Communication Efficient LLM Pre-training with SparseLoCo

Amir Sarfi[1]*
Benjamin Thérien[2]
Joel Lidin[1]
Eugene Belilovsky[3]
[1]*Covenant AI*
[2]*Université de Montréal, Mila*
[3]*Concordia University, Mila*

Abstract

Communication-efficient distributed training algorithms have received considerable interest recently due to their benefits for training Large Language Models (LLMs) in bandwidth-constrained settings, such as across data centers and over the internet. Despite reducing communication frequency, these methods still typically require communicating a full copy of the model's gradients—resulting in a communication bottleneck even for cross-datacenter links. Furthermore, they can slightly degrade performance compared to a naive AdamW DDP baseline. While quantization and error feedback are often applied to reduce the pseudo-gradient's size, in the context of LLM pre-training, existing approaches have been unable to additionally leverage sparsification and have obtained limited quantization. In this work, we introduce SparseLoCo, a communication-efficient training algorithm for LLMs that effectively leverages TOP-k sparsification and quantization to reach extreme compression ratios of up to 1–3% sparsity and 2-bit quantization while outperforming full-precision DiLoCo. Our key observations are that outer momentum can be locally approximated by an error feedback combined with aggressive sparsity and that sparse aggregation can actually improve model performance. We empirically demonstrate in a range of communication-constrained LLM training settings that SparseLoCo provides significant benefits in both performance and communication cost.

1. Introduction

Frontier language models pre-trained on internet-scale data have led to considerable breakthroughs in recent years. However, due to their growing parameter counts, effectively training these models across expensive datacenter hardware while retaining efficiency—a central goal due to the resources spent on these runs—is becoming increasingly challenging. On the other hand, due to the increasing availability of globally distributed computational infrastructure across the world, the pre-training of large-scale models over the internet has recently garnered special attention [12]. Similar to training over the internet, pre-training across multiple data centers requires mitigating the large communication overhead incurred by aggregating updates between workers.

In the context of LLM pre-training, several approaches have been proposed to reduce data-parallel communication cost. Among them are DiLoCo [6], a variant of LocalSGD [26, 31], as well as methods compressing communicated tensors and leveraging error feedback [1, 25, 37] to mitigate information loss. These techniques have complementary advantages of (1) reducing the communication frequency and (2) reducing the size of communicated messages. Combining the two is potentially advantageous for bandwidth-constrained settings like training over the internet

* Correspondence to amir@tplr.ai. Code can be found at: https://github.com/one-covenant/SparseLoCo.

or across data centers. However, existing works focused on LLM pre-training [3] do not take full advantage of compression schemes.

Indeed, combining these approaches raises the challenge of how to incorporate error feedback with an outer momentum, which is known to be important for DiLoCo's performance. Our key observation is that when aggressive TOP-k sparsification is combined with error feedback on DiLoCo pseudo-gradients, two effects emerge: (a) error feedback naturally acts as a local approximation of outer momentum, and (b) sparse aggregation is induced on the pseudo-gradients, a property recently shown in model merging contexts to improve performance [4, 38]. Building on this, we introduce **SparseLoCo**, which replaces global outer momentum with a single error feedback accumulator, thereby unifying infrequent and sparsified communication. This enables aggressive TOP-k sparsification and quantization of pseudo-gradients, while outperforming full-communication DiLoCo and frequency-compressed baselines.

Our contributions can be summarized as follows:

- We demonstrate empirically that outer momentum in DiLoCo can be replaced with a local approximation, which we link to TOP-k with error feedback.
- Leveraging this observation, we introduce SparseLoCo, a novel algorithm that blends the benefits of multi-iteration methods like DiLoCo with TOP-k sparsification and error feedback without compromising on performance or communication cost.
- Through our experiments, we demonstrate that SparseLoCo can significantly reduce the communication volume compared to existing LLM training methods (e.g., DiLoCo and DeMo), while simultaneously outperforming them.

Please refer to Appendix B for an extended discussion of related work.

2. Methodology

In this section, we first review DiLoCo. We then propose replacing the global outer momentum in DiLoCo with per-replica local outer momentum (LOM), where each replica maintains its own accumulator, an approach that will be used to empirically analyze the need for global momentum. Finally, we present our proposed method, **SparseLoCo**, which combines TOP-k compression with DiLoCo's infrequent communication.

2.1. Background and Notation

Consider the DiLoCo/FedOpt [5, 26] framework, which utilizes the following basic rule on each worker or replica to produce a pseudo-gradient at each outer step, $\Delta_r^{(t)}$, as follows:

$$\theta_r^{(t)} \leftarrow \mathrm{InnerOpt}_H\left(\theta^{(t-1)}; \mathcal{D}_r\right), \quad \forall r \in [R]$$
$$\Delta_r^{(t)} \leftarrow \theta^{(t-1)} - \theta_r^{(t)}$$

Here, H corresponds to the number of inner steps of the optimizer (typically AdamW), and R is the number of replicas. DiLoCo, which corresponds to an instantiation of FedOpt with AdamW as the inner optimizer and outer (server) momentum using Nesterov [8], is given as follows:

$$\bar{\Delta}^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^{R} \Delta_r^{(t)}$$
$$m^{(t)} \leftarrow \beta\, m^{(t-1)} + \bar{\Delta}^{(t)}, \quad \tilde{\Delta}^{(t)} = \bar{\Delta}^{(t)} + \beta\, m^{(t)}$$
$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\, \tilde{\Delta}^{(t)}$$

