Incentivizing Permissionless Distributed Learning of LLMs

Joel Lidin[1]*
Amir Sarfi[1]
Evangelos Pappas[1]
Samuel Dare[1]
Eugene Belilovsky[2]
Jacob Steeves[3]
[1] *Covenant AI*
[2] *Concordia University, Mila*
[3] *Opentensor Foundation*

Abstract

We describe an incentive system for distributed deep learning of foundational models where peers are rewarded for contributions. The incentive system, *Gauntlet*, has been deployed on the bittensor blockchain and used to train a 1.2B LLM with completely permissionless contributions of pseudo-gradients: no control over the users that can register or their hardware. *Gauntlet* can be applied to any synchronous distributed training scheme that relies on aggregating updates or pseudo-gradients. We rely on a two-stage mechanism for fast filtering of peer uptime, reliability, and synchronization, combined with the core component that estimates the loss before and after individual pseudo-gradient contributions. We utilized an OpenSkill rating system to track competitiveness of pseudo-gradient scores across time. Finally, we introduce a novel mechanism to ensure peers on the network perform unique computations. Our live 1.2B training run, which has paid out real-valued monetary rewards to participants based on the value of their contributions, yielded a competitive (on a per-iteration basis) 1.2B model that demonstrates the utility of our incentive system.

1. Introduction

Training large foundational models, such as large language models (LLMs), remains dominated by centralized actors with access to vast computational resources. However, as these models grow in importance and influence, so does the imperative to democratize their development. Foundational models such as LLMs are typically trained in large, well-interconnected centralized data centers, in part due to the heavy communication costs of typical data parallel distributed learning methods. However, recent advancements in communication-efficient distributed learning [1, 4, 12, 19] open the possibility to decentralize computation. This opens the door to new paradigms such as permissionless distributed training, where anyone can contribute updates to a shared model. While recent works have demonstrated pre-training with decentralized or federated learning [3, 6, 16], these approaches have not fully addressed the challenges of incentivization and quality control in open networks opting for vetted contributors. In particular, ensuring honest participation in such systems remains a challenge.

Recent work has considered verification of machine learning programs[2], from untrusted users which can include distributed training. However, this work on verification focuses on assuring that exact pre-described computations have been performed. On the other hand, contributors in incentivized systems may vary their computations (e.g., data selection or hyperparameters). Proof of Learning (PoL) has also been proposed in [7] and can be seen as a type of verification mechanism

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

that attempts to recompute the exact calculations that a learner should have performed. It avoids re-training of the entire model by only recomputing part of the computation. Similar to verification, existing PoL work does not consider the case where a learner can deviate and even improve on the prescribed learning scheme (such as using more data than specified for a gradient step).

Incentivized distributed training can be seen as a generalization of verification systems in distributed learning. The goal is to both ensure that untrusted users provide useful computations to the system, and to incentivize competition and innovation between users. For example, the incentive system may encourage participants to optimize their local hardware, networking, as well as their implementation in order to maximize throughput and utility of their contributions.

In this paper, we introduce the Gauntlet incentive mechanism, a system designed to enable and reward high-quality contributions in a permissionless distributed training setting. Gauntlet efficiently evaluates and compares pseudo-gradient contributions from peers participating in a distributed training run. Gauntlet has been deployed on the Bittensor blockchain and used in a live training run of a 1.2B parameter language model, where contributors provided compressed pseudo-gradients without any centralized registration or approval. This model achieved competitive performance per iteration, with the Gauntlet protocol paying real-valued tokens to participants in proportion to the utility of their updates.

Our 1.2B model run is, to our knowledge, the first truly permissionless pre-training LLM run. Any user with a valid internet connection is able to make a contribution without needing approval, coordination, or identification. It demonstrates that foundational model training can be conducted in a completely open network with minimal assumptions about trust, identity, or compute capabilities. Our study opens the door to decentralized AI models sustained by market-driven incentives.

