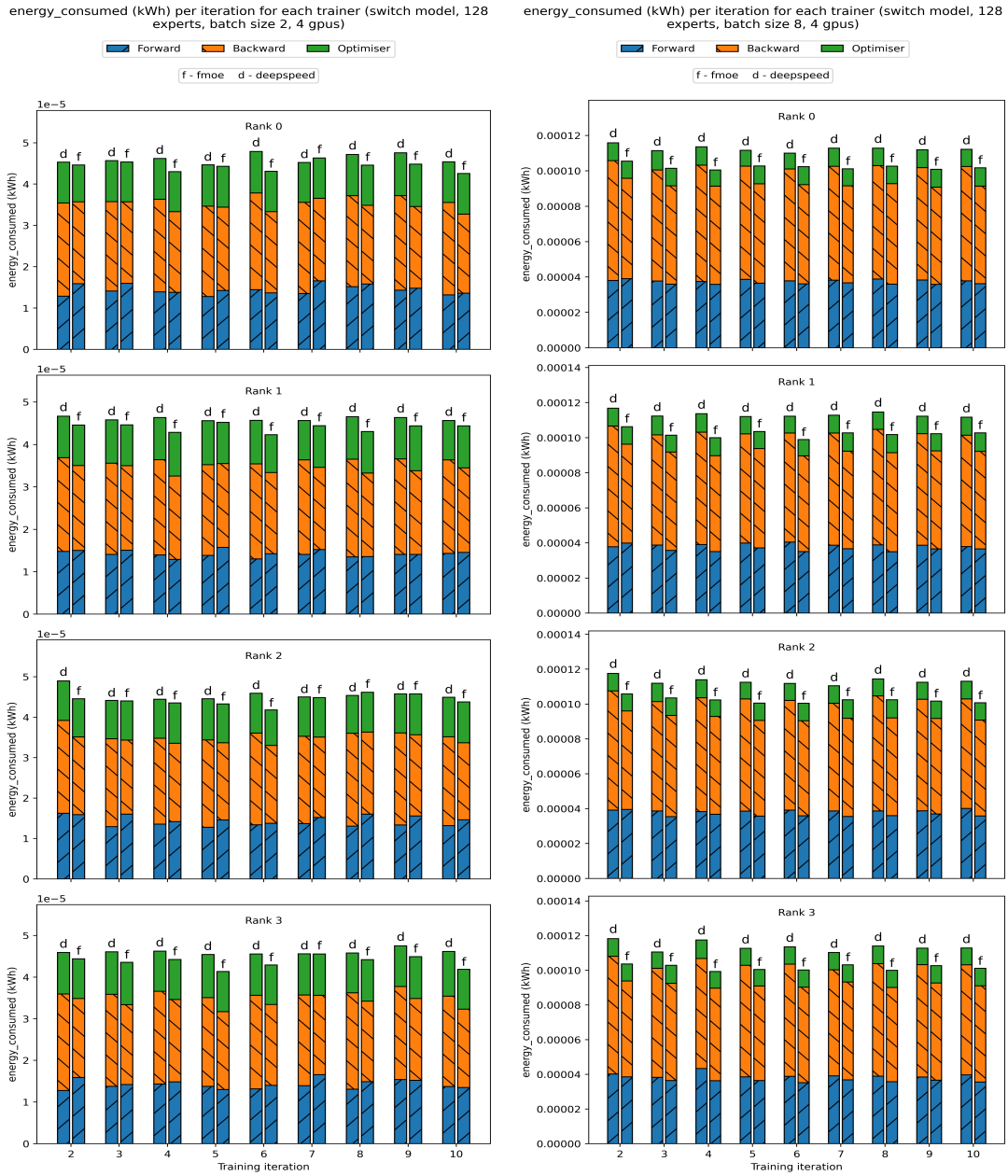
[5] Omar Sanseviero, Lewis Tunstall, Philipp Schmid, Sourab Mangrulkar, Younes Belkada, and Pedro Cuenca. Mixture of experts explained, 2023.

[6] Victor Schmidt, Kamal Goyal, Aditya Joshi, Boris Feld, Liam Conell, Nikolas Laskaris, Ziyao Wang, Doug Blank, Jonathan Wilson, Sorelle Friedler, and Sasha Luccioni. Codecarbon: Estimate and track carbon emissions from machine learning computing, 2021.