2.2. Local Outer Momentum

We first propose a variant of DiLoCo that utilizes a per-replica local outer momentum instead of the unified global momentum. The goal of this algorithm is to provide insight into how well the outer momentum can be locally approximated. We denote this variant of DiLoCo as DiLoCo-LOM (Local Outer Momentum):

$$m_r^{(t)} \leftarrow \beta\, m_r^{(t-1)} + \Delta_r^{(t)}, \quad \tilde{\Delta}_r^{(t)} \leftarrow \Delta_r^{(t)} + \beta\, m_r^{(t)}$$
$$\bar{\Delta}^{(t)} \leftarrow \frac{1}{R} \sum_{r=1}^{R} \tilde{\Delta}_r^{(t)}$$
$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha\, \bar{\Delta}^{(t)}$$

Here, the outer momentum is updated locally, solely based on the local pseudo-gradient, while the final update is based on the average of the local momentum accumulators. Note that typical implementations of DiLoCo store the outer momentum locally on each replica, meaning that DiLoCo-LOM does not add any memory overhead compared to the global momentum variant.

Building up to SparseLoCo, we consider an additional method, denoted DiLoCo-LOM-Sub-k, where the local momentums have their largest components removed at the end of each outer step:

$$m_r^{(t)} \leftarrow m_r^{(t)} - \text{TOP-}k\left(m_r^{(t)}\right)$$

Here, the local momentum has its largest-magnitude components removed. This allows us to study the impact of TOP-k subtraction, used in error feedback, without sparsifying the pseudo-gradient.

2.3. SparseLoCo: Sparse Aggregation meets Local Outer Momentum

We now introduce SparseLoCo, which blends TOP-k sparsification and error feedback in place of the local outer momentum. We consider error feedback, e_r, applied to the pseudo-gradients, which we denote OuterEF:

$$e_r^{(t)} \leftarrow \beta\, e_r^{(t)} + \Delta_r^{(t)}$$
$$\hat{\Delta}_r^{(t)} \leftarrow Q\left(\text{TOP-}k\left(e_r^{(t)}\right)\right), \quad e_r^{(t+1)} \leftarrow e_r^{(t)} - \hat{\Delta}_r^{(t)}$$

Here, Q is the quantization function that also allows us to leverage the error feedback to further reduce the message size transmitted. When k is sufficiently small, OuterEF closely approximates the local outer momentum in LOM, since only a few components will be subtracted from e_r. On the other hand, unlike LOM and LOM-Sub-k, SparseLoCo only aggregates quantized sparse vectors, drastically reducing the message size needed for communication. The full algorithm for SparseLoCo is given in Algorithm 1.

SparseLoCo uses a variant of the TOP-k operation inspired by [25] that divides tensors into discrete chunks and applies the TOP-k within each chunk. This has three benefits compared to applying it at the full-tensor or global level: (a) the cost of naively storing indices for transmission is significantly reduced as each chunk's index space is bounded. (b) TOP-k applied to entire models or individual tensors can overemphasize correlated variables; thus, chunking can have benefits on performance as further discussed in Appendix D. (c) Finally, chunking can allow for more easily integrating tensor parallelism and FSDP, which often require sharding across tensors, thereby creating inefficiencies for TOP-k operations over entire tensors or models.

Table 1: SparseLoCo compared to DiLoCo and gradient compression (DeMo). We show the size of the pseudo-gradients sent, the number of synchronizations, quantization supported, and loss. SparseLoCo outperforms other communication-efficient baselines in both communication efficiency and loss.

Method	Density	Loss	Pseudo-Grad Size	# of Syncs	Quantization
AdamW DDP	100%	2.70	1.03 GB	2445	16-bit
DiLoCo (H=15)	100%	2.76	0.48 GB	163	8-bit
DeMo	0.78%	2.83	8.5 MB	2445	8-bit
DeMo	3.12%	2.86	32.5 MB	2445	8-bit
SparseLoCo (H=15)	0.78%	2.80	5.5 MB	163	2-bit
SparseLoCo (H=15)	3.12%	2.73	17 MB	163	2-bit

3. Experiments

Our experiments use a 512M-parameter, LLaMA-style decoder-only transformer on DCLM [18] using the LLaMA-2 tokenizer [34]. Following [11], we allocate a token budget equal to $20\times$ the model size. Our experimental protocol follows [3]. Unless otherwise stated, we use a per-replica batch size B=256 with sequence length L=2048 and R=8 replicas, yielding a global batch of $B \times L \times R \approx 4.19$M tokens per step. In SparseLoCo, we employ 2-bit quantization with a chunk size of 4096. We report the hyperparameter sweep ranges, architectural details, and the best settings in Appendix F. As baselines, we include DiLoCo and DeMo—two strong communication-efficient methods (one using local iterations and the other using error feedback) for LLMs—as well as an AdamW baseline.

