Towards the Decentralized and Collaborative Training and Inference of Large Models

Aarush Gupta

May 2024

Abstract

Modern natural language tasks increasingly rely on Large Language Models (LLMs), which often contain over one hundred billion parameters, making them impractical for use on consumer hardware. To address this challenge, we propose a novel decentralized and collaborative training and inference framework that leverages a peer-to-peer network to democratize access to these models, akin to Bitcoin. Our approach decomposes the computation of neural networks into smaller segments, specifically by splitting the forward pass over parameters and the backward pass by using the computed Directed Acyclic Graph (DAG). This methodology enables a sequence of computations across the network, facilitating weight aggregation through neighboring peers. Additionally, we introduce a set of optimizations tailored to the specific requirements of LLMs, including the parallelization of the attention mechanism. Practical aspects such as disk representation and verification within our untrusted system are also considered. Our contributions include a scalable and efficient distributed framework for LLMs. This work has significant implications for making advanced and large models more accessible and collaborative, potentially transforming fields that rely heavily on natural language processing by providing a more equitable distribution of computational resources.[1]

1 Introduction

The advent of deep learning and the recent development of expansive LLMs have revealed that performance tends to scale proportionally with an increase in model size [16]. This has driven the creation of modern models with hundreds of billions of parameters [6], such as the Falcon-180B, a causal decoder-only language model with 180 billion parameters [2]. However, these models, due to their considerable density, necessitate substantial memory and computational resources. For instance, Falcon-180B requires 640GB of memory at 16-bit weights and 1280GB for LoRA [11] fine-tuning.

These substantial constraints pose numerous challenges, particularly hindering the advancement of open and accessible science due to the immense computational requirements. Efforts have been made to address these issues through the quantization of pre-trained models [12] and faster, more efficient attention layers [8]. However, these approaches result in the loss of learned features or necessitate the laborious design of kernels at the GPU level. Consequently, there is a pressing need for a more optimized training and inference regime, as traditional approaches that work on specific components often come with significant drawbacks.

In this whitepaper, we propose a novel training and inference strategy, drawing inspiration from the peer-to-peer architecture of Bitcoin [15], parallel computation methodologies for LLMs

[1] The webpage for the foundation established to build this whitepaper is available at this https URL.

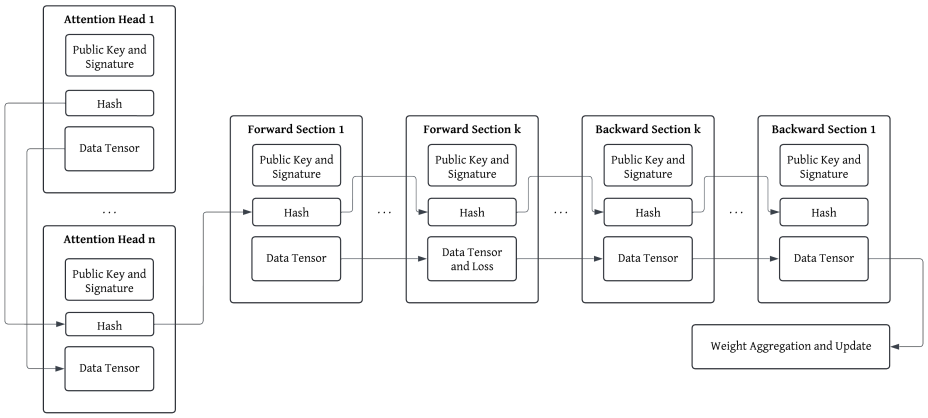

Figure 1: Complete step of training a transformer in our system.

[14], and the concept of Proof-of-Learning [13]. Our approach conceptualizes training an LLM based on the transformer architecture [18] as a linked list ledger, similar to Bitcoin, by partitioning the forward and backpropagation passes. This strategy allows us to preserve the optimizations inherent in Bitcoin's ledger system and network while introducing an incentive mechanism for participants who contribute GPUs to the system. It also enables us to maintain efficient computation at scale. We then discuss the advantages of our framework, including anonymous inference, democratized and decentralized training, and the promotion of open science.

2 Model

We begin with mathematical models of the nature of our novel decentralized framework as shown in Figure 1.

