P2P-LLM: Collaborative LLM Inference

Andrew Boessen BOESSENA@BC.EDU

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1 Introduction

In recent years, large language models (LLMs) have demonstrated remarkable capabilities across diverse tasks, exhibiting unprecedented rates of improvement within a relatively brief developmental timeline. This extraordinary progress can be primarily attributed to the immense computational resources allocated to training these models on datasets comprising trillions of tokens, resulting in systems that subsequently demand substantial computing power for deployment. The computational intensity of these models has resulted in significant investment, with billions of dollars directed toward the construction of specialized super-clusters housing hundreds of thousands of GPUs dedicated to the training and serving of LLMs. Given the substantial memory and computational requirements of state-of-the-art LLMs, most personal computing devices lack the necessary specifications to run or train these models independently. Consequently, access to LLM capabilities is predominantly channeled through services offered by large technology corporations, which may present concerns regarding privacy, autonomy, and accessibility.

Crowdsourcing methodologies offer a promising alternative to address these limitations. The crowdsourcing approach has demonstrated considerable success across various domains, including knowledge aggregation (Wikipedia), real-time traffic information collection (Waze), and venture capital formation (Kickstarter). These same foundational principles can be effectively applied to computational resource allocation and utilization.

My work explores fully decentralized crowd computing architectures, where users establish a peer-to-peer network in which each participant can leverage the collective computational resources for individual tasks. This structure shares conceptual similarities with BitTorrent file-sharing networks. While current state-of-the-art language models exceed the capacity of individual personal computers, the aggregation of distributed computational resources through a decentralized network could potentially enable users to host their own LLMs, thereby democratizing access to advanced artificial intelligence capabilities.

2 Realted Work

2.1 Crowd Computing

The idea of pooling compute power from many computers to form a distributed supercomputer is not new. This concept of Crowd Computing has been used ever since large networks of computer were formed. The predominant use of this method is to fascilitate academic research. Researchers publish problem that that are working on, and anyone can help contribute by volunteering to donate computing power. Projects that have successfully used crowd computing include SETI@Home (Anderson et al., 2002) and Einstein@Home (Steltner et al., 2021). This approach has proven to be successful in the past, but it has become less popular as the computational power of computers increased and access to supercomputers increased. However, the use of idle compute power still has untapped applications, and the principles from crowd computing can be applied to modern tasks like LLM inference.

2.2 P2P Networks

LLM inference is a perfect candidate for P2P network because it has technical requirments that prohibit most personal computers from being able to run large models with billions of parameters, and it can easilly be split into small chunks that can be run on several computers at the same time. BitWorker (Durand et al., 2015) is a protocol designed to compute any parallel computing task across a peer-to-peer network. Although it is possible to apply LLM inference to this protocol, BitWorker is designed for general tasks and would not able to efficiently scale to the demands of many users. The most notable research being done in this area is Petals (Borzunov et al., 2023) which is a BitTorrent like peer-to-peer network for LLM inference and finetuning. Petals creates a network with client and servers. Clients send requests that will be processed by the servers on the network. The model is split into multiple sequential blocks of layers, and each server in the network is capable of serving one or multiple of these blocks. In order to run an inference step, a client will find a optimal path of servers in the network thats forms the complete model and minimizes time required. In my work I explore a similar approach with a network of distributed computers that serve a specific part of the whole LLM, but I expand upon this and add incentive mechanisms that reward nodes for donating their computational power. This incentive mechanism also fucilitates optimal allocation of nodes to maximize throughput, by each node maximizing expected revenue individually.

3 Peer-to-Peer Network

Fundamentally, a peer-to-peer network is a decentralized system where each devices acts as a client and a sever. Each devices can send and recieve data which eliminates the need for a centralized server. The central benefit of a P2P network is that recources are pooled and shared between peers, so the combined resources available to an individual peer is greater than that nodes own resources. A peer-to-peer network can be built to share any type of resource. In the case of BitTorrent, bandwidth and files are shared between peers. For the case of LLM inference, peers will need to ba able to share bandwidth, data, and computational power.