3.1. SparseLoCo

We first demonstrate that the local momentums of DiLoCo-LOM match DiLoCo's loss and closely approximate its global outer momentums (Appendix A.1). Table 1 compares SparseLoCo at H=15 against existing methods. We utilize 2-bit quantization for SparseLoCo with no observed loss degradation, while using the prescribed quantization settings for baselines [5, 25]. We observe that SparseLoCo obtains lower final loss than DiLoCo and DeMo baselines, while enjoying the simultaneous communication benefits of aggressively sparsified pseudo-gradients and reduced synchronization frequency. As SparseLoCo inherently utilizes error feedback, SparseLoCo further reduces communication size by quantizing the sparsified values. We further compare the performance on simple downstream tasks relevant at this model scale in Table 2, demonstrating that the performance improvements are consistent.

In Figure 1, we further demonstrate the performance improvements of SparseLoCo at increasing $H \in \{15, 30, 50, 100\}$ values compared to well-tuned DiLoCo and DiLoCo without outer momentum baselines. We observe that SparseLoCo can consistently obtain a better loss than DiLoCo in each case. Finally, following [3], we study an overtraining regime with a doubled token budget and a large communication interval (H=250), where SparseLoCo still outperforms DiLoCo (see Table 8 of the Appendix).

SparseLoCo's performance at different communication intervals (H) In Figure 1, we observe a trend that aligns with the hypothesis that SparseLoCo's Outer EF can provide similar benefits as a DiLoCo outer momentum. In particular, we first observe that not using outer momentum in

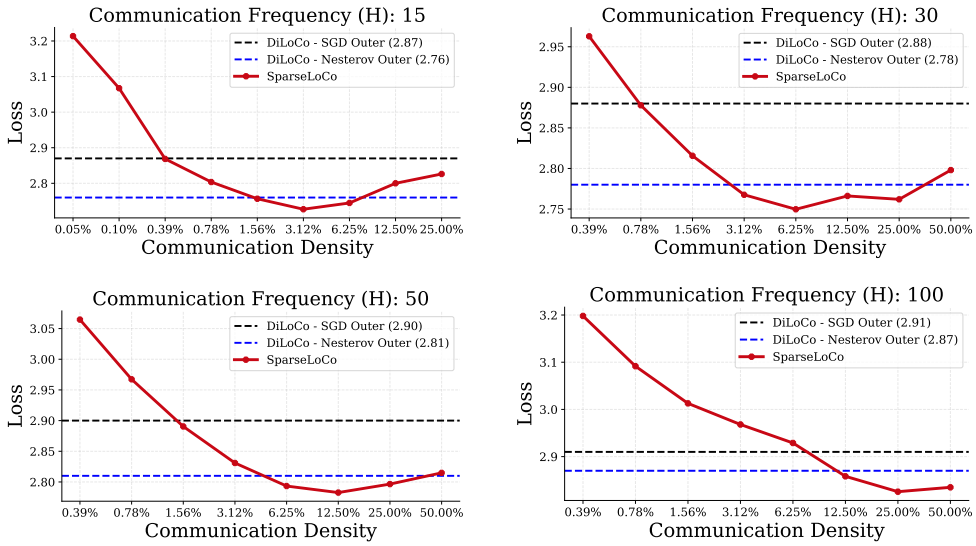


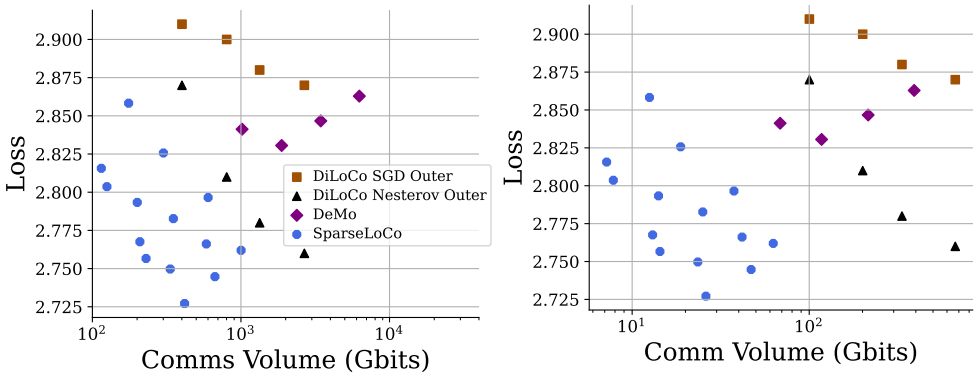
Figure 1: **SparseLoCo outperforms DiLoCo for $H \in \{15, 30, 50, 100\}$ communication steps.** We tuned SparseLoCo, DiLoCo, and DiLoCo without Nesterov for different communication intervals and at different sparsity levels for SparseLoCo. We report the best performance in each case. Crucially, SparseLoCo can outperform DiLoCo while communicating significantly less. We also observe that the optimal sparsity grows with the number of communication steps. All experiments were conducted with $R = 8$ workers.

DiLoCo leads to significant performance degradation and that this setting corresponds exactly to the fully dense case for TOP-k. With SparseLoCo, we observe that (i) extreme sparsity levels (when nearly nothing is sent, e.g., 0.05%) degrade performance. (ii) with increasing density (while remaining sparse), performance improves and eventually exceeds DiLoCo for all values of H. (iii) finally, as k approaches dense communication, the EF buffer becomes more sparse (due to line 14 in Algorithm 1), trending towards the performance of DiLoCo with no outer momentum and again degrading performance.