2.1 Splitting Computation

In a decentralized setting, we must represent the neural network in a scalable format that distributes computation across the peer network. Bitcoin represents its list of transactions as a linked list ledger. We inherit this and modify it to fit our task of peer-to-peer training and inference. We begin with the forward pass of any given input $\mathbf{x}$, which is computed for both training and inference. If we let f denote a given layer of a sequential neural network, we can represent any network as $\{f_1, f_2, \ldots, f_n\}$, where n is the total number of layers. Therefore, a forward pass of a conventionally trained neural network can be represented as:

$$\mathbf{h}_1 = f_1(\mathbf{x}), \quad \mathbf{h}_2 = f_2(\mathbf{h}_1), \quad \ldots, \quad \mathbf{h}_n = f_n(\mathbf{h}_{n-1}) \tag{1}$$

To distribute compute across peers, we can trivially split this forward pass into computationally smaller sections. We partition the layers into k sections. We define S_i as the i-th section containing layers with indices $\{i_1, i_2, \ldots, i_p\}$. Hence, each section S_i computes as:

$$\mathbf{h}_{S_i} = f_{i_p}(\ldots f_{i_2}(f_{i_1}(\mathbf{h}_{S_{i-1}}))\ldots) \tag{2}$$

This assumes that each k section will require equal compute and that all n layers have equal parameters. This is often not true. Therefore, we must partition the layers into k sections

such that each section has an approximately equal number of parameters. Let $P(f_i)$ denote the number of parameters in layer f_i. The total number of parameters P_{total} in the network can be expressed as:

$$P_{\text{total}} = \sum_{i=1}^{n} P(f_i) \tag{3}$$

Hence, we aim to divide the layers into k sections $\{S_1, S_2, \ldots, S_k\}$ such that the sum of parameters in each section is approximately $\frac{P_{\text{total}}}{k}$. Each section S_i still computes as defined in (2). The condition for splitting is defined as:

$$\sum_{j \in S_i} P(f_j) \approx \frac{P_{\text{total}}}{k} \tag{4}$$

This can be achieved by an analysis of the model architecture before runtime. Our proposed solution is to compute weights in the same layer in parallel across nodes and step through layers sequentially in that fashion. Through this methodology, we would achieve the list of computations as shown in Figure 1, wherein a given section would contain a computation for a layer and clusters of sections would form the computation for the full layer.

The added benefit of partitioning by parameters rather than layers is that it allows each block in the resulting list of computations to be nearly equal in size. This consequently allows us to make assumptions and optimizations on disk representations of the model and in networking between peers. The last peer to compute section S_k would also compute the loss, for it is not computationally expensive. Any validation phases of training can be run with this same framework, swapping out input data sources as necessary.

As for computing the backpropagation of the model, we can employ the computed DAG. Each node in the DAG corresponds to a layer, and edges represent the dependency of gradient computations. To split the backward pass, we partition the DAG into the k sections delimited when splitting the forward pass. Each section S_i will compute the gradients for its layers and pass the gradient information to the previous section. If we let $\nabla \mathcal{L}_{S_i}$ denote the gradients computed in section S_i, we arrive at the following representation of the backpropagation pass for section S_i as:

$$\nabla \mathbf{h}_{S_{i-1}} = \sum_{j \in S_i} \nabla_{\mathbf{h}_{S_{i-1}}} f_j(\mathbf{h}_{S_{i-1}}) \cdot \nabla \mathcal{L}_{S_i} \tag{5}$$

This enables us to rely on the split over parameters defined in (4) and adapt it for computing gradients over the same sections.

2.2 Weight Aggregation

Due to the decentralized nature of our process, aggregating the weights is crucial to ensure peers maintain the same model across the network. We begin with a model of training a neural network with gradient descent. If we let $\mathbf{w}$ represent the weights of the neural network, we arrive at the following conventional update rule:

$$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \eta \nabla_{\mathbf{w}_i} \mathcal{L}(\mathbf{w}_i^{(t)}, \mathcal{D}_i) \tag{6}$$

where η is the learning rate, and $\mathcal{L}$ is the loss function. However, our goal is to decentralize this training among numerous peers. If we let $\mathcal{N}(i)$ represent the set of peers connected to peer i, we achieve a model of decentralized communication and aggregation of weights from neighbors as:

$$\mathbf{w}_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{w}_j^{(t)} \tag{7}$$

where α_{ij} are the aggregation weights satisfying $\sum_{j \in \mathcal{N}(i)} \alpha_{ij} = 1$. If we combine (6) and (7), we arrive at the following update rule for given peer i and dataset $\mathcal{D}$:

$$\mathbf{w}_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} (\mathbf{w}_j^{(t)} - \eta \nabla_{\mathbf{w}_j} \mathcal{L}(\mathbf{w}_j^{(t)}, \mathcal{D}_j)) \tag{8}$$

Through this framework, we can express the update rule for optimizing weights in a decentralized manner.

