SPARTA: Sparse Parameter Averaging for Reduced-bandwidth Training

Anonymous Authors[1]

Abstract

Training large language models (LLMs) at scale requires efficient training across multiple computational nodes. Traditional distributed training approaches synchronize data between nodes at each iteration on the scale of the full model size, resulting in high communication overhead. While recent advances in federated learning mitigate this by allowing infrequent synchronization, they still require transmitting the entire set of model gradients during updates. In this work, we propose sparse parameter averaging, where a small portion of the model parameters are averaged between workers at each step. SPARTA offers several advantages: it supports asynchronous updates of up to 10 steps without performance degradation, is resilient to network faults and is simple to implement. Our experiments demonstrate that SPARTA outperforms state-of-the-art federated learning methods such as DiLoCo in both bandwidth efficiency and practical robustness.

1. Introduction

Training large language models (LLMs) at scale necessitates distributing computation across large GPU clusters, where substantial volumes of data are exchanged continuously among devices. Conventional approaches, such as Distributed Data Parallelism (DDP) (Li et al., 2020; Shoeybi et al., 2019; Narayanan et al., 2021) mandate synchronizing either all model parameters or gradients between devices at every training step. This requirement imposes significant bandwidth demands and often becomes a bottleneck, particularly in environments with limited interconnect performance.

Federated learning mitigates communication overhead by synchronizing infrequently. The Federated Averaging (FedAvg) algorithm (McMahan et al., 2017) allows K models to train on a separate subset of data, then averages all models every H steps to synchronize them. This strategy lowers bandwidth usage by a factor of H while often maintaining comparable convergence rates as regular training. This has proven effective for training large language models (LLMs) as demonstrated by DiLoCo (Douillard et al., 2023; 2025). Yet, this approach still requires the entire model to be communicated during synchronization.

Additionally, FedAvg is mathematically equivalent to performing gradient descent on the synchronized (outer) gradients (e.g. Section 1 of (Reddi et al., 2020)). Based on this view, recent methods (Reddi et al., 2020; Douillard et al., 2023) have explored using more sophisticated outer-loop optimizers, rather than simply averaging the parameters. In this work, we argue against this approach: such methods increase algorithmic complexity and introduce more hyperparameters to tune. Furthermore, the infrequent nature of outer gradient updates is ill-suited to momentum-based methods, which typically require careful tuning or disabling altogether. Instead, we show that simple approaches such as SPARTA can outperform state-of-the-art methods including DiLoCo.

We present a novel algorithm called SPARTA, where only a small subset of model parameters is synchronized at each step. While FedAvg averages full models every H steps, SPARTA averages a sparse subset continuously. This approach is lightweight and simple to implement, enables more granular control over communication costs, and opens new avenues for robustness and flexibility not offered by traditional federated learning methods.

In Section 3, we provide a detailed description of the algorithm. SPARTA shares a proportion p of parameters between nodes, which can be done either synchronously or asynchronously.

Lastly, in Section 4, we demonstrate how SPARTA performs in practice through experimental validation. Empirically, we confirm that SPARTA matches FedAvg in performance when total bandwidth is held constant. We compare the quantitative performance of SPARTA with state-of-the-art methods and find that, despite its simplicity, it can outperform methods such as FedAvg and DiLoCo. Furthermore, we analyze weight correlation over time, showing that the trajectories remain more robust and consistent. We em-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

pirically demonstrate that this results in improved robustness and fault tolerance, withstanding up to 95% message loss. We show that SPARTA functions effectively with asynchronous/delayed communication of up to 10 steps with negligible performance loss. Finally, we show that our method continues to outperform competitive approaches in large-scale training of models with more than a billion parameters.

2. Background

Let θ denote the parameters of a model trained on dataset $\mathcal{D} = \{(x, y), \ldots\}$, where (x, y) are input-output pairs. In language modeling, for instance, inputs and outputs are often token embeddings (Vaswani et al., 2017). We assume $\mathcal{D}$ is partitioned into K disjoint shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$, with K workers each holding a model copy θ_k and local data shard $\mathcal{D}_k$.

In Distributed Data Parallelism (DDP) (Li et al., 2020; Shoeybi et al., 2019; Narayanan et al., 2021), each worker computes gradients locally and synchronizes them across workers at every step. When using SGD, this is equivalent to synchronizing parameters via FedAvg with $H = 1$, since SGD updates and parameter averaging are both linear and commute.

Federated Averaging (FedAvg) is well studied, with convergence analyses in (Stich, 2019), and extensions like FedOpt (Reddi et al., 2020), SCAFFOLD (Karimireddy et al., 2020), and asynchronous variants (Leconte et al., 2023).