2. Distributed Training Framework

We consider a general synchronous distributed training scheme where a model θ is updated with learning rate α as follows:

$$\theta_t = \theta_{t-1} - \alpha \sum_{k=1}^{K} w_k \Delta_k \tag{1}$$

In this scheme, K distributed peers contribute "pseudo-gradients", Δ_k, which are aggregated and used to perform an optimization step. We note that this is a generic framework encompassing a number of popular data parallel distributed learning schemes [4, 5, 12, 14, 15, 17, 19]

Training proceeds in communication rounds t, each with a specified duration. At the end of each round, we define a 'put window': a short period during which peers must publish their pseudo-gradients. Submissions made outside this window (i.e., too early or too late) are ignored.

To implement incentivization, we consider the concept of a validator that can access any contribution Δ_k for evaluation. The validator maintains a score for all participants, which is periodically posted on the public blockchain. This score is used to determine the amount of monetary rewards given to participants on the network.

The instantiation of this framework that we evaluate uses the communication-efficient Decoupled Momentum Optimizer (DeMo)[12] to produce pseudo-gradients (see Algo. 2). DeMo is a variant of compression with error feedback methods [9, 17, 19]. The compressor utilized by DeMo applies a Discrete Cosine Transform (DCT) operation on chunked tensors, decorrelating the values before applying a top-k operation. This method has been shown to achieve competitive compression ratios on LLM training [1, 12].

3. Gauntlet Incentive

Our incentive system is built with two phases: (a) a compute-intensive primary evaluation applied to a small number of peers per communication round (b) a low-cost fast evaluation applied to a large number of peers in each round. Incentives are calculated in each communication window and the scores of each participant are updated locally by the validator. The overall behavior of peers and validators in the system is summarized in Algorithm 1.

3.1. Primary evaluation

Loss Rating The heart of the incentive mechanism attempts to judge the value of each pseudo-gradient contribution.

$$\text{LossScore}_p(\Delta_t^p, D) = L(\theta_t, D) - L(\theta_t - \beta\Delta_t^p, D) \tag{2}$$

where Δ_t^p is the pseudo-gradient from peer p at round t, D is a random subset of data from the training dataset, and β is a scaling factor. This essentially measures how much a peer's contribution decreases the loss. Naturally, poor gradients will lead to highly negative scores allowing the system to quickly downweight malicious contributors.

Note that since an individual contribution has a higher variance than the aggregated pseudo-gradients, the β will typically be set to a smaller value than the current learning rate. In practice, when using a learning rate scheduler, we found it was sufficient to set $\beta_t = c * \alpha_t$ where $c < 1$. Using a smaller value also allows us to reduce the noise in the LossScore. Specifically, stepping with too large a step size is more likely to lead to negative loss scores, and in our empirical observations inconsistent rankings between peers.

A significant issue with loss-based scores is that they are not consistent over time; indeed, even adjacent iterates can lead to very different scores for the same peer running the same strategy. This problem is exacerbated by the fact that practically, the validator cannot evaluate all peers' contributions at each communication round. On the other hand, we observed that at any given round, ranking based on LossScores correlated well with high quality contributions (e.g. peers processing more data achieved better LossScores). We thus utilize a rank-based rating system OpenSkill [8], which is well suited to estimating relative peer ranks under sparse evaluation.

In each evaluation round t, a random subset S of the K participating peers is chosen and ranked by their LossScore_p. Subsequently their OpenSkill rating, LossRating_p, is updated.

Proof of Computation A key challenge in a completely open permissionless system is that peers broadcast their pseudo-gradients to all peers on the network. This leads to several related problems where peers can avoid performing computation while achieving positive loss scores:

- Peer Copying - A peer attempts to copy a valid pseudo-gradient uploaded by another peer.
- Duplicating Contributions - A peer attempts to register multiple times on the network and upload identical pseudo-gradients.

