Energy-Aware Decentralized Federated Learning

David Pidugu* Martin Strauss†

August, 2025

Abstract

In the explosion of Artificial Intelligence in the past few years, one concern that goes noted is energy use. We find that training compute for AI has grown at an extraordinary rate, nearly doubling roughly every 6 months in the deep learning era [SHS+22]. This exponential growth translates into dramatically higher energy demands. Being Energy Aware is crucial because unchecked scaling could drive large electricity use, higher carbon emissions, and strain on infrastructure. Conscious design choices such as efficient architectures, hardware, and training practices are therefore essential to make AI development sustainable while still enabling progress. We will explore methods in both analyzing and designing algorithms in the realm of Machine Learning in an energy aware methodology, specifically an energy intensive algorithm called Decentralized Federated Learning

1. Introduction

The rapid growth of artificial intelligence has brought both innovation and concern. One critical concern is the energy footprint of machine learning. Traditional centralized learning approaches and large scale training of models consume substantial energy, contributing to environmental and economic costs. Recent work in federated and decentralized learning offers an alternative approach, specifically distributing computation across multiple clients while being mindful of energy efficiency.

*Department of Electrical Engineering and Computer Science, University of Michigan
†Department of Mathematics, University of Michigan

This paper explores energy-aware decentralized machine learning, focusing on how algorithmic design, communication protocols, and model optimization can reduce energy consumption without severely compromising performance.

2. Preliminaries

2.1. Machine Learning Model

At its core, machine learning can be viewed as a process of constructing a stochastic model, which is a mathematical representation that captures relationships in data by relying on probability and randomness. Unlike deterministic algorithms, which produce the same output given identical input, a stochastic model incorporates uncertainty and variability. In practice, this is what allows machine learning systems to handle noisy, incomplete, or diverse real world data. Within training, stochastic models are updated through iterative optimization (e.g., stochastic gradient descent), where small random samples of data are used to approximate broader trends. This makes them computationally tractable and effective, but also energy-intensive: each random update requires floating point operations, memory access, and communication overhead. In this sense, the "black box" of machine learning is the stochastic model, which encodes knowledge not as fixed rules, but as probabilistic weights learned through repeated computation. In this paper, we would represent learning in this was as the following in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent

1: **for** $e \in [E]$ **do**
2: $\quad \xi \leftarrow$ mini-batch of samples from D
3: $\quad x \leftarrow x - \eta \nabla f(x, \xi)$
4: **end for**

Here, we represent E as the total epochs, D as a data set, x as a model, η as the learning rate, and ∇f as the gradient of the loss function given the model and a mini batch of samples.

2.2. Federated Learning

As machine learning has scaled, so too has the energy required to train these stochastic models. Traditional centralized learning architectures collect raw data from multiple sources into a single repository where training occurs. This

centralization simplifies coordination but introduces significant costs which includes large-scale data transfer, storage overhead, and high computational demand concentrated on powerful servers. The central server becomes both a bottleneck and a single point of failure, while also raising privacy concerns since user data must leave local devices.

Federated Learning emerged as an alternative paradigm. Instead of centralizing raw data, FL keeps data local to client devices while sharing only model updates. In a centralized federated system, a central server still exists, but it orchestrates training by sending an initial model to clients, aggregating their updates, and redistributing the improved model. This approach reduces raw data movement and enhances privacy, but the server remains a coordination bottleneck.

In contrast, **decentralized federated learning** removes the central server entirely. Clients communicate directly with one another, exchanging model parameters with their neighbors in a peer to peer fashion. Each client integrates both its local updates and the updates received from connected peers, gradually converging toward a global model. This decentralized architecture avoids single points of failure, reduces communication overhead in large-scale systems, and distributes the energy load more evenly across the network. Importantly, it also opens the door to energy-aware design, since both training and communication strategies can be tailored to reduce redundant computations while still achieving model accuracy.