DiLoCo (Douillard et al., 2023) builds on this literature for LLM training. Unlike FedAvg, which averages parameters every H steps, DiLoCo applies an outer optimizer (e.g., Nesterov Momentum (Sutskever et al., 2013)) after each synchronization. This approach has been used to train 1B–4B parameter models efficiently, reducing communication by a factor of $H \in [30, 100]$ (Jaghouar et al., 2024; Charles et al., 2025).

However, DiLoCo introduces additional complexity: local training typically uses AdamW, while the outer loop requires selecting and tuning a separate optimizer. The best choice depends on model size, number of nodes, synchronization interval H, and interactions between inner and outer optimizers (Charles et al., 2025).

3. SPARTA

3.1. The SPARTA Algorithm

FedAvg reduces communication costs by synchronizing the full model only once every H steps, averaging parameters across all nodes. In this section, we introduce *SPARTA*, a communication-efficient alternative where only a sparse subset of parameters is averaged at every step. Specifically, each parameter has a small independent probability p of being selected—for example, $p = 0.05\%$. In expectation, each parameter is averaged once every $1/p$ steps.

Initially, all nodes are identical: $\theta_k^{(0)} \leftarrow \theta^{(0)}$, after which each model trains independently to form an ensemble. At each training iteration t, after performing a forward pass, backward pass, and optimizer update, a global binary mask is sampled $m^{(t)} \sim \text{Bernoulli}(p)^D$. This shared mask determines the subset of parameters to communicate:

$$\mathcal{P}^{(t)} = \{i : m_i^{(t)} = 1\}. \tag{1}$$

Each node then updates its local parameter with the global average of that parameter across models:

$$\theta_{k,i}^{(t)} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \theta_{k,i}^{(t)}. \tag{2}$$

The *AllReduce* operation aggregates and redistributes these values across all nodes. Because only a sparse subset is communicated, SPARTA significantly reduces the effective communication cost compared to full-model averaging.

If we compare FedAvg and SPARTA under the condition $H = 1/p$, the total communication budget is the same. However, SPARTA distributes communication evenly over time-steps, and can be seen as a smoothed version of FedAvg. We demonstrate this property empirically in Figure 2. Note one critical difference with FedAvg: the models are never entirely synchronized but that rather remain an ensemble during an entire training run.

We provide theoretical justification of the equivalence of SPARTA and Federated Averaging in section A of the appendix, by demonstrating the same convergence properties between both SPARTA and FedAvg. This equivalence will then be demonstrated empirically in section 4.

Sampling: With vs. Without Replacement At each synchronization step, parameters can be sampled *with* or *without* replacement. With replacement, each parameter is selected independently with probability p (a Bernoulli process), leading to an expected $p \cdot D$ parameters per step, though the actual count varies. This method is simple, low-overhead, and assumes independent parameter behavior, but may result in uneven coverage.

Without replacement, exactly $\lfloor p \cdot D \rfloor$ unique parameters are selected uniformly, yielding more balanced updates and fixed communication cost per step, at the expense of added complexity.

For clarity and analysis, we adopt sampling with replacement; for experiments, we use sampling without replace-

ment to maintain constant bandwidth. As shown in the appendix, both strategies yield equivalent performance.

Coordination Without Communication Sampling is deterministic across nodes via a shared pseudo-random number generator (PRNG) and seed. Since all nodes know the global step t, they agree on the mask $\mathcal{P}^{(t)}$ without any communication. This ensures synchronization remains lightweight and scalable in decentralized settings.

Algorithm 1 Sparse Parameter Averaging (SPARTA)

Require: Data shards $\{D_1, \dots, D_k\}$, frequency $H \in \mathbb{R}$, sparsity rate $p \in \mathbb{R}$
for step $t = 1, \dots, T$ **do**
  Set $\mathcal{P}^{(t)}$ according to sampling strategy
  for worker $k = 1, \dots, K$ **in parallel do**
    $\theta_k^{(t)} \leftarrow \texttt{AdamW}(\theta_k^{(t-1)}, D_k^{(t)})$
    $\texttt{AllReduce}(\tilde{\theta}_j^{(t)} \leftarrow \theta_{k,j}^{(t)}, j \in \mathcal{P}^{(t)})$
    Update $\theta_k^{(t)}$ with $\tilde{\theta}^{(t)}$ at the sampled indices
  end for
end for $\frac{1}{K}\sum_{k=1}^{K} \theta_k^{(T)}$

3.2. SPARTA Extensions

3.2.1. Asynchronous Communication

A key property of SPARTA is that it does not require full-model synchronization. Since only a sparse subset of parameters is communicated, it is straightforward to allow these parameters to arrive with delay. This enables *asynchronous communication*: updates sent at time $t - \tau$ can be applied at time t, without blocking local training.