SparseLoCo can outperform DiLoCo and DiLoCo-LOM Across all settings of the inner steps (H), we observe regimes where SparseLoCo outperforms both DiLoCo and DiLoCo-LOM. A plausible explanation is that sparse aggregation at a well-chosen k emphasizes high-saliency components and reduces interference among updates, echoing intuitions from recent model-merging work in multi-task fine-tuning [4, 38].

Higher sparsity is needed with fewer inner steps We observe through Figure 1 that the optimal value is reached at a higher sparsity level with fewer inner steps. This is consistent with the fact that higher inner steps communicate information from a larger total number of samples. Indeed, we would expect that a trajectory with more steps would have a larger support.

SparseLoCo is at the Pareto frontier in communication volume In Figure 2 we compare the communication volume of SparseLoCo to DiLoCo, DiLoCo w.o. outer momentum, and DeMo. The exact communication setting and the underlying implementation of the aggregation can have a significant impact on the communication volume. We consider two common setups from the literature—methods utilize either ring all-reduce or ring all-gather (Fig. A), or a parameter server (Fig. B). We observe that in both cases, SparseLoCo lies on the Pareto frontier while other methods have a strictly worse trade-off. We note that the results in Fig. A assume aggregation using a naive



(A) Ring Communication **(B)** Parameter Server

Figure 2: **SparseLoCo lies on the Pareto frontier between loss and communication volume.** We report communication volume (outbound) for two settings: **(A)** ring communication topology (ring all-gather for SparseLoCo and DeMo, ring all-reduce for DiLoCo) and **(B)** parameter server. The points consider different H for DiLoCo, different densities for DeMo, and combinations of both for SparseLoCo. In both cases, SparseLoCo is on the Pareto frontier.

Table 2: Benchmark accuracy (%; higher is better). **Best** is bold; second best is underlined. We observe that SparseLoCo outperforms DeMo and DiLoCo baselines across all benchmarks.

Method	**arc_easy**	**hellaswag**	**piqa**
AdamW DDP	43.31%	**35.95%**	65.56%
DiLoCo	44.95%	34.14%	65.07%
DeMo	41.92%	32.37%	64.09%
SparseLoCo	**45.54%**	35.05%	**66.00%**

all-gather operation for implementation, while there is further potential to exploit the structure of the problem, for example, by summing overlapping indices along steps in the all-gather ring or utilizing specially designed all-reduce [19]. In Section C of the Appendix, we also discuss the communication measured during a live deployment of collaborative learning over the internet using SparseLoCo. Further ablation analysis is provided in Appendix A.3.

4. Conclusion

In conclusion, our work establishes that the outer momentum in DiLoCo can be replaced by local momentum accumulators without losing performance. Connecting local momentums with error feedback, we leverage this insight to develop *SparseLoCo*. Our algorithm can blend multi-iteration LLM pre-training methods with TOP-k sparsification and quantization, enabling aggressive compression of DiLoCo's pseudo-gradients. Our extensive experiments confirm that SparseLoCo significantly reduces communication while outperforming strong baselines such as DiLoCo and DeMo, placing it on the Pareto frontier of loss versus communication volume. Additionally, our experiments—outperforming DiLoCo—reveal that sparse aggregation may actually be crucial for improving the performance of multi-iteration, multi-worker methods compared to standard data-parallel pre-training. These results highlight the practical impact of SparseLoCo for communication-efficient LLM pre-training and suggest a path forward to scalable, low-bandwidth pre-training across data centers and globally distributed networks.

References

[1] Kwangjun Ahn and Byron Xu. Dion: A communication-efficient optimizer for large models. *arXiv preprint arXiv:2504.05295*, 2025. URL https://arxiv.org/abs/2504.05295.

[2] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification, and local computations, 2019.

[3] 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, March 2025. URL https://arxiv.org/abs/2503.09799.

[4] MohammadReza Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In *European Conference on Computer Vision*, pages 270–287. Springer, 2024.

[5] 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.

[6] Arthur Douillard, Qixuang 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. *CoRR*, abs/2311.08105, 2023. URL https://doi.org/10.48550/arXiv.2311.08105.

[7] 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, January 2025. URL https://arxiv.org/abs/2501.18512.

[8] Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *Proceedings of the 4th International Conference on Learning Representations, Workshop Track*, 2016. URL https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ.

[9] Ilyas Fatkhullin, Alexander Tyurin, and Peter Richtárik. Momentum provably improves error feedback! In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/f0b1515be276f6ba82b4f2b25e50bef0-Abstract-Conference.html.

[10] Louis Fournier, Adel Nabli, Masih Aminbeidokhti, Marco Pedersoli, Eugene Belilovsky, and Edouard Oyallon. Wash: Train your ensemble with communication-efficient weight shuffling, then average. *arXiv preprint arXiv:2405.17517*, 2024.

[11] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[12] Sami Jaghouar, Jack Min Ong, Manveer Basra, Fares Obeid, Jannik Straube, Michael Keiblinger, Elie Bakouch, Lucas Atkins, Maziyar Panahi, Charles Goddard, Max Ryabinin, and Johannes Hagemann. INTELLECT-1 technical report. *CoRR*, abs/2412.01152, 2024. URL https://doi.org/10.48550/arXiv.2412.01152.

[13] Charles-Étienne Joseph, Benjamin Thérien, Abhinav Moudgil, Boris Knyazev, and Eugene Belilovsky. Meta-learning optimizers for communication-efficient learning. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=uRbf9ANAns.