Our proposed solution relies on assigning a unique subset of data D_t^p to peer p at any given round that must be used as part of its training data for that round. The validator then attempts to determine if the peer has actually performed training on this data by comparing the loss score on this data to the loss score on a random subset of data (already computed as part of the Loss Rating).

$$\mu_p = \gamma\mu_p + (1-\gamma) * \text{sign}(\text{LossScore}_p(\Delta_t^p, D_t^p) - \text{LossScore}_p(\Delta_t^p, D_t^{rand})) \tag{3}$$

Peers training on their assigned data D_t^p are expected to have lower loss on this data compared to their loss on a random data subset D_t^{rand}. This difference tends to yield $\mu_p > 0$ over time for compliant peers. Conversely, peers neglecting D_t^p are expected to have $\mu_p \simeq 0$. The resulting μ_p contributes to the peer's overall incentivization score, as detailed in equation 4.

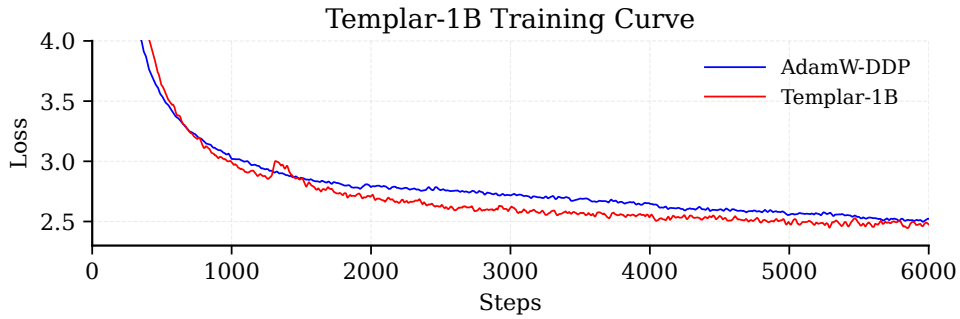

Figure 1: Templar-1B permissionless training curve, compared to a controlled AdamW baseline with the same number of peers and the default per worker batch size.

Signed Descent Following [12], we utilize the sign operation post-aggregation, which provides a number of practical benefits: (a) gradient norm control (b) ability to store the aggregation to allow fast checkpoint catchup. Specifically checkpointing can occur infrequently while catchup can be done through repeated application of the signed updates. For consistency the use of the sign is also done at evaluation.

3.2. Fast Evaluation

On a larger subset of peers, we perform a low cost evaluations including basic sanity checks and a score to estimate synchronization of the local model with the expected model on the validator.

Basic checks We penalize peers for the following: (a) not sending their pseudo-gradient within the specified put window. This is facilitated by our use of cloud-based storage in combination with the blockchain time (detailed in the next section), which provides a consistent global clock. (b) Not putting a pseudo-gradient at all (c) violating the format (e.g., submitting tensors with incorrect dimensions or data types).

Sync Score In each communication round, peers also send a very small set of their model parameters (2 values per tensor). The cost of this is negligible compared to the overall communication cost. Using this, the validator computes a synchronization score from the N communicated parameters as:

$$\text{SYNCSCORE} = \frac{1}{\alpha N} \sum_{i=1}^{N} \left| \theta_i^{(validator)} - \theta_i^{(peer)} \right|$$

Given that pseudo-gradient updates are signed post-aggregation (effectively quantizing updates by the learning rate α), the Sync Score provides a heuristic measure of how many update steps a peer's model diverges from the validator's. We use a threshold for this score as a filter (in practice, setting the threshold to 3).

Violation of either the basic checks or the sync score constraint leads to an additional penalty:

$$\phi_p = \begin{cases} 0.75 & \text{if peer } p \text{ fails any fast evaluation check} \\ 1 & \text{otherwise} \end{cases}$$

We apply the penalty directly to μ_p each time fast evaluation is performed:

$$\mu_p = \phi * \mu_p$$

This allows to rapidly degrade the score when a peer repeatedly fails the fast evaluation, as will be discussed in the next section. It also allows the peer to be quickly removed from the aggregation.