We can describe the high level algorithm of decentralized federated learning as follows: For each round and For each node:

1. Train on a local data set
2. Communicate Model parameters to Neighbors
3. Learn through weighted aggregation of neighbor parameters

2.3. Dataset

In this paper we will use the Federated Extended Modified National Institute of Standards and Technology (FEMNIST) data set. This is a collection of handwritten characters (Upper and lower case alphabet plus the numbers 0-9) written by different clients [CLM+18]. Essentially, each client has samples pertaining to each character written in a different handwriting style. This is

a good benchmark for federated learning as each node can function for one or more clients. In this, each node learns specifically the writing styles given in a data set while still maintaining a global identification scheme given learning from neighbors. We will be using this data set to gauge the algorithms that we will be designing and the accuracy given an unknown global test set with each individual node's training set consisting of one or more unique clients.

3. Energy Aware Learning Algorithm Design

3.1. Decentralized Learning

We will formalize the idea of Decentralized Learning in this subsection in the same way of [DDG+24]. The setting consists of n nodes that collaboratively learn a data set. Each node has access only to its local data set drawn from a distribution D_i for each node i, which may be different per node.

3.1.1. Training

In the training process, similar to most learning algorithms, the goal is to find parameters of a model x_i that performs well per node. However, we also require that each model x_i performs well on the entire data set $D = \bigcup_{i\in[n]} D_i$ which we can quanitfy by minimizing the average loss function:

$$\min_{x_i} f(x_i) = \frac{1}{n} \sum_{i\in[n]} \mathbb{E}_{\xi \leftarrow D_i}[F_i(x_i, \xi)] := \frac{1}{n} \sum_{i\in[n]} f_i(x_i)$$

Here, we can call f_i the local objective function of node i and $F_i(x, \xi)$ as the loss of the model x_i on the sample $\xi \in D_i$.

3.1.2. Communication

We state that the communication topology is represented as an undirected graph $G = (V, E)$, where V is the set of all nodes and $(i, j) \in E$ represents a communication channel between nodes i and j. In this paper we will not rigorously analyze the communication step as analyzing energy of communication ultimately falls into the hardware and network category. Thus, we will assume a polynomial efficient algorithm that cannot be modified that transports information of each model parameter's of over a communication channel. We will denote this function as $\textsc{com}(x_i, i, j)$.

3.1.3. Collaboration

Once a node receives the parameters of its neighbors in a graph (this is done after it sends out its own parameters to its neighbors), it will perform a weighted average of its local models with its neighbors. To perform this, we will use a symmetric matrix $W \in \mathbb{R}^{n \times n}$ that encodes the weights per each neighbor. We also hope for the matrix to be doubly stochastic which we will define as $\sum_{j \in V} W_{i,j} = \sum_{i \in V} W_{i,j} = 1$. Using these conditions, we can ensure convergence to a single point in the overall loss function [LZZ+17].

To accomplish this, we use Metropolis-Hastings weights [XB04] based on our graph $G = (V, E)$. Using this, we define W as

$$W = \begin{cases} \frac{1}{\max\{\deg(i), \deg(j)\}+1} & \text{if } i \neq j \text{ and } (i,j) \in E \\ 1 - \sum_{i \neq j} W_{i,j} & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

3.2. D-PSGD

3.2.1. Algorithm

Putting together each step, we can create an algorithm that for T rounds, performs decentralized learning for each node. Specifically, the standard algorithm named Decentralized Parallel Stochastic Gradient Descent (D-PSGD) algorithm [LZZ+17] as seen in Algorithm 2.