[14] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3252–3261, 2019. URL https://arxiv.org/abs/1901.09847.

[15] 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.

[16] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[17] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. *arXiv preprint arXiv:2304.13960*, 2023.

[18] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Scott Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F. Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M. Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Raghavi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/

hash/19e4ea30dded58259665db375885e412-Abstract-Datasets_and_Benchmarks_Track.html.

[19] Shigang Li and Torsten Hoefler. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 135–149, 2022.

[20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html.

[21] Joel Lidin, Amir Sarfi, Evangelos Pappas, Samuel Dare, Eugene Belilovsky, and Jacob Steeves. Incentivizing permissionless distributed learning of llms. *arXiv preprint arXiv:2505.21684*, 2025.

[22] Tao Lin, Sebastian U. Stich, and Martin Jaggi. Don't use large mini-batches, use local SGD. *CoRR*, abs/1808.07217, 2018.

[23] 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.

[24] Jose Javier Gonzalez Ortiz, Jonathan Frankle, Mike Rabbat, Ari Morcos, and Nicolas Ballas. Trade-offs of local sgd at scale: An empirical study. *arXiv preprint arXiv:2110.08133*, 2021.

[25] Bowen Peng, Jeffrey Quesnelle, and Diederik P Kingma. Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

[26] 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.

[27] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2021–2031. PMLR, 2020. URL http://proceedings.mlr.press/v108/reisizadeh20a.html.

[28] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8253–8265. PMLR, 2020. URL http://proceedings.mlr.press/v119/rothchild20a.html.

[29] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In Haizhou Li, Helen M. Meng, Bin Ma, Engsiong Chng, and Lei Xie, editors, *15th Annual Conference of the International Speech Communication Association, INTERSPEECH 2014, Singapore, September 14-18, 2014*, pages 1058–1062. ISCA, 2014. URL https://doi.org/10.21437/Interspeech.2014-274.

[30] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *CoRR*, abs/1911.08772, 2019.

[31] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[32] Sebastian U. Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *CoRR*, abs/1909.05350, 2019. URL http://arxiv.org/abs/1909.05350.

[33] Benjamin Thérien, Xiaolong Huang, Irina Rish, and Eugene Belilovsky. Muloco: Muon is a practical inner optimizer for diloco, 2025. URL https://doi.org/10.48550/arXiv.2505.23725.

[34] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[35] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14236–14245, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/d9fbed9da256e344c1fa46bb46c34c5f-Abstract.html.

[36] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael G. Rabbat. Slowmo: Improving communication-efficient distributed SGD with slow momentum. *CoRR*, abs/1910.00643, 2019.

[37] Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. CocktailSGD: Fine-tuning foundation models over 500Mbps networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36058–36076. PMLR, 23–29 Jul 2023.

[38] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 2023. URL https://arxiv.org/abs/2306.01708.

Algorithm 1: SparseLoCo

Require: initial parameters $\{\theta_r^{(0)}\}$; inner steps H; outer steps T; outer learning rate α; error momentum β; R workers

for $t \leftarrow 1$ **to** T **do**
  for $r \leftarrow 1$ **to** R **do**
    Local inner loops
    $\theta_r^{(t)} \leftarrow \theta_r^{(t-1)}$
    for $h \leftarrow 1$ **to** H **do**
      Sample $x \sim \mathcal{D}_r$
      $L \leftarrow f(x, \theta_r^{(t)})$
      $\theta_r^{(t)} \leftarrow \mathrm{AdamW}(\theta_r^{(t)}, \nabla L)$
    end
    $\Delta_r^{(t)} \leftarrow \theta_r^{(t-1)} - \theta_r^{(t)}$ `// Pseudo-gradient`
    Compression + Error Feedback
    $e_r^{(t)} \leftarrow \beta\, e_r^{(t)} + \Delta_r^{(t)}$
    $\hat{\Delta}_r^{(t)} \leftarrow Q\Big(\mathrm{Top}\text{–}k\Big(e_r^{(t)}\Big)\Big)$ `// Transmit` $\hat{\Delta}_r^{(t)}$
    $e_r^{(t+1)} \leftarrow e_r^{(t)} - \hat{\Delta}_r^{(t)}$
    Aggregate + Outer Update
    $\Delta^{(t)} \leftarrow \frac{1}{R}\sum_{r=1}^{R} \hat{\Delta}_r^{(t)}$
    $\theta_r^{(t+1)} \leftarrow \theta_r^{(t)} - \alpha\, \Delta^{(t)}$
  end
end

Appendix A. Extended Experiments

A.1. Building intuition with Local Outer Momentum

We first use the DiLoCo-LOM and DiLoCo-LOM-Sub-k algorithms to empirically link the standard outer momentum in DiLoCo to the error feedback mechanism in SparseLoCo. In DiLoCo-LOM, each replica maintains a local outer momentum accumulator that is averaged only at synchronization. In DiLoCo-LOM-Sub-k, we subtract the largest entries of the local momentum after each synchronization, isolating the effect of TOP-k subtraction while keeping the communicated pseudo-gradients dense. As shown in Table 3, DiLoCo-LOM matches DiLoCo's loss, and pruning 25% of the largest accumulator entries has a negligible impact, whereas removing outer momentum entirely degrades performance.