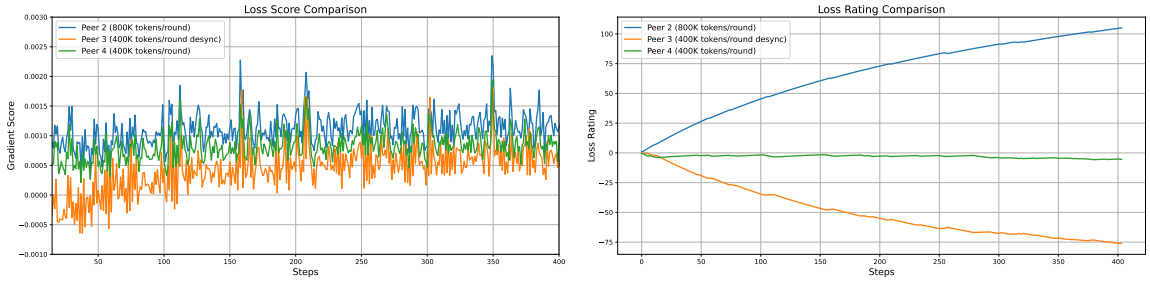

Figure 2: Simulating how LossScore and LossRating evolve for three peers: one processing more data, one desynchronized, and a baseline peer. while the loss score is highly variable from step to step, relative performance is consistent and the loss rating can quickly differentiate between peers exhibiting favorable behavior.

3.3. Putting it all together

Thus, our final pre-normalized score, PEERSCORE for peer p in round t is given by:

$$\text{PEERSCORE}_p^t = \mu_p * \text{LossRating}_p \tag{4}$$

Finally, the scores are normalized as follows:

$$x_p^{norm} = \frac{(\text{PEERSCORE}_p - \min \text{PEERSCORE})^c}{\sum_k (\text{PEERSCORE}_k - \min \text{PEERSCORE})^c} \tag{5}$$

The validator uses these normalized scores x_p^{norm} to assign incentives (that sum to 1) for all peers. These incentive values x_p^{norm} are posted by the validator to the blockchain and used to determine the monetary reward given by the protocol to each peer. In our current design, we use $c = 2$, with the goal to increase competition amongst peers. Indeed, the non-linear incentive is designed to encourage participants to register fewer high-performing peers versus many weaker peers. For example if a user has access to 10 GPUs it is preferred they take care of optimizing their configuration to produce a single high quality pseudo-gradient with all 10 GPU as opposed to registering 10 individual peers.

Finally, the weights w_p are used as part of the weighted aggregation in equation 1. For simplicity, in Templar-1B, we choose to set the aggregation weight of peers in the top G to $1/G$ and all others zero. This serves to encourage a smaller number of high quality peers with a large amount of compute behind them while also allowing for natural redundancy as when top peers become less reliable they are quickly swapped out in the aggregation with other high quality peers.

$$w_p^t = \begin{cases} 1/G & \text{if } x_p^t \in \text{Top-}G(x) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

During fast evaluation we ensure that the current top G peers are included in the fast evaluation set, such that they can be rapidly downgraded out of this set (and no longer impact the aggregation) if they begin to fail fast evaluation.

Byzantine fault tolerance A challenge in permissionless systems is that participants can violate the prescribed distributed algorithms either intentionally (e.g., by pseudo-gradient poisoning or rescaling) or due to a fault. This problem in the distributed learning literature is often referred to as byzantine fault tolerance, and problematic participants as byzantine workers [10, 20]. In an incentivized system, peers might also provide faulty contributions through good faith attempts to increase their incentives. Our incentive system can quickly reduce the weight of byzantine workers, removing them from the

aggregation. For example, peers sending poorly scaled contributions will often receive poor loss scores, and de-synchronized peers will be downweighted. To counteract this problem we further normalize the pseudo-gradients across contributors. This is discussed in more detail in Appendix A.