2.3 Optimizations

Our work thus far addresses general neural networks. However, we can optimize our process for the more prevalent LLM and transformer architecture. As discussed in [14], the attention mechanism presented in the original transformer paper can be parallelized to enhance its computational efficiency, which is particularly crucial in our decentralized setting. We begin by recalling multi-head attention as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O \tag{9}$$

with each head computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{10}$$

where W_i^Q, W_i^K, W_i^V are learned projection matrices. Note that the computation for each individual head can be done on separate nodes and are not interdependent on one another. The nature of the attention mechanism also allows parallelization across the sequence length. Let n be the length of the input sequence. We can split the sequence into k segments of equal length and distribute these to different nodes for parallel computation. For segment S_i of the input sequence, the attention computation can be thereby be expressed by computing the learned project matrices W_i^Q, W_i^K, W_i^V for each segment rather than the complete input $\mathbf{x}$. Each segment's attention computation is independent of the others and can be distributed across k nodes, computed, then aggregated, as described in (9).

3 Additional Notes

In this section, we discuss some additional considerations to be made in the practical implementation of this process in the real world.

3.1 Disk Representation

Our training and inference regime is inspired by the Bitcoin system and therefore, we inherit the data structure representation of our "ledger" as a Merkle Tree as well. Recall that a Merkle Tree works on hashes. Therefore, we must similarly hash each block of computation. We delimit a block to represent one unit of computation, whether that be of the forward pass, backward pass, or attention. These hashes form the leaf nodes of the tree and internal nodes are constructed by concatenating the hashes of their child nodes and block data. For example, for leaf nodes L_1 and L_2, the hash of the parent node would be computed as:

$$H_{\text{parent}} = H(H(L_1) \| H(L_2)) \tag{11}$$

Therefore, the root of the Merkle Tree would represent the combined hash of all subsequent computations.

3.2 Verification

To trace those who computed specific blocks, the public key of that party will be stored with the hash in the Merkle Tree. To verify the integrity of a specific section of the computation, we only require the hashes along the path from the leaf node representing that section to the root of the Merkle Tree. This allows efficient verification without needing to recompute the entire model. Bad actors may attempt to poison model training with bad data sources alongside numerous other potential attacks. Proof-of-Learning, as outline in [13], uses metadata from the gradient-based optimization processes to construct certifications of work, which can be used to verify whether productive learning occured.

This would enable the network to verify: a) whether a given list of computations is valid. This would be crucial when a new node is introduced into the system and requires updates, or when a node comes online after a period where it was not being updated, and b) whether a new block of computation is valid before reaching consensus.

We maintain the optimizations utilized by Bitcoin in its implementation of the Merkle Tree, such as storing only the root block's hash with every new block and pruning redundant blocks. This approach supports the conclusion that block headers can be stored in memory, leveraging the proliferation of available memory, year-over-year.

3.3 Dataset

The dataset used to train the model can be stored in another distributed file store like IPFS [3], perhaps chunked for decentralized processing before runtime by the network or another research group. Our work concerns with the training and inference of a given model and not necessarily data and its implementation is left as an open question.

3.4 Incentive

Alongside our process for training and inference of large models, we introduce a incentive over our network. An incentive would motivate nodes to support the network, expending GPU time and electricity in exchange for value. It would also maintain a steady distribution of value during training.

Hence, we also introduce a cryptocurrency to fuel our network. Its implementation can follow or be built on the numerous chains [7] that already exist. We envision that this currency will be given value as typical cryptocurrencies are, but have added value in that they enable inference. To reward early nodes in the system during training, each added block will grant the computing party a reward proportional to compute, which can be expended back into the network when inference is possible. Computational work will be generally equal across blocks, given that we can split computation equally based on parameters as described in Section 2.1. Therefore, it makes logical sense to reward one coin per block computed and let the free market economy formed dictate the monetary value.

However, in the spirit of open science, the resulting model's weights will be made open source. This ensures that other researchers can utilize the trained model beyond the API we develop for inference, allowing them to investigate the internal state or modify parameters.

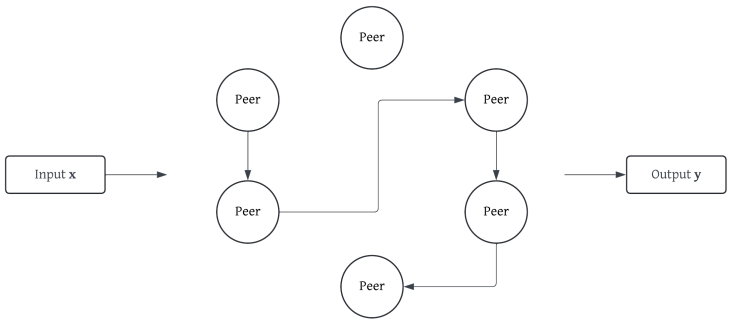

Figure 2: Overview of inference in our network. Note how, after attention is computed, most peers remain idle as the forward pass is computed sequentially. Also note the anonymity provided by the system, for no one party can trace a request to an IP address.

