# Optimizing Resource Allocation for Geographically-Distributed Inference by Large Language Models

Tingyang Sun[1], Ting He[1*], Bo Ji[2], Parimal Parag[3]
[1]Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA
[2]Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
[3]Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, India

## 1. INTRODUCTION

Large language models (LLMs) have produced exceptional performance in many artificial intelligence tasks [2]. Modern LLMs pre-trained on large datasets have demonstrated promising utility for diverse applications [2]. However, to achieve state-of-the-art accuracy, such models often need to contain more than 50 billion parameters, which makes it expensive to work with these models due to the requirement of expensive hardware such as high-end GPUs.

To expand access to LLMs, several research groups have open-sourced their pre-trained LLMs [5]. However, the sheer size of these models makes it difficult to adopt them, even if no training is needed. Prior studies have shown that the bottleneck of running LLMs is not in the computation speed, but in the GPU memory. A 100B-parameter model will require 200 GB of GPU memory to load the model parameters at the standard half precision, and many LLMs are even larger. Providing this much GPU memory is financially challenging for many users.

Recently, a model-parallel approach has been proposed to address the above challenge through resource pooling across distributed devices. This approach distributes a model to multiple devices at the granularity of transformer blocks (a.k.a. pipeline parallelism [3]) or neurons (a.k.a. tensor parallelism) to run large models on devices with small GPUs. In particular, [1] has shown that pipeline parallelism can be used to run LLM inference tasks on geographically distributed servers, each with only a few GB of GPU memory, at a much faster rate than local parameter offloading. This is achieved by letting each server host a subset of consecutive blocks, and each inference request is routed through a chain of servers that collectively host the entire model.

In this work, we consider LLM inference over geographically distributed servers using pipeline parallelism and client-centric communication, as illustrated in Fig. 1. Although the feasibility of such systems has been validated in [1], there is a lack of fundamental understanding of how to optimally manage their performance. The current solution in [1] based on heuristics for resource allocation. It remains open how to optimize the performance of such systems, while taking into account the unique characteristics of GPUs and LLM inference tasks. This work aims to fill this gap by rigorously formulating and addressing the block placement and request routing problem, using PETALS [1] as a concrete example.

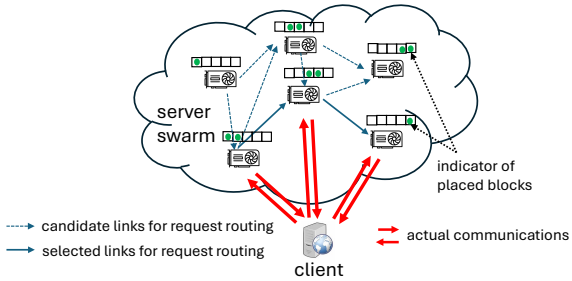


**Figure 1: System architecture for pipeline-parallel LLM inference using client-centric communication.**

## 2. MODEL

We consider the inference process of a pre-trained LLM with a decoder-only architecture [4], which is commonly adopted by state-of-the-art LLMs (e.g., GPT and its variants [2]). The input/output layers only contain a small percentage of the model parameters (e.g. $< 3\%$ for BLOOM-176B [5]) and can therefore be hosted by the client. The model blocks contain most of the parameters and are thus delegated to the servers [1].

Given a pre-trained LLM with $L$ blocks, we consider a distributed system with client-side and server-side caching, which uses this LLM to serve autoregressive sequence generation requests via a hub-spoke communication pattern as shown in Fig. 1. In this work, we focus on supporting short-prompt, long-response queries, where users ask concise questions to obtain detailed and comprehensive answers. The system contains a set of servers $V_s$ and a set of clients $V_c$, communicating through TCP connections. Each client represents an ingress point for inference requests, which can be a host owned by end users or a proxy server that submits requests on behalf of end users.

Define $|R|$ as the total number of requests. For each inference request, the client initiates an inference session. We first consider the offline case with a given set of requests to understand the structure of the problem, and then address the online case with sequentially arriving requests.

To formulate the offline case, each request has up to $l^I_{\max}$ input tokens and generates up to $l_{\max}$ output tokens. For

*This work was partly supported by the National Science Foundation under award CNS-2106294. Parimal was supported by Qualcomm Inc. under Qualcomm University Relations 6G India, Anusandhan National Research Foundation (ANRF) under Grant CRG/2023/008854, and Office of International Relations at IISc under IISc-PSU collaboration research grant. Corresponding author: tinghe@psu.edu

short-prompt long-response queries, we have $l^I_{\max} \ll l_{\max}$. As for the online case, we need to additionally model the state of each server as detailed in the methods section.

Each server $j \in V_s$ has a per-block processing time of up to $\tau^I_j(l^I_{\max})$ during the prefill phase and a per-token-per-block processing time of $\tau_j$ during the decoding phase. The connection between each client $c$ and each server $j$ has a per-input round trip time (RTT) of up to $t^I_{cj}(l^I_{\max})$ and a per-token RTT of $t_{cj}$.