4. Results and Discussion

We deployed Gauntlet to train a 1.2B model for 20K communication rounds in a completely permissionless manner. The evaluation windows were set to be the same as the communication rounds. We used the FineWebEdu dataset[11]. Participants were provided a baseline training code through a public link, which they could adapt to their particular configuration. The baseline code targeted approximately 400,000 tokens per peer per iteration. However, the length of a communication round was set sufficiently long to allow more data to be processed on a single H100. We aggregated pseudo-gradients from the $G = 15$ top-scoring peers in each communication round. The validators were able to evaluate and compare 5 peers each communication round, updating their LossRating and μ_p score. The training loss curve is presented in Figure 1 and compared to an AdamW baseline (hyperparameters taken from [12], training with 15 peers processing 400K tokens per communication round. This represents a comparison to a centralized training algorithm not compatible with training over the internet. We note that, based on prior experiments, the DeMo algorithm roughly follows the convergence dynamics of Adam. Although we cannot measure the exact amount of data each peer processed, we observe that our convergence rate exceeds that of the Adam baseline in the first half of the run, suggesting participants were successfully incentivized to process more data or optimize their pseudo-gradients (e.g., by tuning local hyperparameters to instantaneously improve loss).

We also compare our downstream metrics against the baseline Adam trained model and the published results of [12], which was trained for the same number of iterations in Table 1. We see that our downstream metrics are competitive.

Table 1: Base model evaluation results on downstream benchmarks (zero-shot). We compare with published results of DeMo and custom training with AdamW using the same number of steps. TEMPLAR-1B token counts are estimated as number of tokens processed by participants is not controlled.

Model	Dataset	Tokens	HellaSwag	PIQA	ARC-E
TEMPLAR-1B	FineWebEdu	100B-200B	51.0	71.4	59.2
DEMO 1B [12]	Dolmo	100B	48.0	70.0	55.0
AdamW DDP 1B	FineWebEdu	120B	51.0	71.9	58.9

Simulating LossRating We performed controlled simulations of the Gauntlet incentive system focusing on identifying whether LossRating fulfills two basic properties: (a) peers training on more data get higher rating (b) peers who deviate from the global state get downweighted. The desynchronized peer was simulated by having the peer pause early on for 3 communication periods (thus representing a peer who is 3 steps behind) and then continue with the deviating model. In general our simulations showed that the LossRating can robustly detect both these scenarios, as observed in Figure 2. One peer training with 800K tokens per communication round significantly outperforms a peer training with the default 400K tokens per communication round. Similarly, a peer who is delayed by several steps rapidly begins to underperform.

5. Conclusion

We have introduced an incentive system for distributed permissionless pre-training of LLMs. We demonstrated that our system, when combined with a communication efficient distributed learning scheme, encourages high quality pseudo-gradient contributions from peers participating all over the world. We demonstrated that it can lead to effective convergence of a 1.2B model with completely permissionless peers participating on the bittensor blockchain. Future work will consider scaling to larger models and increasing the efficiency of the underlying communication algorithm.

References

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

[2] Arasu Arun, Adam St Arnaud, Alexey Titov, Brian Wilcox, Viktor Kolobaric, Marc Brinkmann, Oguzhan Ersoy, Ben Fielding, and Joseph Bonneau. Verde: Verification via refereed delegation for machine learning programs. *arXiv preprint arXiv:2502.19405*, 2025.

[3] Alexander Borzunov, Max Ryabinin, Tim Dettmers, Quentin Lhoest, Lucile Saulnier, Michael Diskin, Yacine Jernite, and Thomas Wolf. Training transformers together. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 335–342. PMLR, 2022.

[4] 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.

[5] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[6] Sami Jaghouar, Jack Min Ong, Manveer Basra, Fares Obeid, Jannik Straube, Michael Keiblinger, Elie Bakouch, Lucas Atkins, Maziyar Panahi, Charles Goddard, et al. Intellect-1 technical report. *arXiv preprint arXiv:2412.01152*, 2024.