To quantify how closely local outer momentum tracks the target global outer momentum in DiLoCo, we maintain a reference global accumulator and, over the first 20 outer steps, compute the cosine similarity between this reference and each replica's local accumulator at corresponding steps. The average similarity is ≥ 0.90 for DiLoCo-LOM and ≥ 0.75 for DiLoCo-LOM-Sub-k (25%), indicating that sparsified local accumulators remain a strong directional proxy for the

Table 3: **DiLoCo's global outer momentum is well approximated by Local outer momentum and sparsified local outer momentum.** We report the final performance of decoder-only transformers trained with each method and find that both our proposed *local momentum* methods match the performance of DiLoCo.

Method	Loss
DiLoCo	2.760
DiLoCo w.o. outer momentum	2.868
DiLoCo-LOM	2.759
DiLoCo-LOM-Sub-k - 25%	2.761

global momentum and supporting our interpretation of SparseLoCo's error feedback state as a local approximation of DiLoCo's outer momentum.

A.2. SparseLoCo can be used with Ring All-Reduce as a drop-in for DiLoCo

Although our analysis and motivation in the work focuses on aggressively compressing the per-iteration message size we note that in communication settings where efficient all-reduce is already available and preferred, SparseLoCo still provides significant benefit over DiLoCo while incurring no additional memory or compute overhead. Concretely, the aggregation step in Algorithm 1 Line 17 can be performed directly by an all-reduce over a sparse vector. This has two significant benefits over DiLoCo with AR: (1) As observed in Table 8 and Figure 1, the performance when k is optimally selected is improved over DiLoCo (2) the Outer error feedback naturally supports more aggressive quantization than the naive DiLoCo, allowing for 2-bit quantization to be used *without an additional accumulator*, unlike [33].

A.3. Ablations

We highlight key design choices of SparseLoCo through a series of ablations.

Outer Momentum + OuterEF A natural way to combine DiLoCo with OuterEF is by adding an error feedback as well as a Nesterov outer optimizer, as has been attempted by [33], who showed it can provide benefits for quantization but not sparsification. This approach requires an additional accumulator. In contrast, our finding is that, in the case of a high sparsification, using a global outer momentum can be detrimental to performance. This is illustrated in Table 4, where we equip SparseLoCo's outer optimizer with Nesterov outer momentum. This significantly degrades performance at high sparsity. We hypothesize that this is due to the conflicting directions of the error feedback momentum and the outer momentum, since the largest components become amplified by the outer momentum but not the error feedback.

Table 4: Naively combining DiLoCo's standard Nesterov outer optimizer yields poor results. We use 3.12% communication density in both settings.

Method	Loss
SparseLoCo	2.72
SparseLoCo + Nesterov Outer optimizer	3.39

Random-K We ablate the choice of TOP-k compared to the alternative Random-k [30, 37] in Table 5. We observe that performance is significantly degraded when using random-k for the same number of indices selected, emphasizing the importance of this design choice.

Table 5: SparseLoCo with Random-k sparsification vs TOP-k sparsification. We observe that TOP-k significantly outperforms Random-k across all communication densities.

Density	Random-k Loss	TOP-k Loss
1.56%	3.05	2.75
3.12%	2.98	2.72
6.25%	2.93	2.75

Quantization As discussed, SparseLoCo benefits from stronger quantization than non-EF methods and supports up to 2-bit quantization and was generally observed to give results very close to full precision. In Table 6 below, we show the performance at different quantization values, showing that 2-bit quantization can be achieved at almost no performance cost.

Table 6: Effect of Quantization on Loss (Lower is Better). 2-bit quantization brings almost no performance degradation compared to full precision.

Quantization bits	Loss
1	4.98
2	2.73
3	2.73
4	2.73
32	2.72

Appendix B. Related Work

Federated Learning In the federated learning literature, communication efficiency has been a central focus from the outset, as participating clients often operate over highly constrained and heterogeneous networks. A canonical example is Federated Averaging (FedAvg) [16, 23], which reduces communication frequency by performing multiple local updates before averaging model parameters. Other works explore compressed updates through sketching or quantization in the context of federated learning [27, 28]. Beyond reducing communication overhead, numerous approaches such as SCAFFOLD [15] and FedProx [20] address the unique challenge of data heterogeneity—where each client's dataset may follow a different distribution—by introducing control variates or proximal terms to stabilize convergence. While our work shares federated learning's emphasis on reducing communication overhead, it differs fundamentally in scope: we focus on large-scale pre-training of LLMs in settings with homogeneous data partitions (e.g., sharded web-scale corpora), where heterogeneity-mitigation strategies are unnecessary.

LocalSGD and extensions to LLM training Local Stochastic Gradient Descent (LocalSGD) [31] is a widely studied approach for reducing communication in distributed training by allowing workers to perform multiple local updates before synchronizing. [31] formally introduced the method

and proved its convergence, while [22] highlighted that LocalSGD can lead to improved generalization compared to simply increasing the batch size. Extensions of LocalSGD include SlowMo [36], which incorporates a slow outer momentum to stabilize training in datacenter-style environments—while still using SGD as the inner optimizer—and meta-learning approaches [13] that adapt the aggregation function for improved performance. However, these approaches were not shown to scale well to pre-training in [24]. More recently, DiLoCo [6] adapted the LocalSGD framework to Large Language Model (LLM) pre-training, demonstrating that replacing the inner optimizer with AdamW can yield substantial benefits. Our work builds upon this line of research by enabling aggressive TOP-k sparsification of the communicated pseudo-gradients in a LocalSGD-style framework, something that prior methods have not achieved while maintaining or improving upon state-of-the-art LLM training performance. Finally, [7] and [10] consider communicating a small subset of model parameters more frequently instead of communicating a message the size of the model infrequently by allowing the models to desynchronize while still remaining relatively close to each other (in terms of, e.g., consensus distance). SparseLoCo, on the other hand, maintains the benefit of infrequent communication while communicating a small-sized message and maintaining model synchronization.