Let $\theta_k^{(t)}$ denote the local model on node k at step t, and $\mathcal{P}^{(t)}$ the mask of parameters selected for synchronization. After applying a local SGD update, each node performs:

$$\theta_{k,i}^{(t)} \leftarrow \tilde{\theta}_i^{(t-\tau_i)} \quad \text{for } i \in \mathcal{P}^{(t)}, \tag{3}$$

where $\tilde{\theta}_i^{(t-\tau_i)}$ is the average value of parameter i received from other nodes, computed at time $t - \tau_i$, and $\tau_i \leq \tau_{\max}$ is a bounded delay.

This allows local SGD computation and communication to overlap. While computing gradients for step t, the node can send out updates from step $t-1$ and receive updates from previous steps. Since $\mathcal{P}^{(t)}$ is sparse, communication remains efficient even under delay.

We can model the overall update as a variant of delayed SGD:

$$\theta_k^{(t+1)} = \theta_k^{(t)} - \eta \nabla \mathcal{L}_k(\theta_k^{(t)}) + \Delta^{(t)}, \tag{4}$$

where $\Delta^{(t)}$ is the sparse averaging correction:

$$\Delta_i^{(t)} = \begin{cases} \tilde{\theta}_i^{(t-\tau_i)} - \theta_{k,i}^{(t)} & \text{if } i \in \mathcal{P}^{(t)} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

In table 1 in the appendix, we demonstrate empirically the small change in performance under asynchronous communication.

3.2.2. Fault Tolerance

SPARTA is inherently tolerant to message loss. Since only a sparse subset of parameters is synchronized and full-model consistency is never required, training proceeds uninterrupted even when updates are dropped.

In our experiments (see Section 4), SPARTA maintains strong performance even when up to 95% of messages are randomly dropped ($q = 0.95$). This corresponds to $\tilde{p} = 0.05p$. Despite the severe communication loss, model quality degrades minimally, confirming that SPARTA is robust to random message failure without requiring retries, acknowledgments, or global synchronization.

In table 2 in the appendix, we show SPARTA's performance under dropped packets.

4. Results

In this section we provide experimental validation for the SPARTA algorithm. In all experiments excluding the 'larger-scale' experiments, we train a 124M parameter NanoGPT (Karpathy, 2022) model on the OpenWebText (Gokaslan and Cohen, 2019) across 4 nodes. Full details of our exprimental setup is left for the Appendix.

4.1. Quantitative Performance of SPARTA and FedAvg

In this section we emperically test the performance of SPARTA and FedAvg, and validate the assumption of equivalence between them. This will be shown theoretically in section A of the appendix. We compare SPARTA and FedAvg for an equal communication budget, when $p = \frac{1}{H}$.

In Figure 1 we see how SPARTA with compares to FedAvg for the same communication budget. The case of $H = 1$ is equivalent to data parallelism. As H increases, performance of both algorithms decreases at the same rate.

In Figure 4, we perform a learning rate sweep and again see similar performance. SPARTA is slightly more robust to learning rate changes but the results are extremely close.

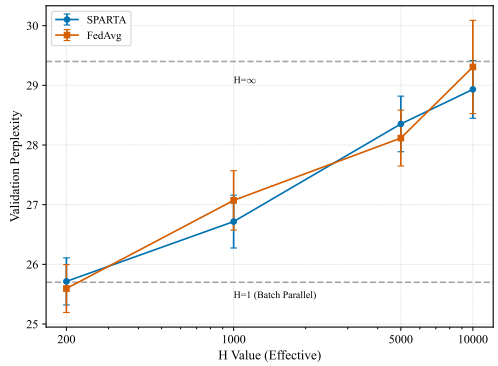

Figure 1. SPARTA and FedAvg achieve the same accuracy for a fixed communication budget.

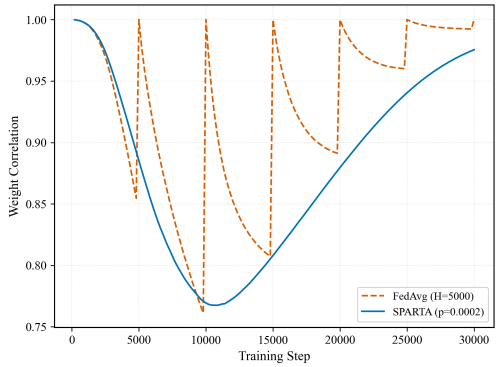

Figure 2. Weight matrix correlation

4.2. Communication Robustness and Fault-tolerance of SPARTA

We analyze the robustness of SPARTA for when updates are performed asynchronously as discussed in 3.2.1. The validation loss is measured after 30k training steps for synchronous training then compared to the validation loss with a time-step delay. Any additional delay results in slightly higher validation loss and the difference is reported in table 1. As shown, even a 10-step delay results in a performance drop (increase in validation loss) of 0.76%. A time-step delay of 1-step, results in no loss in performance. We additionally report an infinite delay, which is equivalent to never updating the model. Here, the drop in performance is enormous as models simply diverge during training and this results in poor performance when averaged together. This results highlights that SPARTA can be effectively used asynchronously with little to no performance loss.