[7] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056. IEEE, 2021.

[8] Vivek Joshy. Openskill: A faster asymmetric multi-team, multiplayer rating system. *arXiv preprint arXiv:2401.05451*, 2024.

[9] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pages 3252–3261. PMLR, 2019.

[10] Grigory Malinovsky, Peter Richtárik, Samuel Horváth, and Eduard Gorbunov. Byzantine robustness and partial participation can be achieved at once: Just clip gradient differences. *arXiv preprint arXiv:2311.14127*, 2023.

[11] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

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

[13] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022.

[14] Xun Qian, Peter Richtárik, and Tong Zhang. Error compensated distributed sgd can be accelerated. *Advances in Neural Information Processing Systems*, 34:30401–30413, 2021.

[15] 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.

[16] Lorenzo Sani, Alex Iacob, Zeyu Cao, Bill Marino, Yan Gao, Tomas Paulik, Wanru Zhao, William F Shen, Preslav Aleksandrov, Xinchi Qiu, et al. The future of large language model pre-training is federated. *arXiv preprint arXiv:2405.10853*, 2024.

[17] Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

[18] Jacob Steeves, Ala Shaabana, Yuqian Hu, Francois Luus, Sin Tai Liu, and Jacqueline Dawn Tasker-Steeves. Incentivizing intelligence: The bittensor approach, 2022. URL https://ai-secure.github.io/DMLW2022/assets/papers/6.pdf.

[19] 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 *International Conference on Machine Learning*, pages 36058–36076. PMLR, 2023.

[20] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

Appendix A. Byzantine Fault Tolerance

A challenge in permissionless systems is that participants can violate the prescribed distributed algorithms either intentionally (e.g. by pseudo-gradient poisoning or rescaling) or due to a fault. This problem in the distributed learning literature is often referred to as byzantine fault tolerance, and problematic participants as byzantine workers [10, 20]. In an incentivized system, peers might also inadvertently provide faulty contributions through good faith attempts to increase their incentives. Our incentive system can quickly reduce the weight of byzantine workers, removing them from the aggregation. For example, peers sending poorly scaled contributions will often receive poor loss scores, and peers that are not synchronized will be downweighted. Despite these measures, the system remains vulnerable to risks such as: (a) peers whose malicious behavior is not detected by the incentive mechanism, and (b) a single bad value sent before the peer can be downweighted. A simple example of (b) is a peer intentionally sending a pseudo-gradient with an excessively large magnitude, enough to disrupt the aggregation if it is included even once.

Such problems have been studied in the literature on byzantine fault tolerance in federated learning. However, many of the more sophisticated methods for addressing this problem either introduce significant overhead or slow down convergence [13, 20]. Some of the more practical approaches rely on gradient clipping [10].

In Templar-1B, we rely on the sign as in [12] as a final step in the aggregation, which has been found to help the DeMo method converge for LLMs. This naturally reduces the impact of direct attacks on the norm of the final update, but an individual peer can still dominate the aggregation by rescaling their pseudo-gradients. We thus rely on a simple strategy of normalizing the contributions q_t^k (see line 13 Algorithm 2) so that each peer contributes equally. As our aggregation is done in the DCT encoded domain, we also perform this normalization on the encoded vectors. Since we assume that each participant is training on an i.i.d. subset of the data, we do not anticipate large variations in the norms of valid pseudo-gradients. In practice, we observed that this approach significantly reduced the impact of byzantine peers while having no impact on convergence in the fully cooperative (simulated) setting.