Error Feedback and Compressed Updates Error feedback (EF) has been extensively studied, particularly from a theoretical perspective, as a means to compensate for the information loss introduced by various gradient compression methods [14, 29, 32]. It has been combined with various compression techniques, including quantization, sparsification [30], and low-rank approximation in [1, 35], and applied to LLMs in recent works [25, 37]. EF21-SGDM [9] analyzed how to combine error feedback with momentum, introducing a momentum-compatible variant that requires two accumulators and is largely focused on theoretical aspects and does not address the multi-iteration setting or the practical challenges of LLM pre-training. QSparseLocalSGD [2] is, to our knowledge, one of the few works that combines multi-iteration methods such as LocalSGD with error feedback, but its focus was on theoretical analysis with non-adaptive optimizers and without outer momentum. In contrast, our work targets the LLM pre-training regime and develops a method to combine aggressive TOP-k compression and error feedback with an efficient approximation of outer momentum. DeMo [25] considers EF with DCT encoding and TOP-k compression in the LLM setting, demonstrating it can achieve competitive performance, but without incorporating local updates or the ability to leverage adaptive optimizers. Similarly, CocktailSGD [37] uses error feedback with multiple compression operators in an LLM fine-tuning setting, yet does not explore the integration of local iteration methods. To the best of our knowledge, this is the first work to study the combination of these approaches in the context of LLMs and more generally in the context of modern variations of multi-iteration methods that have been shown to scale to pre-training.

Appendix C. Real-World Deployment for Collaborative Permissionless Distributed Training over the Internet

SparseLoCo has been deployed in a real-world setting and used to collaboratively train models up to 8B with permissionless global participants using an incentive scheme [21] that rewards participants purely based on analysis of their compressed pseudo-gradients. This was done through the Bittensor blockchain and the templar project. The addition and coordination of peers and their rewards being handled through the blockchain. Communication of pseudo-gradients was routed through globally distributed, S3-compliant object storage—specifically Cloudflare R2—which enabled rapid dissem-

ination of model updates worldwide. This setup allowed updates to be time-stamped and verified as part of the reward mechanism [21]. Each peer maintained their own storage bucket, posting read credentials to a blockchain so that both other peers and the reward mechanism could access their compressed pseudo-gradients.

The deployment used R=20 replicas each associated to a peer, H=30 inner steps and a global batch size of ∼8M tokens per inner step. Although the system design allows peers to use any target hardware that can achieve reasonable throughput, the suggested hardware requirements targeted an 8× H200.

SparseLoCo is particularly advantageous in this communication setup, as cloud providers have large bandwidth for peer downloads and are able to rapidly distribute and mirror files across the globe. For upload, a peer only sends their pseudo-gradient through the cloud provider. Therefore, their outbound communication (and required upload bandwidth) is kept low. They then download the pseudo-gradients from the cloud provider, which is able to easily handle the high bandwidth constraints. An example of a practical communication time measured with the 8B model is on average 12 seconds, including sending their compressed pseudo-gradients and downloading other workers' messages with the test node never exceeding 500 Mb/s. Compared to processing with 8× H200, which takes around 4.5 minutes, leading to minimal wall-clock time degradation despite traffic over the internet. For reference, [12], which trained a similarly sized model (10B) with 8-bit DiLoCo, reports a globally distributed all-reduce synchronization time of an average 8.3 minutes for a peak of R=14 nodes participating and processing time of 38 minutes. We also performed test measurements of communication time for a 70B model with the same setup as above (R=20 peers), measuring a total communication time of 70 seconds on average, with the test node never reaching more than 500 Mb/s downlink and 110Mb/s uplink.

Appendix D. Effect of Chunking, DCT, and inner steps

A recently introduced method [25] considered the single-step setting with error feedback, using a compression function that first applies a discrete cosine transform (DCT) on tensor chunks and then selects the TOP-k values in the DCT domain. It further employed sign descent on the final aggregated update [17]. Without the DCT transform, this approach can be seen as a special case of SparseLoCo when $H = 1$, the inner optimizer is plain SGD, and the outer optimizer utilizes sign descent. Since the effect of the DCT transform, designed for data with sequential structure where the order of elements matters, is not well understood in this context, and given the additional uncertainty about the role of chunking, in this section we disentangle the contributions of both for DeMo, as well as in the multi-step ($H > 1$) setting. The ablations of these three factors, evaluated purely in terms of loss, are presented in Table 7. Here, the TOP-k EF baseline is a simplified DeMo that applies TOP-k selection globally to the entire tensor (rather than within chunks) while still utilizing sign descent.