Similarly, we observe in table 2 that SPARTA is extremely fault-tolerant. As discussed in 3.2.2, a fault rate q refers to the probability of synchronization messages being lost and never retried. Even with $q = 0.95$, SPARTA still works, but with a performance drop. Note that $q = 0.95$ means that in expectation, only 1 in 20 updates is synchronized between nodes.

Delay	Performance drop
5-step delay	0.36%
10-step delay	0.74%
∞ delay	805.6%

Table 1. SPARTA's performance with asynchronous communication.

Fault Rate	Performance drop
80% Fault Rate	2.48%
90% Fault Rate	3.66%
95% Fault Rate	4.88%

Table 2. SPARTA's fault-tolerance for lost messages.

This shows that SPARTA is asynchronous and fault-tolerant *by default*. This is in contrast the works on Async FedAvg (Leconte et al., 2023; Douillard et al., 2025) which require large algorithmic changes to achieve asynchronism. The asynchronous nature of SPARTA ensures that communication and computation are always overlapped, therefore adding no overhead to wall-clock time. Furthermore, the sharing parameter p can be chosen as a function of the available bandwidth. If a full forwards-backwards pass takes F seconds, and communicating the entire model would take B seconds, then we can choose to communicate $p = F/B$ of the model at each step asynchronously with no addition to wall-clock time.

4.3. Performance with an adaptive schedule

By default when using SPARTA, p is constant. However, we can adjust p according to schedule. One particular schedule we explore is setting $p = 1$ every H steps.

This is equivalent to the federated learning approaches, but additionally a small number of parameters are shared during the local steps. The updates are performed asynchronously with $p = 0.005$, incurring no additional wall-clock time. Additionally, $p = 1$ every $H = 10,000$ to match the DiLoCo interval. Both methods thus have **identical wall-clock time**. The result is shown in Figure 3 where we see SPARTA drastically outperforming DiLoCo. Notably, we see SPARTA outperforming DiLoCo even with $H = 100$, despite the total synchronous communication performed by SPARTA being **100x less**.

4.4. Large Scale runs

In the all the above experiments, we explore aspect of SPARTA on 124M LLMs. In this section, we provide 2 experiments at large scale: 770M parameters and 1.5B parameters. We compare SPARTA with with DiLoCo similar to the set-up in the previous section where SPARTA updates

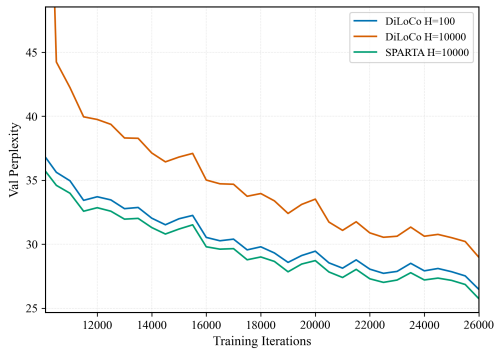

Figure 3. SPARTA reduces synchronous communication by 100× while *improving* perplexity at the 125 M parameter scale.

	770 M	**1.5 B**
Nodes	4	8
H	10 K	5 K
Training steps	76 K	23 K
p	0.5 %	0.5 %
PPL (DiLoCo)	18.1	24.3
PPL (SPARTA)	**16.5**	**19.8**

Table 3. Perplexity comparison of SPARTA and DiLoCo at the 770 M- and 1.5 B-parameter scales.

are performed async with $p = 0.005$ and a full sync at the same interval as DiLoCo to keep wall-clock time consistent between the two runs. We similar result in table 3 for larger models, demonstrating that SPARTA can greatly outperform DiLoCo with no increase in wall-clock time.

5. Conclusion

We introduced SPARTA, a sparse parameter averaging algorithm for bandwidth-constrained LLM training. We showed that SPARTA and FedAvg are theoretically and empirically equivalent under equal communication budgets, but SPARTA offers practical advantages: it enables continuous, fine-grained synchronization, supports asynchronous updates, and is inherently robust to message loss.