Appendix B. Synchronous Model States Simplify Validation

Distributed learning methods can be broadly categorized into those that maintain the same model on all peers and those which allow models to diverge (e.g. asynchronous SGD, gossip-based methods). An advantage of methods that allow models to diverge is that they can typically support heterogeneous communication patterns more easily as well as overlapping communication; on the other hand, they are more challenging to debug and work with. We have found that, in the context of incentivization, synchronized model states are critical for allowing the validator to easily compare the contributions of peers. In an earlier experimental version of our system we allowed peers and validators to partially diverge, but this creates significant issues, and differences in evaluation of the loss are challenging to attribute to model divergences between validator and peer states. Furthermore, even in asynchronous methods, peers need to attempt to stay tightly coupled together, thus encouraging synchronization is still important. This, however, is more challenging to do without a single global reference state that can be available for the validator.

Appendix C. Cloud-Based Communication

A novel aspect of our system deployment is the use of cloud-based communication backend. Peers and validators in the network communicate using S3-compliant storage buckets. Each peer on the network creates their own bucket and posts the read-access keys to the blockchain, making them visible to other peers and validators. Broadcasting pseudo-gradients is done by simply writing to a local bucket. This has several advantages:

- Peer-to-peer network complexities, such as firewall configuration, are avoided.
- Pseudo-gradient contributions can be easily tracked and robustly timestamped.
- Cloud providers, such as Cloudflare, have globally distributed networks which can often ensure competitive upload and download times for participants worldwide.

A disadvantage is that all communication must pass through the cloud provider, making the system limited to the reliability of the cloud provider. Our incentive mechanism encourages peers to optimize their configuration to work robustly with the cloud provider.

Appendix D. Coordinated Aggregation

Although peers can freely modify their local implementation, the incentive mechanism pushes peers to perform the same aggregation as specified by the validator (e.g., using the same set of peers G specified by the validator) in order to stay synchronized to the validator state $\theta_t^{validator}$. This allows the system to easily propagate various control mechanisms. For instance, a time window for communication is specified and any pseudo-gradients arriving outside of this time window are ignored by the validator, and thus the peers should also ignore them. Similarly, peers are encouraged to use the peer scores w_p in the same way as the validator (equation 6).

Appendix E. Validator Consensus and Stake

In a decentralized system, the evaluation of incentives must also be decentralized. On the Bittensor blockchain, this is achieved through the use of multiple validators who are required to provide stake—an amount of tokens placed at risk as a form of economic commitment. Validators participate in the evaluation process and are subject to penalties for dishonest or faulty behavior. A set of validators typically operates under the Yuma consensus protocol [18], which combines the incentives x_p^{norm} from different validators. A full description of the Bittensor validator and consensus mechanisms is beyond the scope of this technical report[1]. For the sake of simplicity in the current implementation of the protocol, the highest staked validator is chosen to provide the location of consistent checkpoints (for peers joining later or restarting) and the list of top-G peers. However, even these can be decentralized in future iterations.

1. https://docs.bittensor.com/yuma-consensus

Algorithm 1: Gauntlet Incentive Scheme

Input: number of peers K, iterations T, learning rate α, EMA decay β, mixing γ, top-G size G, random seed

Initialize model parameters θ_0
Initialize generalization scores $\mu_p \leftarrow 0$ for all p
Initialize peer scores $\text{PEERSCORE}_p \leftarrow 0$ for all p

Peers

for $t \leftarrow 0$ **to** $T-1$ **do**
  foreach $p \in \{1, \ldots, K\}$ **do**
    $D_t^p \leftarrow \texttt{SelectData}(\text{seed}, p, t)$
    $\Delta_t^p \leftarrow \texttt{PseudoGradient}(D_t^p, \theta_t)$
    $\texttt{Broadcast}\ (\Delta_t^p)$
    // Local aggregation and update
    $w \leftarrow \texttt{SelectTopG}\big(\{\text{PEERSCORE}_w\}_{w=1}^K, G\big)$
    $\Delta_t^{\text{agg}} \leftarrow \texttt{Aggregate}\big(\{\Delta_t^w \mid w_p > 0\}\big)$
    $\theta_{t+1}^p \leftarrow \theta_t - \alpha\, \Delta_t^{\text{agg}}$
  end
end