Algorithm 2 D-PSGD, to be run in each Node i

initialize base model x_i
for $t \in [T]$ **do**
    $x_i^p \leftarrow x_i$
    for $e \in [E]$ **do**
        $\xi_i \leftarrow$ mini-batch of samples from D_i
        $x_i^p \leftarrow x_i^p - \eta \nabla f(x_i^p, \xi_i)$
    end for
    for $j \in$ NEIGHBORS(i) **do**
        COM(x_i^p, i, j)
    end for
    Receive neighbor models x_j^p from Neighbors
    $x_i \leftarrow \sum_{j \in V} W_{i,j} x_j^p$
end for
return x_i

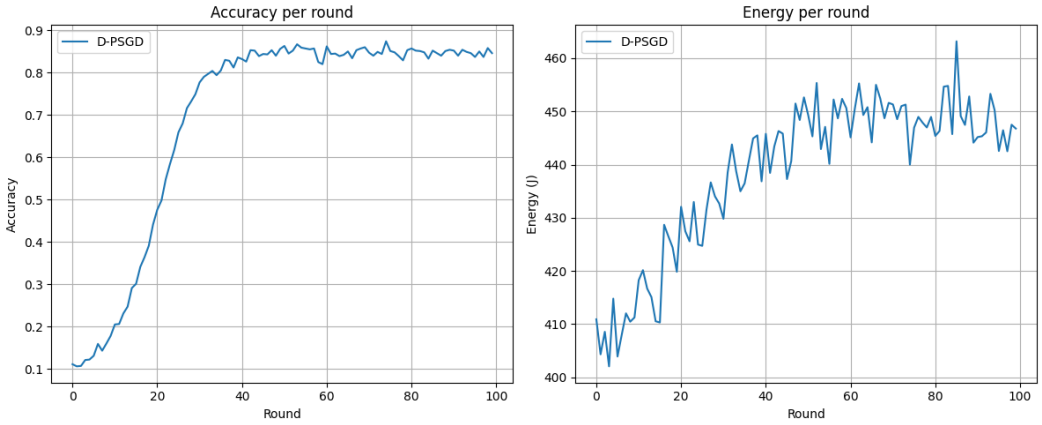

Figure 3.1: D-PSGD Accuracy and Energy plot

Algorithm 1 works in parallel by each node. We see that in Algorithm 1, that lines 4-6 represents the training process. In lines 8-11 we send to and receive the model parameters from neighbors. Finally in line 12 we execute the collaboration process of learning from other neighbors.

This will be our base algorithm that we hope to improve.

3.2.2. Energy Analysis

In this paper we hope to analyze the energy consumption of the entire process (other than communication). As such, there are many variables in the real world which contributes to overall energy, thus implementing a "rigorous" definition of energy consumption per algorithm will not be implemented here.

Instead, we will use the data set as described in section 2.3 to measure energy. To implement the algorithm, we employed the use of PyTorch [PGM+19] to create our models. To measure energy, we utilized a python library named pyJoules [UI21]. As this library simply measures the energy consumption of the comptuter as a whole, as little processes as possible was run in the background and a "baseline" energy was recorded such that the data could be normalized.

In training we utilized 100 nodes in a 6-regular graph, each with a different user from our FEMNIST data set. We used parameters for learning of a batchsize of 32, learning rate of 0.01, 3 local epochs, and 100 total rounds. From this we can analyze a "base" result that we hope to improve on energy[1].

[1]This paper aims to only improve D-PSGD on energy use. Often this means that we sacrifice energy in favor of lower Energy consumption

As communication is not measured in this paper, we simulate multiple devices through a Node data structure. Each Node object consists of the required data to use D-PSGD and later algorithms in the paper. Thus, the communication protocol is simply done by simulating the objects communicating information to each other.

The data collected can be found in Figure 3.1. Note that due to the dynamic background processes and a multitude of other factors, the energy reading is not consistent and will be within a bound of error.

3.3. SkipTrain

3.3.1. Algorithm

An improvement on D-PSGD has been researched in [DDG+24] wiht the algorithm SkipTrain[2]. We define the algorithm in Algorithm 3.