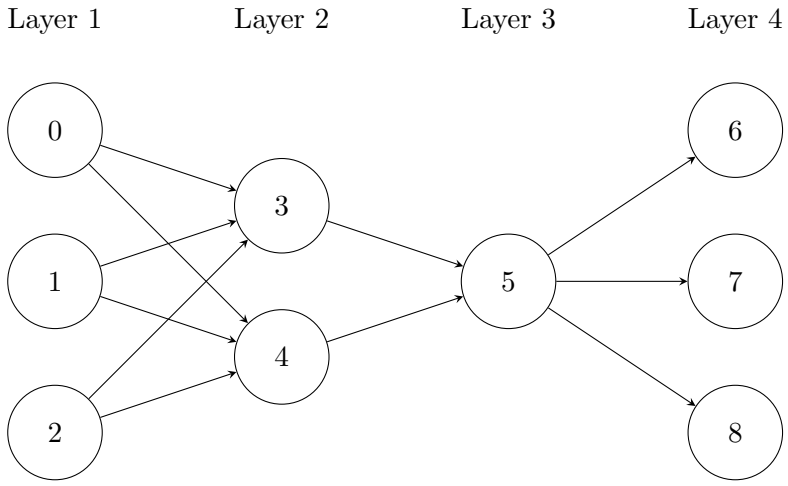


Figure 1: P2P-LLM Network Directed Graph

A peer-to-peer network is defined as having n nodes in the network. There is an LLM model with m total layers which will be devided between nodes. Each node N in the network is a tuple $N(\ell, l, c, p)$ where $\ell_i = (x, y) : x, y \in [0, m); y > x$ is the range of layers the node is able to serve, l_{ij} is the network latency between nodes i and j, c_i is the computational cost of the node, measured in miliseconds, and p_i is the price the nodes charges for its compute.

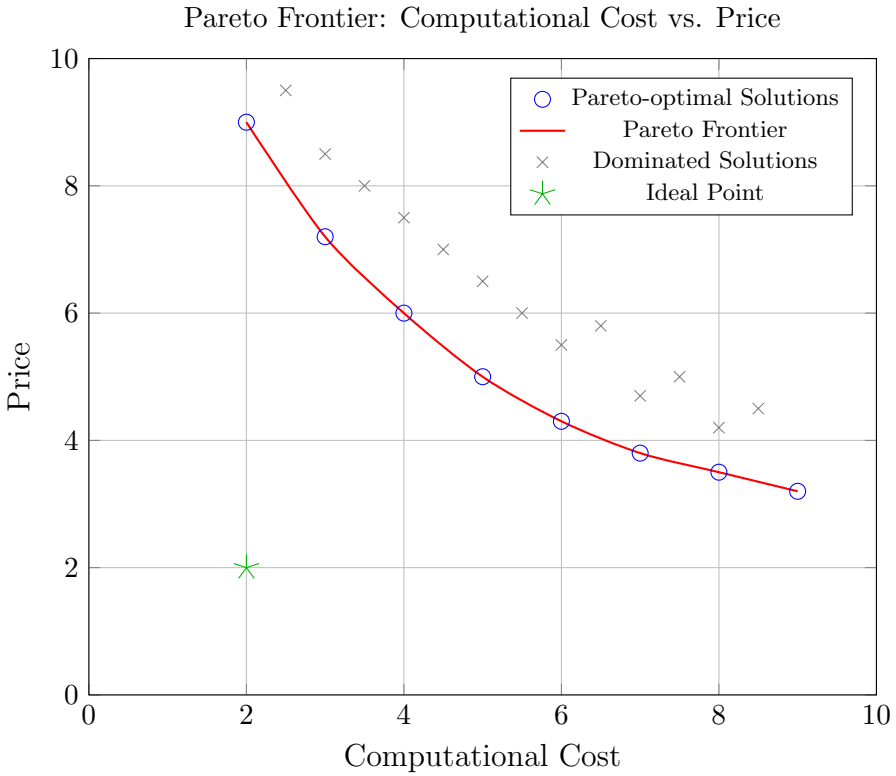
3.1 Path Finding

In order to use the network to run the LLM, a node must first find a sequence of nodes in the graph that form a complete model. Because the LLM layers must be computed sequentially, the node must find a path through the network that ideally will compute the LLM in the least amount of time possible and at the lowest price. This problem can be solved by viewing the network as a directed graph where weight correspond to the computational cost of nodes. The edges of the graph correspond the compatability between the layers that each node in serving. Specifically, every node is connected to all nodes that are serving the consecutive layer of the model. A vizualization of this directed graph is shown in Figure 1.