Leveraging these properties, SPARTA outperforms FedAvg-based baselines like DiLoCo in our experiments, demonstrating improved stability and efficiency in low-bandwidth settings.

SPARTA represents a step toward more flexible, decentralized training. By decoupling synchronization from rigid schedules and introducing sparsity in both time and parameter space, it enables training on heterogeneous or unreliable infrastructure.

Future work includes (1) integrating compression and quantization, (2) adapting sampling using gradient or sensitivity metrics, and (3) exploring advanced merging strategies such as curvature-aware model fusion.

References

Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. *arXiv preprint arXiv:2503.09799*, 2025.

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming diloco with overlapping communication: Towards a distributed free lunch, 2025. URL https://arxiv.org/abs/2501.18512.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. Opendiloco: An open-source framework for globally distributed low-communication training, 2024. URL https://arxiv.org/abs/2407.07852.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pages 5132–5143. PMLR, 2020.

Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Louis Leconte, Van Minh Nguyen, and Eric Moulines. Favano: Federated averaging with asynchronous nodes, 2023. URL https://arxiv.org/abs/2305.16099.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Sebastian U. Stich. Local sgd converges fast and communicates little, 2019. URL https://arxiv.org/abs/1805.09767.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Hyper parameter	125M	770M	1.5B
n_{layer}	12	36	48
n_{head}	12	20	25
n_{embd}	768	1280	1600
n_{hidden}	3072	5120	6400
Block size	1024	1024	1024
Vocab size	50 304	50 304	50 304
Dropout	0.0	0.0	0.0
Bias	True	True	True

Table 4. NanoGPT hyper-parameter configurations.[2]

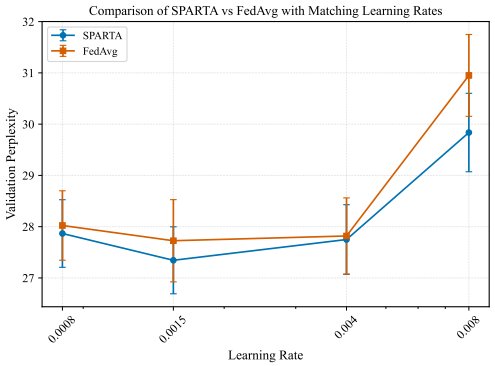

Figure 4. SPARTA is more robust to the learning-rate choice (H=200).

A. Experimental Setup

In our experiments, we simulate distributed training using one model per 4090 GPUs. All GPUs are on the same node. Since wall-clock time varies depending on the network bandwidth between workers, we quote training results in terms of training iterations instead of wall-clock time. We instead quantify the reduction in synchronous communication requirements as a function of the training run, from which a wall-clock time speedup can be inferred.

A.1. Model Architecture

We use a NanoGPT-style transformer model with 125M parameters in all experiments except the large-scale experiments in section 5.5. The hyperparameter configurations used are as follows:

A.2. Training Details

We use AdamW (Kingma and Ba, 2017) as our optimizer with a learning rate of 0.0008. We use a per-node batch size of 16 and a sequence length of 1024. Models are initialized to the same parameters, and are trained for 30k steps. The dataset used is OpenWebText (Gokaslan and Cohen, 2019), yielding 5B tokens after formatting. In all experiments comparing Federated Averaging and SPARTA, all hyperparameters are kept identical, including random seed to ensure batches of data are received in the same order. The only variable changed is the scheduling of paramter sharing; from Federated Averaging 'all-at-once' style to SPARTA's 'continuous' model parameter sharing.

For the large-scale experiments, we perform SPARTA parameter sharing whilst also doing a DiLoCo-style outer step every 5k or 10k iterations. In these experiments, we use the standard DiLoCo hyperparameters: outer learning rate 0.7, Nesterov momentum 0.9. We use SPARTA with $p = 0.005$. Given 3.6s per iteration for the 770M model, this requires a network bandwidth of 50Mb/s for communication and computation to take the same time and thus saturate the network.

[2] n_{hidden} (a.k.a. d_{ff}) is fixed at $4 \times n_{\text{embd}}$ for all sizes.

A.3. Asynchronous & Fault Tolerant SPARTA

To simulate asynchronous SPARTA, we store a buffer on each node to store the received SPARTA parameters before applying to the model. Averaged parameters wait in the buffer for a number of steps before being released to update the model. This is equivalent to asynchronous SPARTA over a delayed network.

To test fault-tolerance, we choose to ignore parameter updates periodically in order to simulate dropped packets. If we wish to simulate 90% fault-rate, we only use parameter updates every 10 iterations; the rest of the updates get dropped.