We observe that in the setting with no local steps (TOP-k EF, DeMo) the impact of chunking is very significant and the performance of DeMo can be nearly recovered without resorting to the DCT. When using the local setting (H>1), we observed that DCT actually degrades performance; however, we also find that the impact of chunking is more limited than in the setting of H=1. We hypothesize that chunking and DCT both serve to reduce the effect of outlier values on the scale of individual workers' contributions, which may be less critical in the case of SparseLoCo due to its adaptive inner optimizer.

Notably, DeMo does not have a natural way to incorporate adaptive optimization, and in practice, the sign descent is used to approximate the benefits of the Adam optimizer [25]. A significant advantage of SparseLoCo is that operating on the pseudo-gradients allows easy integration of adaptive optimizers like Adam in the inner loop.

Table 7: Ablation of tensor chunking and DCT (lower loss is better). We observe that chunking is critical for the performance of DeMo. With $H>1$, DCT degrades performance. All runs use full precision (FP32).

Method	**No DCT**	**DCT**
SparseLoCo ($H > 1$, Chunking, TOP-k EF)	**2.72**	2.75
SparseLoCo w/o Chunking ($H > 1$, TOP-k EF)	2.73	2.76
DeMo (Chunking, TOP-k EF w/ Sign Descent)	2.87	2.83
TOP-k EF (w/ Sign Descent)	3.48	2.84
DiLoCo	2.76	–

Appendix E. Overtraining Regime with Large Communication Interval

Following [3], we put SparseLoCo to the test in an overtraining regime by doubling the token budget to 20B and using a larger communication interval of H=250. Our observations are consistent with the trends in Figure 1, and SparseLoCo outperforms DILOCO at this setting (Table 8).

Table 8: Overtraining on 2× data (20B token budget) with communication interval H=250.

Method	**Density**	**Loss**
DiLoCo	100%	2.77
SparseLoCo	50%	**2.73**
SparseLoCo	25%	2.74
SparseLoCo	12.5%	2.79
SparseLoCo	3.12%	2.97
SparseLoCo	1.56%	3.00

Appendix F. Hyperparameter Selection Details

In Table 9, we show the hyperparameter search spaces. We tune all methods at H=15 and reuse the best settings for other H; for DiLoCo, we also sweep H=30, and use its best setting for higher H. General and model settings are fixed across all runs unless stated otherwise.

Appendix G. Compression of indices in TOP-k

In TOP-k methods, the indices of the selected values need to be transmitted alongside the values. When values are aggressively quantized (as in SparseLoCo), this index-transmission overhead becomes significant. In SparseLoCo, we utilize chunk sizes of C=4096, so, naively, we can transmit indices in 12 bits per transmitted value. However, with 2-bit quantization, this overhead becomes significant, motivating further index compression. Assuming a chunk size of C and TOP-k selection, we observe that the information-theoretic limit is $\log_2\binom{C}{k}$ bits. For practical cases considered

in this work (C=4096 and $k \in \{32, 128, 256\}$), this corresponds to 8.3, 6.3, and 5.3 bits per transmitted value, respectively. In practice, we were able to design a custom compression algorithm based on sub-chunking and coding that achieves 8.9, 6.6, and 5.6 bits per value for these cases.

Table 9: Hyperparameter search spaces. Bold entries indicate the best settings. Model and general settings (top) are fixed across all runs. We tune all methods at H=15 and reuse the best hyperparameters when varying H, while we further sweep H=30 for DiLoCo and use its best setting. The effective batch size is given per inner step across all workers.

General Settings	**Value**
Token Budget	10.26B
Effective batch size	4,194,304
Sequence length	2048
Local batch size	256
Workers R	8
Warmup steps	500
Inner gradient clipping	1.0
LR Decay	Cosine
Inner optimizer	AdamW
AdamW β_1	0.9
AdamW β_2	0.95

Parameter	**Value**
Total Parameters	512,398,848
Number of Layers	12
Hidden Size	1536
Intermediate Size	5,440
Attention Heads	12
Vocabulary Size	32,000
FFN Activation	SwiGLU

Setting	**Hyperparameter**	**Search Space**
AdamW Baseline	α	4e-4, 6e-4, 8e-4, 1e-3 **2e-3**, **3e-3**, **4e-3**, 6e-3
DiLoCo - Nesterov Outer	*H=15*	
	α_{inner}	4e-4, 6e-4, **8e-4**, 1e-3
	α_{outer}	0.2, 0.4, **0.6**, 0.8, 1.0
	momentum	**0.9**
	H=30	
	α_{inner}	**6e-4**, 8e-4
	α_{outer}	**0.8**, 0.6
	H=50, 100, 250	
	α_{inner}	**6e-4**
	α_{outer}	**0.8**
SparseLoCo (H=15, Density=0.78%)	α_{inner}	6e-4, 8e-4, **1e-3**, 3e-3
	α_{outer}	0.4, 0.6, 0.8, **1.0**
	error momentum (β)	0.9, **0.95**, 0.999
DiLoCo - SGD Outer	α_{inner}	6e-4, **1e-3**
	α_{outer}	0.8, **1.0**
	momentum	**0.0**
DiLoCo-LOM	α_{inner}	6e-4, **8e-4**, 1e-3
	α_{outer}	0.4, **0.6**, 0.8, 1.0
	momentum	**0.9**
DeMo	α	8e-4, **1e-3**, 3e-3
	error momentum (β)	0.95, **0.999**