Algorithm 3 SkipTrain, to be run in each Node i

1: initialize base model x_i
2: **for** $t \in [T]$ **do**
3: $\quad x_i^p \leftarrow x_i$
4: $\quad$ **if** $t \mod (\Gamma_{\text{train}} + \Gamma_{\text{sync}}) < \Gamma_{\text{train}}$ **then**
5: $\quad\quad$ **for** $e \in [E]$ **do**
6: $\quad\quad\quad \xi_i \leftarrow$ mini-batch of samples from D_i
7: $\quad\quad\quad x_i^p \leftarrow x_i^p - \eta \nabla f(x_i^p, \xi_i)$
8: $\quad\quad$ **end for**
9: $\quad$ **end if**
10: $\quad$ **for** $j \in$ NEIGHBORS(i) **do**
11: $\quad\quad$ COM(x_i^p, i, j)
12: $\quad$ **end for**
13: $\quad$ Receive neighbor models x_j^p from Neighbors
14: $\quad x_i \leftarrow \sum_{j \in V} W_{i,j} x_j^p$
15: **end for**
16: **return** x_i

We find that the algorithm is almost identical to D-PSGD. The first difference is the introduction of the variables Γ_{train} and Γ_{sync}. Using the only addition of SkipTrain being in line 4 of Algorithm 3, we see that the model

[2]The general version of the algorithm is SkipTrain-Constrained, but this works better in a device-specific context. As we simulate a network rather than use devices, we will not be using SkipTrain-Constrained.

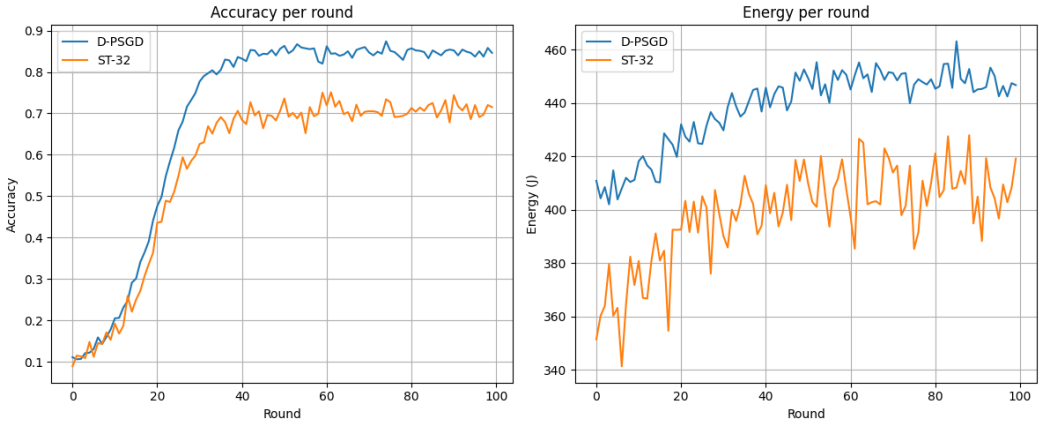

Figure 3.2: SkipTrain Accuracy and Energy plot

iterates through Γ_{train} train rounds where training is being performed along with communication and learning, followed by Γ_{sync} rounds where the model is not training and rather just communicating and learning. Hence, the overall idea of the algorithm is the fine-tune training and syncing rounds to sacrifice some accuracy for lower energy use. We can state that the energy wil be lower as energy consumption during Decentralized Learning is highly concentrated during the trainining step [PGL+21].

3.3.2. Energy Analysis

We will use the same scenario as described in sectin 3.2.2. However, we hope to fine-tune Γ_{train} and Γ_{sync} to minimize accuracy loss and maximize Energy saving. We found that 3 training rounds and 2 syncing rounds was optimal for our data set and scenario and this will be the parameters for Γ_{train} and Γ_{sync} for the paper.

Looking at figure 3.2 we find our result of our simulation in comparison to the previous algorithm. However, we note the dramatic accuracy loss. This is to be expected due to the lower amount of total training rounds (As $\Gamma_{\text{train}} = 3$ and $\Gamma_{\text{sync}} = 2$, out of the 100 rounds only 60 rounds will be spent training). The reduced and intermittent training rounds are especially relevent in FEMNIST as each data set may differ by a large amount per client[3]. However, we notice that in the accuracy loss, we achieve a high energy saving compared to our base algorithm.

[3]This large accuracy loss is not seen in the regular MNIST dataset where handwriting styles do not differ.