A.4. Learning-Rate Sweep for SPARTA vs FedAvg

We notice that SPARTA and FedAvg perform very similarly when varying learning rate, as observed in plot 4.

A.5. Weight Matrix correlation

In FedAvg, each nodes train independently with no communication with other nodes. This causes drift between local models, until they synchronize again. In Figure 2 we see this correlation between model weights. On the other hand, SPARTA never has a full synchronization in this experiment. Rather, a small set of parameters is shared, $p = 0.0002$. The models train locally and thus also drift apart, however as parameters are continuously shared, this results in a smooth correlation curve rather than a spiky one.

In Figure 5, we carry out a similar experiment but with 2 modifications. First we use DiLoCo rather than FedAvg as it performs much better for LLMs. In DiLoCo, rather that just averaging the parameters, an outer optimizer is used (Nesterov Momentum). Second, we use SPARTA with a

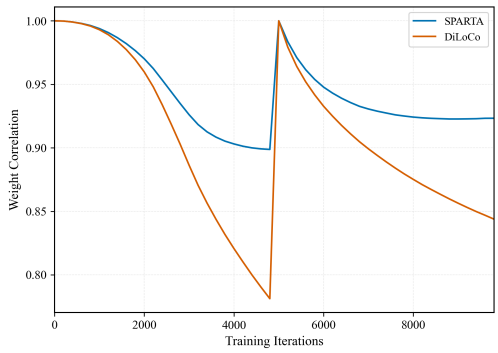

Figure 5. When used in conjunction with DiLoCo, SPARTA prevents models from catastrophic divergence

scheduled p such that $p = 1$ at the same interval used in DiLoCo. This effectively can be see as a federated learning algorithm with sparse parameter averaging during the local steps. The result is that correlation does not drop as low as the federated case due to the sparse parameter sharing reducing drift.

A. Theoretical Analysis

A.1. SPARTA and FA Equivalence

The primary difference between SPARTA and FedAvg is how parameter sharing is scheduled. FedAvg averages all the parameters once every H steps, which SPARTA averages a fraction p of the parameters every step. We will bound the convergence rate of SPARTA and demonstrate that SPARTA and FedAvg converge at the same rate when $p = 1/H$, showing that the different communication schedule does not affect performance. This is further justified experimentally in section 4.1, where we show that SPARTA and FedAvg perform almost identically for a given communication budget. We follow a similar analysis to (Stich, 2019) in finding the following theorem:

Theorem A.1 (Convergence of FedAvg and SPARTA)**.** *Let $\theta_k^{(t)}$ be the parameters on worker $k \in \{1, \dots, K\}$ after t SGD steps on an objective f that is L-smooth and μ-strongly convex. Assume stochastic gradients satisfy*

$$\mathbb{E}_i \left\| \nabla f_i(\theta_k^{(t)}) - \nabla f(\theta_k^{(t)}) \right\| \le \sigma^2, \quad \mathbb{E}_i \left\| \nabla f_i(\theta_{k,j}^{(t)}) \right\| \le \frac{G^2}{D},$$

for every coordinate $j \in \{1, \dots, D\}$ and step t. Choose $\eta_t = \frac{4}{\mu\,(a+t)}$ with $a > \max\{16\kappa,\ H\}$ and $\kappa = L/\mu$.

FedAvg *synchronises the entire model once every H steps;* SPARTA *synchronises a uniformly random subset of size pD every step, with $p = 1/H$. Then for both methods*

$$\mathbb{E}\, f(\hat{\theta}_T) - f^\star \;=\; O\Big(\tfrac{1}{\mu K T} + \tfrac{\kappa + H}{\mu K T^2}\Big)\sigma^2 + O\Big(\tfrac{\kappa H^2}{\mu T^2} + \tfrac{\kappa^3 H^3}{\mu T^3}\Big)G^2,$$

where $\hat{\theta}_T = \frac{1}{K S_T} \sum_{k=1}^{K} \sum_{t=0}^{T-1} w^{(t)} \theta_k^{(t)}$, $w^{(t)} = (a+t)^2$ and $S_T = \sum_{t=0}^{T-1} w^{(t)}$.

