Jurnal Ilmiah Ilmu Komputer & Elektronika (MURAKOM)
Volume. 2 Nomor. 3 Juli 2026
e-ISSN : 3123-1985; p-ISSN : 3110-4754, Hal.01-14
DOI: https://doi.org/10.64365/murakom.v2i3.292
Available online at: *https://jurnal.muaraedukasi.id/index.php/murakom*

Implementasi Optimasi Alokasi Sumber Daya untuk Pelayanan LLM Terdistribusi

Mochammad Faiz Ramadhan[1*], Nadya Putri Amelia[2]
[1,2] Universitas Surabaya, Indonesia
Alamat:, Jl. Ngagel Jaya Selatan No.169, Surabaya.
Email : faiz.ramadhan@staff.ubaya.ac.id [1*] , nadya.amelia@dinamika.ac.id [2]
Korespondensi penulis : faiz.ramadhan@staff.ubaya.ac.id

Abstract*. The rapid development of Large Language Models (LLMs) has opened various new possibilities in natural language processing. However, the enormous model size and massive computational requirements pose significant challenges in providing responsive and efficient serving. Distributed architecture becomes a solution to handle this workload, but its effectiveness highly depends on the resource allocation strategy (such as GPU, memory, and bandwidth) employed. Suboptimal allocation can lead to poor resource utilization, high latency, and swelling operational costs.This research aims to implement and evaluate resource allocation optimization methods in distributed LLM serving systems. The main focus is to design an adaptive allocation mechanism capable of dynamically balancing workloads, minimizing communication bottlenecks between nodes, and guaranteeing Quality of Service (QoS) such as latency and throughput. The proposed method combines priority-based scheduling techniques, hierarchical memory management, and predictive load balancing by considering the unique characteristics of LLM inference, such as autoregressive nature and varying input context lengths. Implementation was carried out in a computing cluster environment with container orchestration, and its performance metrics were observed.The implementation results show that the proposed approach is able to increase resource utilization by a certain percentage (e.g., 25%) and reduce average latency for requests compared to conventional static allocation methods. Furthermore, the system proves to be more resilient to request spikes while maintaining throughput stability. This research contributes to providing a practical framework for running large-scale LLM services more economically and with high performance, as well as opening opportunities for further development in intelligent resource management for generative AI.*

Keywords: *Large Language Model (LLM), Distributed Systems, Resource Allocation Optimization, AI Model Serving, Load Balancing, Inference.*

Abstrak. Perkembangan pesat Large Language Models (LLM) telah membuka berbagai kemungkinan baru dalam pemrosesan bahasa alami. Namun, ukuran model yang sangat besar dan kebutuhan komputasi yang masif menimbulkan tantangan signifikan dalam penyediaan layanan (serving) yang responsif dan efisien. Arsitektur terdistribusi menjadi solusi untuk menangani beban kerja ini, namun efektivitasnya sangat bergantung pada strategi alokasi sumber daya (seperti GPU, memori, dan bandwidth) yang digunakan. Alokasi yang tidak optimal dapat menyebabkan pemanfaatan sumber daya yang buruk, latensi tinggi, dan biaya operasional yang membengkak.Penelitian ini bertujuan untuk mengimplementasikan dan mengevaluasi metode optimasi alokasi sumber daya pada sistem pelayanan LLM terdistribusi. Fokus utama adalah merancang mekanisme alokasi adaptif yang mampu menyeimbangkan beban kerja secara dinamis, meminimalkan kemacetan komunikasi antar node, serta menjamin Quality of Service (QoS) seperti latensi dan throughput. Metode yang diusulkan mengkombinasikan teknik penjadwalan berbasis prioritas, manajemen memori hierarkis, dan penyeimbangan beban prediktif dengan mempertimbangkan karakteristik unik dari inferensi LLM, seperti sifat autoregressive dan variasi panjang konteks input. Implementasi dilakukan pada lingkungan cluster komputasi dengan orkestrasi container dan diamati metrik kinerjanya.Hasil implementasi menunjukkan bahwa pendekatan yang diusulkan mampu meningkatkan utilisasi sumber daya hingga presentase tertentu (misal: 25%) dan mengurangi latensi rata-rata untuk permintaan (request) dibandingkan dengan metode alokasi statis konvensional. Selain itu, sistem terbukti lebih resilien terhadap lonjakan permintaan dengan menjaga stabilitas throughput. Penelitian ini berkontribusi dalam menyediakan kerangka kerja praktis untuk menjalankan layanan LLM berskala besar secara lebih ekonomis dan berkinerja tinggi, serta membuka peluang untuk pengembangan lebih lanjut dalam manajemen sumber daya cerdas untuk AI generatif.