3.4. Priority Learning

3.4.1. Algorithm

As SkipTrain reduces the amount of training rounds, we hope to also implement an algorithm that may reduce the number of aggregation steps in collaboration/learning. To accomplish this we perform an algorithm labeled Priority Learning which will be implemented in our SkipTrain algorithm to create SkipTrain-PL. We define this algorithm in Algorithm 4

Algorithm 4 SkipTrain-PL, to be run in each Node i

initialize base model x_i
for $t \in [T]$ **do**
  $x_i^p \leftarrow x_i$
  if $t \mod (\Gamma_{\text{train}} + \Gamma_{\text{sync}}) < \Gamma_{\text{train}}$ **then**
    for $e \in [E]$ **do**
      $\xi_i \leftarrow$ mini-batch of samples from D_i
      $x_i^p \leftarrow x_i^p - \eta \nabla f(x_i^p, \xi_i)$
    end for
  end if
  for $j \in \text{NEIGHBORS}(i)$ **do**
    $\text{COM}(x_i^p, i, j)$
  end for
  Receive neighbor models x_j^p from Neighbors
  Let $\mathcal{N}_i^{\text{top}} \subseteq \{x_j^p \mid j \in \text{NEIGHBORS}(i)\}$ be the top-N neighbors by α_j
  $x_i \leftarrow \frac{\sum_{j \in \mathcal{N}_i^{\text{top}}} W_{i,j} x_j^p}{\sum_{k \in \mathcal{N}_i^{\text{top}}} W_{i,k}}$
end for
return x_i

In SkipTrain-PL we introduce the new variable of N. From N we get a new parameter of choosing only a select "best" nodes to learn of. We do this by comparing α_j or the accuracy of the model that is done through evaluation[4]. We call this set $\mathcal{N}_i^{\text{top}}$ and re-normalize our weight matrix W when aggregating our data. From this, while our main goal is done to reduce aggregation steps which in turn helps improve energy, we have a trade-off of re-normalizing our Weight Matrix. Thus, in turn, we expect that while the energy tradeoff

[4]This step assumes that one regularly evaluates a model and stores the accuracy during the learning process

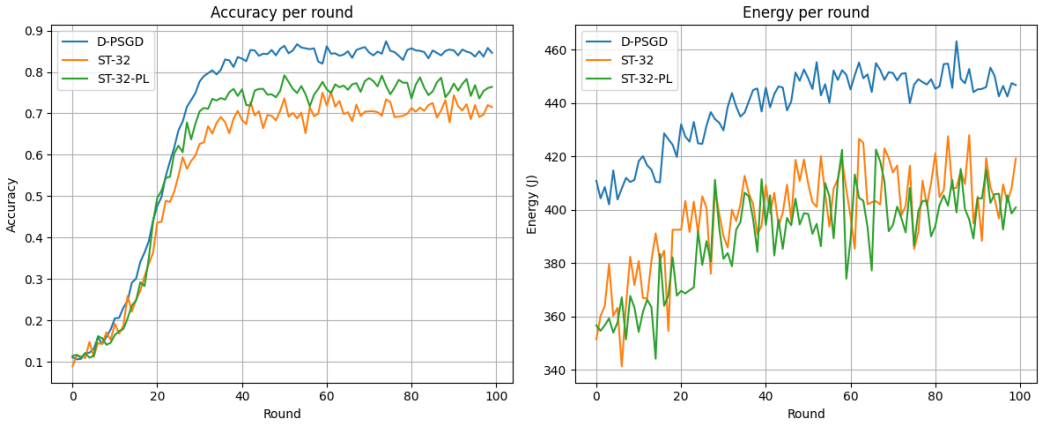

Figure 3.3: SkipTrain-PL Accuracy and Energy plot

should be roughly equal given an optimal choice of the value N, the accuracy of SkipTrain may be increased due to a more intelligent collaboration process.