Proof. The analysis of (Stich, 2019) carries through verbatim once we show that the *drift* of each local model from the virtual average $\bar{\theta}^{(t)} = \frac{1}{K}\sum_k \theta_k^{(t)}$ obeys the same quadratic bound under both synchronisation schemes. Lemmas A.2 and A.3 below establish this. Using either bound in place of lemma 3.3 of (Stich, 2019) allows the theorem proof from (Stich, 2019) to be used. □

Lemma A.2 (Drift bound for FedAvg)**.** *Models $\theta_k^{(t)}$ are trained independently, with every H steps the whole model is averaged across workers. A sequence of decreasing positive stepsizes $\{\eta^{(t)}\}_{t\ge 0}$ satisfies $\eta^{(t)} \le 2\eta^{(t+H)}$ for all $t \ge 0$. Then,*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\|\bar{\theta}^{(t)} - \theta_k^{(t)}\|^2 \le 4(\eta^{(t)})^2 G^2 H^2, \tag{6}$$

under the assumption that $\mathbb{E}\left\|\nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(t)})\right\|^2 \le G^2$ for all $t \ge 0$.

Proof. Consider that all parameters were synchronized at timestep t_0 where $t - t_0 \le H$. Then we can expand the squared error term, using the fact that $\mathbb{E}\,\|X - \mathbb{E}[X]\|^2 = \mathbb{E}\,\|X\|^2 - \|\mathbb{E}[X]\|^2$,

$$\begin{aligned}
\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\bar{\theta}^{(t)} - \theta_k^{(t)}\right\|^2 &= \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|(\theta_k^{(t)} - \theta^{(t_0)}) - (\bar{\theta}^{(t)} - \theta^{(t_0)})\right\|^2 \\
&\le \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\theta_k^{(t)} - \theta^{(t_0)}\right\|^2
\end{aligned}$$

We can expand this term using the SGD update rule, $\theta_k^{(t)} = \theta^{(t_0)} - \sum_{h=t_0}^{t-1} \eta^{(h)} \nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(h)})$, where $\mathbf{x}_k^{(h)}$ represents the batch given to model k at timestep h.

$$\begin{aligned}
&= \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\sum_{h=t_0}^{t-1} \eta^{(h)} \nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(h)})\right\|^2 && (7)\\
&\le \frac{1}{K}\sum_{k=1}^{K} H\left(\eta^{(t_0)}\right)^2 \sum_{h=t_0}^{t-1} \mathbb{E}\left\|\nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(h)})\right\|^2 && (8)\\
&\le H^2\left(\eta^{(t_0)}\right)^2 G^2 && (9)\\
&\le 4H^2\left(\eta^{(t)}\right)^2 G^2 && (10)
\end{aligned}$$

Where we use $\left\|\sum_{a=1}^{h} \mathbf{x}_a\right\|^2 \leq h \sum_{a=1}^{h} \|\mathbf{x}_a\|^2$ to go from 7 to 8. □

To extend the proof to the SPARTA case, we state a lemma analogous to lemma A.2. We will prove the lemma for the case of sampling without replacement: parameters are shuffled once at the start, and enumerated to pick parameters during the training run. A similar bound can be found for the case of independent Bernoulli sampling, which requires a little more attention since τ_j, the time since last parameter sync, is a random variable.

Lemma A.3 (Drift bound for SPARTA)**.** *Models $\theta_k^{(t)}$ are trained independently, with parameters synchronized according to the SPARTA update rule, with parameters sampled without replacement. If every parameter was synchronized at most $H = 1/p$ steps ago, and a sequence of decreasing positive stepsizes $\{\eta^{(t)}\}_{t\geq 0}$ satisfies $\eta^{(t)} \leq 2\eta^{(t+H)}$ for all $t \geq 0$, then*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\|\bar{\theta}^{(t)} - \theta_k^{(t)}\|^2 \leq 4(\eta^{(t)})^2 G^2 H^2 \qquad (11)$$

under the assumption that $\mathbb{E}\left\|\nabla f(\theta^{(t)})_j\right\|^2 \leq G^2/D$ for all $t \geq 0$, $j \in \{1, \ldots, D\}$.

Proof. Using linearity of expectation, we view the squared error term as a sum over the parameters, and perform a similar analysis per-parameter:

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\|\bar{\theta}^{(t)} - \theta_k^{(t)}\|^2 = \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \mathbb{E}\left\|\bar{\theta}_j^{(t)} - \theta_{k,j}^{(t)}\right\|^2.$$