Validator

for $t \leftarrow 0$ **to** $T-1$ **do**
  $S_t \leftarrow \texttt{SelectRandomPeers}(v)$ // Small set of peers $|S_t| \ll K$
  // Update scores for a larger set using filtering
  $F_t \leftarrow \texttt{SelectPeersForFiltering}(F)$
  foreach $p \in F_t$ **do**
    $\phi_p \leftarrow \texttt{FastEvaluation}(p)$ // Check if peer passes fast eval
    $\mu_p \leftarrow \phi_p \cdot \mu_p$
  end
  foreach $p \in S_t$ **do**
    $\theta'_p \leftarrow \theta_t - \alpha\, \texttt{Sign}(\Delta_t^p)$
    $D_t^p \leftarrow \texttt{SelectData}(\text{seed}, p, t)$ // Evaluate on peer assigned data
    $\delta_p^{\text{assigned}} \leftarrow L(\theta_t, D_t^p) - L(\theta'_p, D_t^p)$
    // Evaluate on random data
    $D_t^{\text{rand}} \leftarrow \texttt{UnassignedData}(p, t)$
    $\delta_p^{\text{rand}} \leftarrow L(\theta_t, D_t^{\text{rand}}) - L(\theta'_p, D_t^{\text{rand}})$
  end
  $\{\text{LossRating}_p\}_{p \in S_t} \leftarrow \texttt{OpenSkillMatch}\big(S_t, \{\delta_p^{\text{rand}}\}\big)$ // Rank peers in S_t
  foreach $p \in S_t$ **do**
    $\mu_p \leftarrow \gamma \cdot \mu_p + (1-\gamma) \cdot \text{sign}\big(\delta_p^{\text{assigned}} - \delta_p^{\text{rand}}\big)$
    $\text{PEERSCORE}_p \leftarrow \text{LossRating}_p \cdot \mu_p$
  end
  $w \leftarrow \texttt{SelectTopG}\big(\{\text{PEERSCORE}_w\}_{w=1}^K, G\big)$
  $\Delta_t^{\text{agg}} \leftarrow \texttt{Aggregate}\big(\{\Delta_t^w \mid w_p > 0\}\big)$
  $\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \Delta_t^{\text{agg}}$
end

Algorithm 2: DeMo: PseudoGradient and Aggregation

Input: Current model θ_t, local batch D_t^p, momentum buffer e_t, compression hyperparameters s, k, error-feedback decay β

Function `DeMoPseudoGradient`(θ_t, D_t^p, e_t)

- $\tilde{g}_t \leftarrow$ `LocalStochasticGradient`(θ_t, D_t^p);
- $e_t \leftarrow \beta \cdot e_t + \tilde{g}_t$; `// Apply error feedback`
- $q_t \leftarrow$ `DCTEncode`(e_t); `// DCT on chunked tensors`
- $\hat{q}_t \leftarrow$ `TopKCompress`(q_t, s, k);
- $\hat{z}_t \leftarrow$ `DCTDecode`$(\hat{q}_t)$;
- $e_t \leftarrow e_t - \hat{z}_t$; `// Update error feedback`
- **return** q_t; `// Send compressed pseudo-gradient`

Function `DeMoAggregation`$(\{q_t^1, \ldots, q_t^G\})$

- **foreach** $q_t^k \in \{q_t^1, \ldots, q_t^G\}$ **do**
  - $q_t^k \leftarrow \dfrac{q_t^k}{\|q_t^k\|_2}$; `// Robustness to individual peer norms`
- **end**
- $Q_t \leftarrow$ `AggregateCompressed`$(\{q_t^k\})$; `// Weighted average of updates`
- $\Delta_t \leftarrow$ `DCTDecode`(Q_t); `// Decode aggregated update`
- $\Delta_t \leftarrow$ `Sign`(Δ_t); `// Apply Signum`
- **return** Δ_t