3.4.2. Energy Analysis

We proceed with the same parameters of our variables and setting as in section 3.3.2. We found that in our setting of a 6-regular graph, that a choice of $N = 2$ was optimal in increasing accuracy with the least energy trade off. We also see that determining α_j did not require extra computation as we through this paper were evaluating the model every round to properly evaluate the accuracy and energy per round. We find our resulting data in figure 3.3.

Comparing SkipTrain-PL with D-PSGD and SkipTrain, we see that there is still an accuracy fall from D-PSGD, however a better accuracy than SkipTrain alone while still obtaining the large energy savings that is found with SkipTrain. We note that there may be much larger accuracy and energy consumption improvements for very dense graphs as aggregation may be a more intensive process being done with a large number of neighbors.

4. Energy Aware Inference Design

While learning may be performed a fixed number of times, inference (using the finalized model to predict) has no limit in usage. Especially in a modern day setting where using machine learning models is becoming increasingly common, an energy-efficent inference model will be useful for after the training.

4.1. Low-Rank Approximation

The clearest way to reduce energy use when using the model to predict data is simply to have a smaller model. As our model is generally represented by matrices describing the parameters of the network, we may try employing Low-Rank Approximation to help achieve an identical smaller model. As we are in the inference stage where usage cannot be bounded, overhead in reducing network size is not of importance. Hence, we can perform Algorithm 5 on each parameter matrix of the model given a value k.

Algorithm 5 Low-Rank Approximation

1: $[U, \Sigma, V^\top] \leftarrow \text{SVD}(A)$ ▷ SVD is the Single Value decomposition
2: $U_k \leftarrow$ first k columns of U
3: $\Sigma_k \leftarrow$ first $k \times k$ block of Σ
4: $V_k \leftarrow$ first k rows of $V^\top$
5: $A_k \leftarrow U_k \Sigma_k V_k$
6: **return** A_k

We were not able to get notable energy savings with decreasing k without sacrificng much accuracy as our model is "too small". Hence, individual weights carry large amount of information that cannot be simply lost. Hence, we note that this algorithm may be more beneficial for large networks where individual weights may be negligible, such that it may be approximated with smaller matrices.

5. Further Research

While this paper explores three such algorithms that deal with Training and Collaboration, we believe there may be many more algorithms that can be utilized to reduce energy consumption with a lower accuracy trade off. Potential improvements include Pruning of the model parameters, dynamic training/sync rounds (determining whether to be a sync or training round more intelligently), as well as dealing in a lower level with optimizations such as in cache or FLOPS. This paper also only analyzed a CNN with FEMNIST, but especially with the explosion of LLM's in everyday use, one may try to improve energy efficiency in a language context. This paper was done using a simulation in python thus not allowing for much improvement in the communication step, thus if done in a real world setting this may be able to be optimized.

Acknowledgments

I would like to thank my mentor Professor Strauss and The Department of Mathematics Research Experience for Undergraduates (REU) program for the mentorship and sponsorship over this past summer.

References

[CLM+18] S. Caldas, P. LeGresley, S. Mahdavi, J. Konečný, H. B. McMahan, and A. Talwalkar. Leaf: a benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[DDG+24] A. Dhasade, P. Dini, E. Guerra, A.-M. Kermarrec, M. Miozzo, R. Pires, R. Sharma, and M. de Vos. Energy-aware decentralized learning with intermittent model training. *arXiv preprint arXiv:2407.01283*, 2024.

[LZZ+17] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems 30 (NeurIPS 2017),*

[PGL+21] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguía, D. Rothchild, D. So, M. Texier, and J. Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: an imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019),* 8024–8035, 2019.

[SHS+22] J. Sevilla, L. Heim, B. Stefan, D. Ho, T. Besiroglu, R. Thorvaldsen, and M. Hobbhahn. Compute trends across three eras of machine learning. *arXiv preprint arXiv:2202.05924*, 2022.

[UI21] S. research group (University of Lille and Inria). *pyJoules: A Python toolkit for measuring energy consumption of Python code*. urlhttps://pyjoules.readthedocs.io. Accessed on August 27, 2025. 2021.

[XB04] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1): 65–78, 2004.