If parameter j was last synchronized at timestep τ_j then,

$$\begin{aligned}
&= \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \mathbb{E}\left\|\bar{\theta}_j^{(t)} - \theta_j^{(t-\tau_j)} - \left(\theta_{k,j}^{(t)} - \theta_j^{(t-\tau_j)}\right)\right\|^2 \\
&\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \mathbb{E}\left\|\theta_{k,j}^{(t)} - \theta_j^{(t-\tau_j)}\right\|^2 \qquad (12) \\
&= \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \mathbb{E}\left\|\sum_{h=t-\tau_j}^{t-1} \eta^{(h)} \nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(h)})_j\right\|^2 \qquad (13) \\
&\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \tau_j \sum_{h=t-\tau_j}^{t-1} \left(\eta^{(h)}\right)^2 \mathbb{E}\left\|\nabla f_{\mathbf{x}_k^{(h)}}(\theta^{(h)})_j\right\|^2 \\
&\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{D} \tau_j \sum_{h=t-\tau_j}^{t-1} \left(\eta^{(h)}\right)^2 \frac{G^2}{D} \qquad (14) \\
&\leq \sum_{j=1}^{D} \tau_j^2 \frac{G^2}{D} \left(\eta^{(t)}\right)^2 \qquad (15) \\
&\leq 4H^2 G^2 \left(\eta^{(t)}\right)^2 \qquad (16)
\end{aligned}$$

Where the step from (15) to (16) follows from the fact that $\tau_j \leq H$ for all j. □

While this is based on a simplified model relying on the linearity of SGD updates, it is demonstrative of the equivalence between Federated Averaging and SPARTA; the schedule by which parameters are shared doesn't affect rate of convergence.

A.2. Weight Correlation

Frequent, *sparse* synchronisation keeps SPARTA replicas better aligned than FedAvg for the same communication budget. Define the average inter-node correlation at time t by

$$\rho^{(t)} = \frac{1}{K(K-1)} \sum_{k \neq k'} \frac{\langle \theta_k^{(t)}, \theta_{k'}^{(t)} \rangle}{\|\theta_k^{(t)}\| \, \|\theta_{k'}^{(t)}\|}.$$

Theorem A.4 (Steady-state correlation)**.** *Two D-parameter models θ_1, θ_2 evolve by SGD with gradients distributed as $g_k \overset{i.i.d.}{\sim} \mathcal{N}(0, I\sigma^2), \quad k = 1, 2$. and learning rate η. At every step each parameter j is averaged between workers with probability p. Then the steady-state correlation between models is,*

$$\rho = \frac{1}{D}\sum_{j=1}^{D} \frac{\mathrm{Var}\left[H\right]}{\mathrm{Var}\left[H\right] + \frac{j}{pD}\eta^2\sigma^2},$$

where H is the distribution of model parameters at the time that they are averaged.

Proof. Consider first a single parameter j. Condition on the parameter last being averaged t_j time steps ago, at which point the synchronized parameter had value H_j (common to all ranks). The model parameter was allowed to evolve freely under SGD for t_j time steps, giving the distribution,

$$\begin{aligned}
\theta_{k,j} &\sim N\left(H_j, t_j\eta^2\sigma^2\right) \qquad (17) \\
\implies \mathrm{Cov}\left(\theta_{1,j}, \theta_{2,j}\right) &= \mathrm{Var}\left[H_j\right] \qquad (18) \\
\implies \mathrm{Var}\left[\theta_{k,j}\right] &= \mathrm{Var}\left[H_j\right] + t_j\eta^2\sigma^2 \qquad (19)
\end{aligned}$$

So the correlation conditional on time since last synchronization t_j is,

$$\rho_j = \frac{\mathrm{Var}\left[H_j\right]}{\mathrm{Var}\left[H_j\right] + t_j \eta^2 \sigma^2} \tag{20}$$

If parameters are sampled without replacement then they can be partitioned into sets $X_0, \ldots, X_{\lceil \frac{1}{p} \rceil}$ where X_h contains the parameters that were last averaged h steps ago, and $|X_0| = \cdots = \left|X_{\lceil \frac{1}{p} \rceil}\right| = pD$. Up to reordering, the t_j are therefore distributed as $t_j = \frac{j}{pD}$. So the steady-state correlation is,

$$\rho = \frac{1}{D} \sum_{j=1}^{D} \frac{\mathrm{Var}\left[H_j\right]}{\mathrm{Var}\left[H_j\right] + \frac{j}{pD} \eta^2 \sigma^2} \tag{21}$$

□

Corollary A.5. *In particular, if there is no weight decay term then* $\mathrm{Var}\left[H\right] \to \infty$*, so* $\rho \to 1$ *as* $t \to \infty$*.*

Proof. If there is no weight decay term then,

$$\theta_{k,j}^{(t+\tau)} \sim N\left(\theta_{k,j}^{(t)}, \tau \eta^2 \sigma^2\right) \tag{22}$$

$$\implies H_j^{(t)} \sim N\left(0, \frac{1}{2} t \eta^2 \sigma^2\right) \tag{23}$$

Where the $1/2$ follows from averaging the two Gaussians. Therefore as $t \to \infty$, $\mathrm{Var}\left[H_j\right] \to \infty$ and so $\rho \to 1$. □