If a request from client $c$ with input length $l^I_{\max}$ and output length $l_{\max}$ is processed by a chain of servers $p$ with each server $j \in p$ processing $k_j$ blocks, it will incur a total inference time of

$$\sum_{j\in p}\left(t^I_{cj}(l^I_{\max}) + k_j\tau^I_j(l^I_{\max})\right) + (l_{\max}-1)\sum_{j\in p}(t_{cj}+k_j\tau_j). \tag{1}$$

Each server $j \in V_s$ has a GPU memory of $M_j$, which denotes the effective memory capacity for storing model parameters and attention caches. The size of each block is $s_m$ bytes. The size of each attention cache is $s_c := 2d_{\text{model}} \cdot (l^I_{\max} + l_{\max}) \cdot \text{dtype\_bytes}$ bytes, where $d_{\text{model}}$ is the embedding dimension and $\text{dtype\_bytes} = 2$.

A server $j$ storing $m_j$ blocks and processing $k^r_j$ blocks for each request $r$ has a total memory consumption of

$$s_m m_j + s_c \sum_{r\in\mathcal{R}} k^r_j. \tag{2}$$

where $s_m$ is the memory size per block, and $s_c$ is the memory size per attention cache.

# 3. METHODS

We formulate a joint optimization of the placement of model blocks at servers (called block placement BP) and the selection of server chains for inference requests (called request routing RR) based on performance models experimentally validated on a real distributed LLM inference system. Our formulation features a compact representation of the decision variables that allows the problem to be formulated as a mixed integer linear programming (MILP) problem with a polynomial number of variables/constraints.

The NP-hardness of BPRR implies the need of efficient suboptimal algorithms. We develop a three-step algorithm called CG-BPRR by decomposing the joint optimization into subproblems optimizing the number of blocks per server $m$, the starting block per server $a$, and the request routing $f$, achieving polynomial complexity and a guaranteed average inference time per token under feasible block placements. It proceeds as follows:

**Step 1:** Conservatively compute the number of blocks $m_j$ for each server $j$ as $\min\left(\lfloor M_j/(s_m + s_c|R|)\rfloor, L\right)$.

**Step 2:** Sort servers by increasing amortized inference time defined as $\widetilde{t}_j := \frac{1}{m_j}\max_{c\in V_c}(t_{cj} + \tau_j m_j)$ (fastest first), then greedily assign consecutive blocks to each server in this order, prioritizing ranges that improve performance for the most under-served blocks or balance load across well-served ones.

**Step 3:** For each client $c$, build a routing graph with S-client and D-client nodes, edges between servers hosting adjacent block ranges, and costs $t^c_{ij}$, then route requests via shortest path from S-client to D-client.

We then consider the online setting with dynamically arriving requests, for which we show that the offline algorithm can be adapted into a two-time-scale solution that solves the block placement problem via robust optimization and the online request routing problem via a variation of shortest-path routing, with link costs $t^W_{ij}(t) + l_{\max}t^c_{ij}$ designed to approximate the total inference time used at each server. The online algorithm inherits the same performance guarantee as the offline algorithm as long as the number of concurrent requests is within a design limit, which can be tuned to trade off the waiting time and the per-token inference time after waiting.

# 4. RESULTS AND EVALUATION

Evaluations employ PETALS-based experiments on A100 GPUs and MIGs, alongside a validated MATLAB simulator capable of predicting GPU-based performance using CPU resources alone. This simulator replicates block placement and request routing decisions, enabling assessments of large-scale deployments beyond the constraints of available GPU hardware.

**Clustered scenarios:** Configured with three heterogeneous clusters (Cluster0 for remote clients, Cluster1 with high performance A100 servers, Cluster2 with lower performance MIGs), the proposed algorithm achieves 60–80% reductions in average per-token inference times compared to PETALS heuristics. Improvements arise from optimized memory allocation, reducing first-token times by minimizing out-of-memory errors.

**Scattered scenarios:** Tested on topologies like AboveNet (23 nodes), BellCanada (48 nodes), and GTS-CE (149 nodes), gains are confirmed, larger in resource-constrained settings. Varying parameters like number of servers $C$, request rate $\lambda$, and $l_{\max}$ in the simulator shows enhancements more pronounced under higher loads or fewer resources.

**Ablation studies:** Variations of PETALS (optimized ordering, fixed number of blocks, enhanced routing decision) highlight synergistic benefits of the integrated approach.

Algorithm running times remain negligible (under 0.05 seconds) compared to inference durations. The simulator's results align with experiments in times and logic, facilitating scalable evaluations and future research on distributed LLM inference without extensive GPU access.

# 5. REFERENCES


[1] A. Borzunov, M. Ryabinin, A. Chumachenko, et al. Distributed inference and fine-tuning of large language models over the internet. In *Advances in Neural Information Processing Systems*, volume 36, pages 12312–12331, 2023.

[2] T. B. Brown, B. Mann, N. Ryder, et al. Language models are few-shot learners. In *International Conference on Neural Information Processing Systems*, 2020.

[3] Y. Huang, Y. Cheng, A. Bapna, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[5] T. L. Scao, A. Fan, C. Akiki, et al. BLOOM: A 176B-parameter open-access multilingual language model, 2023.