[7] Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, Oliver Curtis, Grigori Fursin, Miro Hodak, Hiwot Kassa, Anton Lokhmotov, Dejan Miskovic, Yuechao Pan, Manu Prasad Manmathan, Liz Raymond, Tom St. John, Arjun Suresh, Rowan Taubitz, Sean Zhan, Scott Wasson, David Kanter, and Vijay Janapa Reddi. Mlperf power: Benchmarking the energy efficiency of machine learning systems from microwatts to megawatts for sustainable ai, 2025.

[8] Xiaorong Wang, Clara Na, Emma Strubell, Sorelle Friedler, and Sasha Luccioni. Energy and carbon considerations of fine-tuning bert. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, page 9058–9069. Association for Computational Linguistics, 2023.

8 Appendix



(a) Switch-Transformers, BS=2, GPU=4

(b) Switch-Transformers, BS=8, GPU=4

Figure 8: Switch-Transformers: Deepspeed (d) vs. FMoE (f) training frameworks, Batch size 2 and 8 comparison

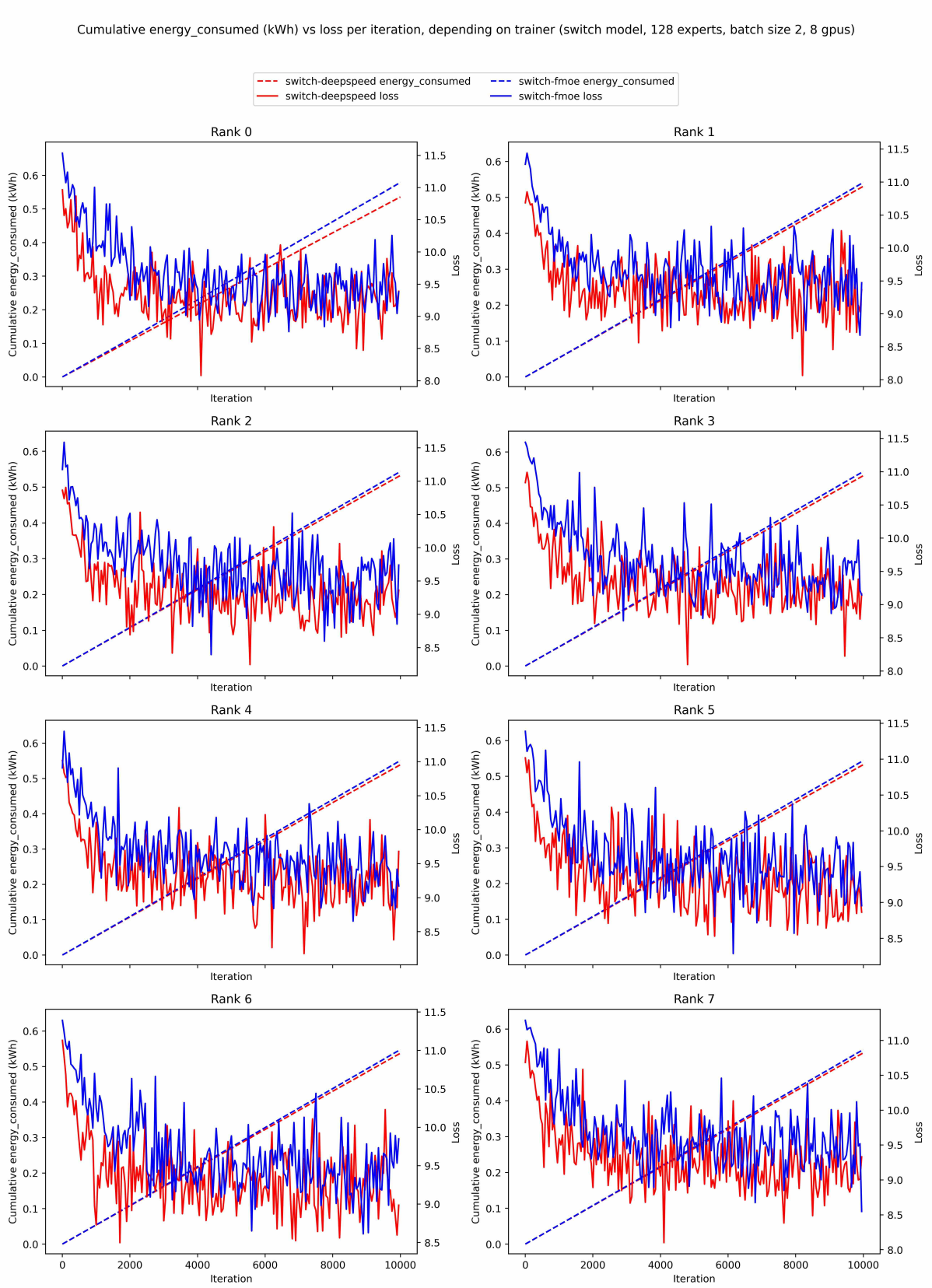


Figure 9: Switch-Transformers, BS=2, Energy consumption vs. loss per iteration