4 Discussion

4.1 Attacks and Limitations

Similar to any untrusted decentralized system, our network is not without potential attack vectors and general limitations. However, due to the inherent difficulty and high computational cost of computing blocks, we do not foresee any sustainable attacks against the network. We grant the longest waiting party the ability to compute a block, meaning an attacker would need supercomputer-level capabilities to compute and append blocks faster than the honest nodes. Given that LLMs are notorious for requiring large amounts of data [10], a genuine attempt to attack the network would necessitate the ability to process a substantial portion of the dataset more rapidly than thousands of other nodes.

However, a key limitation of our network lies in the nature of computing the forward and backward sections as detailed in Section 2.1. After computing parallel attention, the individual sections for the passes must be computed sequentially, meaning that only one node will be performing the computation at any given time. This limitation should, in a sense, encourage more nodes to join the network, as systems can be utilized for other tasks while waiting for their turn to compute.

Perhaps another limitation to note is communication bandwidth between peers. We envision nodes in the system will have high bandwidth, but the transfer of matrices across the network remains an issue. A possible solution would be to compress matrices as detailed in [17] or quantizing directly on activations as described in [1].

4.2 Anonymity

The decentralized nature of our network lends itself well to anonymity, partly inspired by the Tor project's network [9]. During inference, sections of computation will traverse the network in a nearly random manner, thereby concealing the identity of the initiating party by hiding their IP address. This ensures true anonymity and mitigates data mining issues. However, data privacy remains a concern. Since every computation will be stored across the network, the inputs and outputs of the model will also be publicly accessible. To address this, private organizations or research groups with data privacy concerns are encouraged to perform inference on their own hardware.

4.3 Related Work

We acknowledge the Petals [5] model, which similarly attempted inference and fine-tuning of large models, but not training from scratch, and the Bittensor [4] project, which attempts the same work as us, but by ranking peers on intrinsic informational value.

5 Conclusion

In this work, we proposed a system for the distributed training and inference of large models, particularly LLMs, without relying on trust. We started with a mathematical model of training a neural network and adapting it to a decentralized system. We split the forward and backward pass based on paramters and computation and formed a linked list system akin to Bitcoin. We then followed with weight aggregation and optimizations specific to LLMs and transformers by parallizing attention across the network. We considered additional notes, including a disk representation of the model, verification of productive and trustworthy learning, datasets, and incentives for participating in the network. We then discussed attacks, limitations, and anonymity. Our network is robust in its unstructured simplicity and can work without coordination by any central party. It is fault tolerant and processing computations can be done even if nodes fail. With this whitepaper, we hope to increase the accessibility of huge and performant LLMs and enable parties without infrastructure to use and investigate these models.

References

[1] Jue Wang et al. "Fine-tuning Language Models over Slow Networks using Activation Compression with Guarantees". In: (2022). arXiv: 2206.01299.

[2] Ebtesam Almazrouei et al. *The Falcon Series of Language Models: Towards Open Frontier Models*. 2023.

[3] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561.

[4] "BitTensor: An Intermodel Intelligence Measure". In: *CoRR* abs/2003.03917 (2020). Withdrawn. arXiv: 2003.03917.

[5] Alexander Borzunov et al. *Petals: Collaborative Inference and Fine-tuning of Large Models*. 2023. arXiv: 2209.01188.

[6] Tom B Brown et al. "Language models are few-shot learners". In: (2020). arXiv: 2005.14165.

[7] Vitalik Buterin. *Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform*. 2013.

[8] Tri Dao et al. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". In: *CoRR* abs/2205.14135 (2022).

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *Proceedings of the 13th Usenix Security Symposium*. 2004.

[10] Jordan Hoffmann et al. "Training Compute-Optimal Large Language Models". In: (2022). arXiv: 2203.15556 [cs.CL].

[11] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: (2021). arXiv: 2106.09685.

[12] B. Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713. DOI: 10.1109/CVPR.2018.00286.

[13] Hengrui Jia et al. "Proof-of-Learning: Definitions and Practice". In: *SP*. IEEE, 2021, pp. 1039–1056. ISBN: 978-1-7281-8934-5.

[14] Julian Richard Medina and Jugal Kalita. "Parallel Attention Mechanisms in Neural Machine Translation". In: *ICMLA*. IEEE, 2018, pp. 547–552. ISBN: 978-1-5386-6805-4.

[15] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009.

[16] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. 2019.

[17] Rajarshi Saha, Varun Srivastava, and Mert Pilanci. "Matrix Compression via Randomized Low Rank and Low Precision Factorization". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[18] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.