Due to this property, the P2P network graph is a directed acyclical graph (DAG). This means that there are efficient algorithms to find the shortest path between two nodes. The first step is to perform a topological sort on the nodes in the network. This is an ordering of the nodes such that all neighbors of a node come after the node in the ordering. For example, a topological ordering of the DAG in Figure 1 is $[0, 1, 2, 3, 4, 5, 6, 7, 8]$. The algorithm for topological sort is shown in Appendix A. With the topological order, dynamic programming can be used to find the shortest path in the network. This algorithm will iterate over the ordering, and search all the nodes neighbors for the one with the lowest weight. The full algorithm is shown in Appendix B. Most importantly, both the topological sort and path finding algorithm run in linear time with a computational complexity of $O(|V| + |E|)$.

3.2 Pareto Effeciency

The optimal path is well defined when weights for nodes are clearly defined, but there is not a clear way to define weights for nodes in this case. In the P2P-LLM network there are two competing criterea that define the weight of node: computational cost and price. The algorithm described above uses a single objective, while the optimal path in the network is a multi-objective optimization problem.



3.3 Revenue Optimization

3.4 Network Equilibrium

3.5 Strategy-Proofness

4 Cryptoeconomics

4.1 Blockchain

4.2 Contracts

5 Experiment Setup

6 Results

Appendix A. Topological Sort

Algorithm 1 Topological Sort of a Directed Acyclic Graph (DAG)

Require: A directed acyclic graph $G = (V, E)$

Ensure: A linear ordering of vertices such that for every directed edge (u, v), vertex u comes before vertex v

```
result[]                                   ▷ An empty list to store the sorted vertices
visited[v] →false for all v ∈ V            ▷ A map to track visited vertices
temp[v] → false for allv ∈ V               ▷ A map to track vertices in the current recursion stack
for each vertex v ∈ V do
    if visited[v] = false then
        DFS-Visit(G, v, visited, temp, result)
    end if
end for
return Reverse(result)
function DFS-Visit(G, u, visited, temp, result)
    temp[u] ← true                         ▷ Mark current vertex as being processed
    for each vertex v such that (u, v) ∈ E do
        DFS-Visit(G, v, visited, temp, result)
    end for
    temp[u] ← false                        ▷ Mark u as processed
    visited[u] ← true                      ▷ Mark u as visited
    Append u to result
end function
```

Appendix B. Shortest Path

References

David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11): 56–61, November 2002. ISSN 0001-0782. doi: 10.1145/581571.581573. URL `https://doi.org/10.1145/581571.581573`.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models, 2023. URL `https://arxiv.org/abs/2209.01188`.

Arnaud Durand, Mikael Gasparyan, Thomas Rouvinez, Imad Rafic Aad, Torsten Braun, and Tuan Anh Trinh. Bitworker, a decentralized distributed computing system based on bittorrent, May 2015. URL `https://boris-portal.unibe.ch/handle/20.500.12422/134610`.

B. Steltner, M. A. Papa, H.-B. Eggenstein, B. Allen, V. Dergachev, R. Prix, B. Machenschalk, S. Walsh, S. J. Zhu, O. Behnke, and S. Kwang. Einstein@home all-sky search

Algorithm 2 Shortest Path in a DAG

Require: Graph $G = (V, E)$ with edge weights w, source vertex s, target vertex t, topological ordering $topo[]$ of G

Ensure: Length of shortest path from s to t and the path itself

Initialize $dist[v] \leftarrow \infty$ for all $v \in V$
Initialize $prev[v] \leftarrow$ nil for all $v \in V$
$dist[s] \leftarrow 0$
for each vertex u in topological order $topo[]$ **do**
    for each outgoing edge $(u, v) \in E$ **do**
        if $dist[v] > dist[u] + w(u, v)$ **then**
            $dist[v] \leftarrow dist[u] + w(u, v)$
            $prev[v] \leftarrow u$
        end if
    end for
end for
$path \leftarrow \emptyset$ ▷ Reconstruct the shortest path
$curr \leftarrow t$
while $curr \neq$ nil **do**
    Prepend $curr$ to $path$
    $curr \leftarrow prev[curr]$
end while
return $dist[t]$, $path$

for continuous gravitational waves in ligo o2 public data. *The Astrophysical Journal*, 909(1):79, March 2021. ISSN 1538-4357. doi: 10.3847/1538-4357/abc7c9. URL http://dx.doi.org/10.3847/1538-4357/abc7c9.