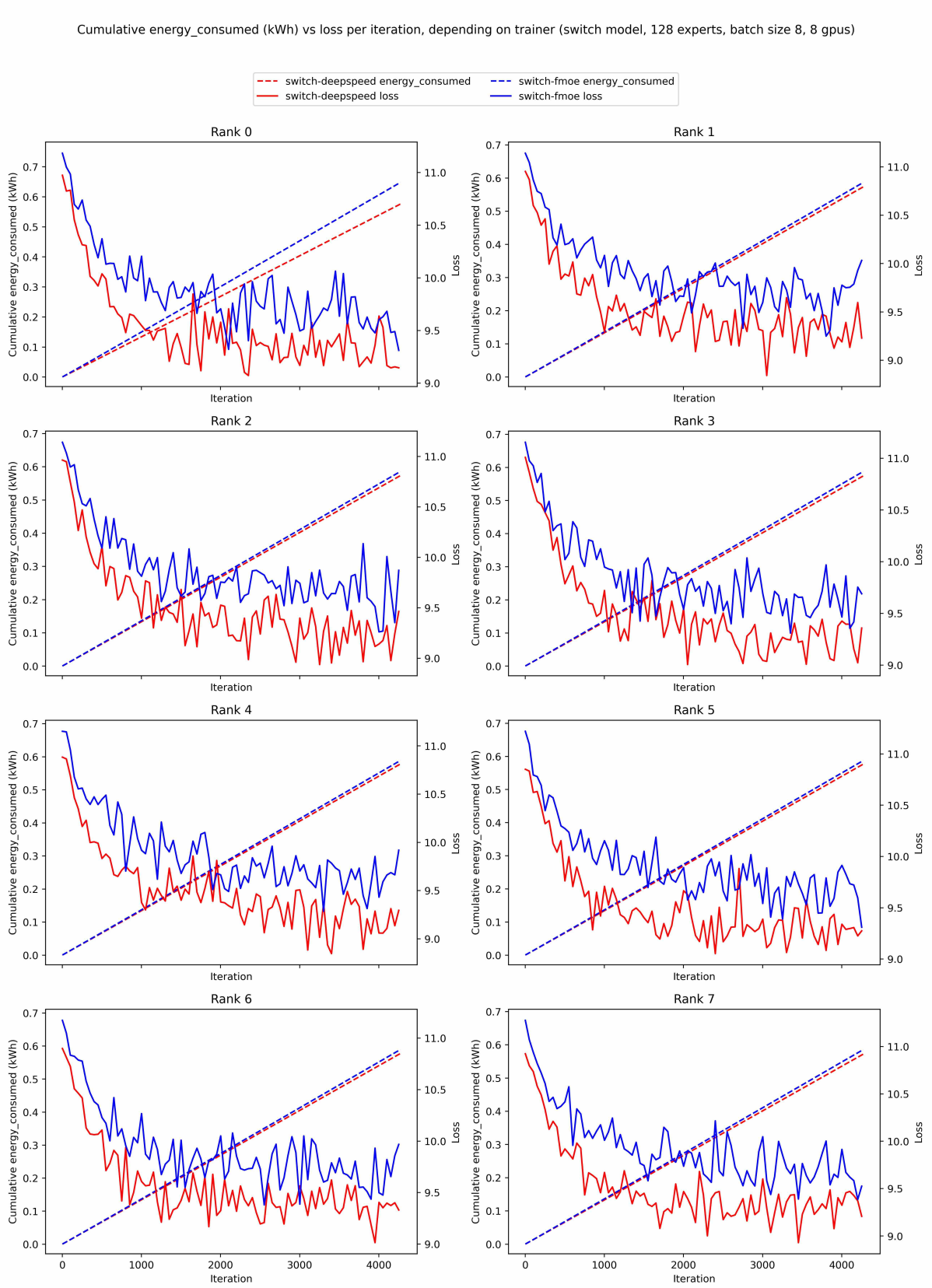


Figure 10: Switch-Transformers, BS=8, Energy consumption vs. loss per iteration

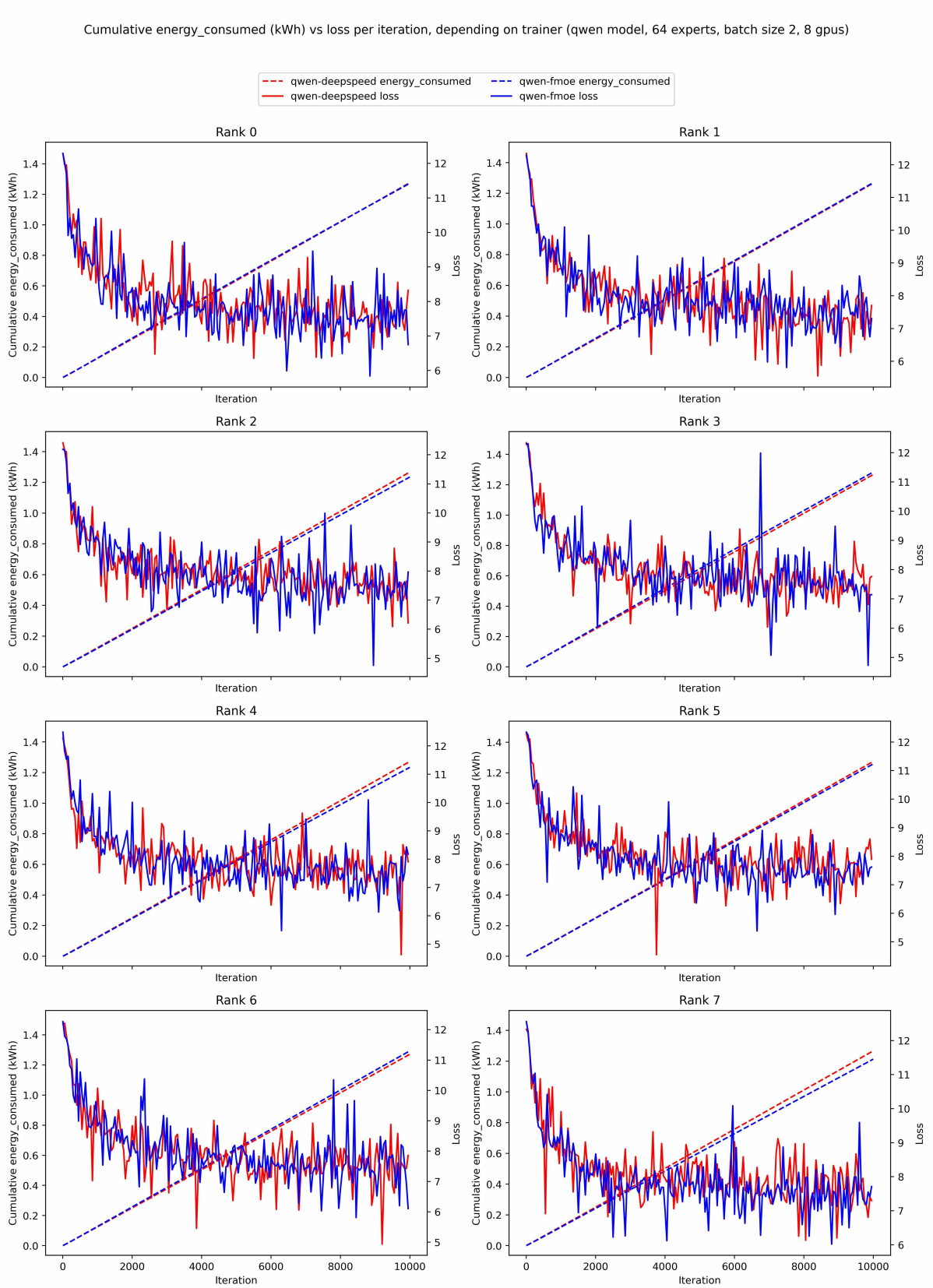


Figure 11: Qwen3-MoE, BS=2, Energy consumption vs. loss per iteration

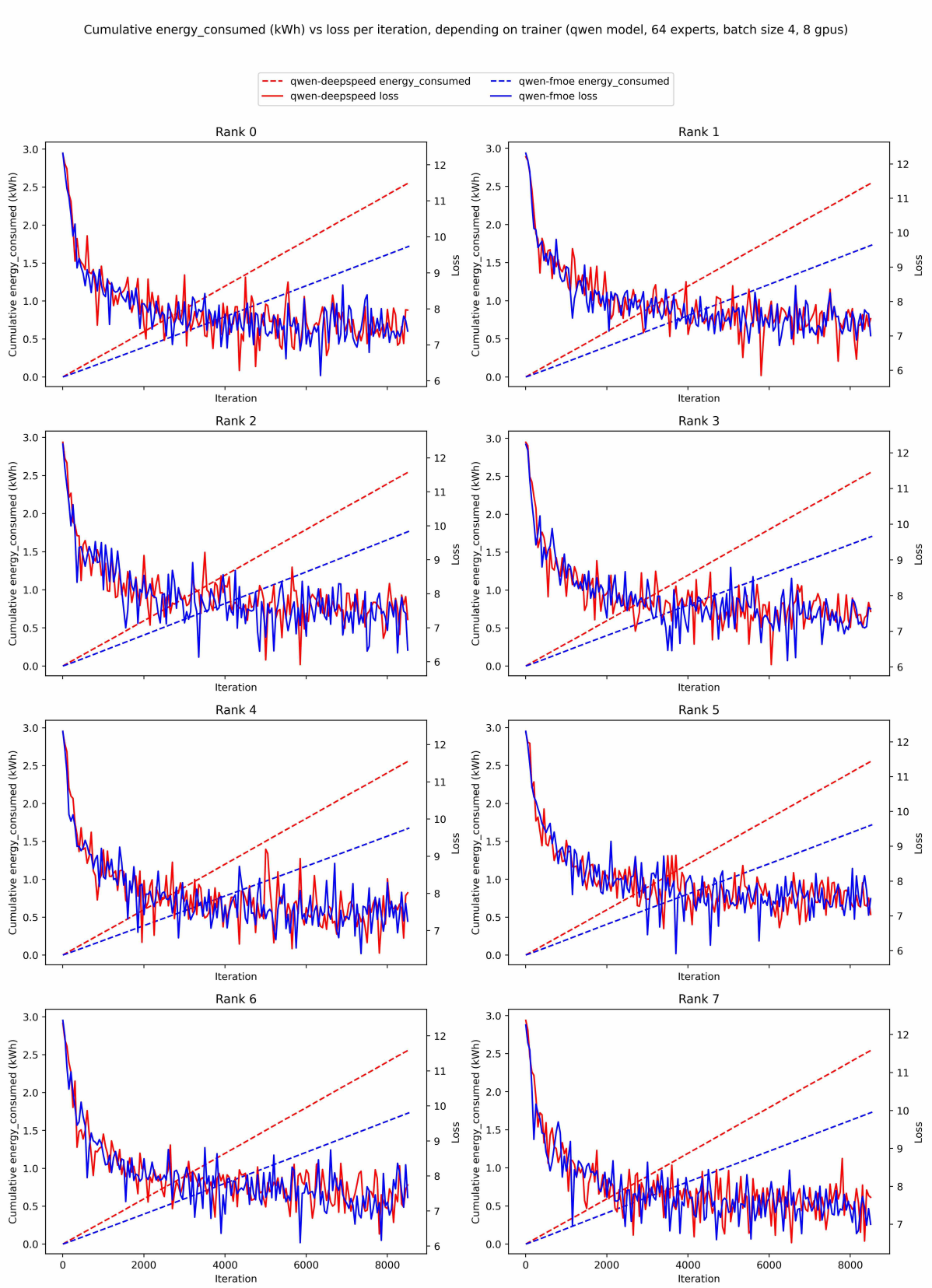


Figure 12: Qwen3-MoE, BS=4, Energy consumption vs. loss per iteration