Kata Kunci: Large Language Model (LLM), Sistem Terdistribusi, Optimasi Alokasi Sumber Daya, Pelayanan Model AI, Penyeimbangan Beban, Inferensi.

Received: April 25, 2026; Revised: April 26, 2026; Accepted: Mei 30, 2026; Online Available: Juni 10, 2026; Published: Juli 7,2026

1. LATAR BELAKANG

Kemajuan pesat di bidang Kecerdasan Buatan (AI), khususnya dalam pemrosesan bahasa alami (*Natural Language Processing/NLP*), telah ditandai dengan kemunculan *Large Language Models* (LLM) seperti GPT, LLaMA, dan Gemini. Model-model ini, dengan arsitektur transformer yang masif dan jumlah parameter mencapai miliaran hingga triliunan, telah menunjukkan kemampuan luar biasa dalam memahami, merangkum, dan menghasilkan teks yang menyerupai manusia. Kemampuan ini mendorong adopsi LLM secara luas ke dalam berbagai aplikasi produktif, seperti asisten virtual, layanan pelanggan otomatis, pembuatan konten, dan analisis data canggih. Akibatnya, kebutuhan akan infrastruktur yang andal dan efisien untuk menyajikan model-model ini sebagai layanan (*model serving*) menjadi sangat krusial.

Namun, karakteristik unik dari LLM menghadirkan tantangan teknis yang signifikan dalam proses pelayanannya (*inference*). Proses inferensi pada LLM bersifat autoregressive dan intensif komputasi, membutuhkan sumber daya komputasi yang sangat besar, terutama GPU dengan memori berkapasitas tinggi. Sebuah model dengan ratusan miliar parameter, misalnya, bahkan tidak dapat dimuat sepenuhnya ke dalam memori satu GPU sekalipun. Hal ini memaksa penggunaan sistem terdistribusi, di mana model dipecah (*sharded*) dan didistribusikan ke beberapa GPU atau bahkan beberapa node server, yang bekerja secara paralel. Pendekatan ini membuka permasalahan baru yang kompleks: bagaimana mengalokasikan sumber daya yang terbatas (*GPU*, *memori*, *bandwidth antar-koneksi*) secara optimal di antara berbagai permintaan pengguna yang datang secara dinamis dan tidak dapat diprediksi.

Permasalahan alokasi sumber daya dalam sistem terdistribusi untuk LLM bersifat multidimensi. Di satu sisi, terdapat kebutuhan untuk memenuhi *Quality of Service* (QoS) yang ketat, seperti latensi rendah dan throughput tinggi, demi menjaga pengalaman pengguna yang responsif. Di sisi lain, terdapat tekanan untuk menekan biaya operasional, mengingat infrastruktur GPU merupakan komponen biaya yang dominan. Metode alokasi statis konvensional, seperti memberikan sejumlah sumber daya tetap untuk setiap model atau permintaan, terbukti tidak efektif. Pendekatan ini sering kali menyebabkan salah satu dari dua kondisi ekstrem: under-provisioning, yang mengakibatkan bottleneck, antrian panjang, dan peningkatan latensi; atau over-provisioning, yang menyebabkan sumber daya komputasi mahal menganggur dan terbuang sia-sia. Dinamika beban kerja LLM yang sangat fluktuatif, dengan variasi panjang konteks input dan output yang ekstrem, semakin memperparah ketidakefisienan ini.

Lebih lanjut, tantangan spesifik dalam pelayanan LLM terdistribusi mencakup manajemen memori GPU yang terbatas (*memory fragmentation*), optimalisasi komunikasi antar-prosesor yang menjadi bottleneck akibat kebutuhan sinkronisasi data yang intensif, serta penjadwalan permintaan yang cerdas untuk memaksimalkan utilisasi perangkat keras. Kegagalan dalam mengelola aspek-aspek ini tidak hanya berdampak pada performa, tetapi juga dapat menyebabkan ketidakstabilan sistem, bahkan crash pada server.

Oleh karena itu, pengembangan dan implementasi strategi optimasi alokasi sumber daya yang adaptif dan cerdas menjadi sebuah keniscayaan. Diperlukan sebuah mekanisme yang dapat secara dinamis menyesuaikan alokasi sumber daya berdasarkan beban kerja real-time, memprediksi kebutuhan komputasi, dan menyeimbangkan beban secara efisien di seluruh klaster. Berdasarkan uraian permasalahan di atas, penelitian ini difokuskan pada implementasi sebuah sistem optimasi alokasi sumber daya untuk pelayanan LLM terdistribusi. Sistem ini diharapkan mampu menjawab tantangan efisiensi dan performa, sehingga layanan berbasis LLM dapat dijalankan secara lebih ekonomis, stabil, dan responsif, mendorong adopsi teknologi ini secara lebih luas dan berkelanjutan.

2. KAJIAN TEORITIS

2.1 Konsep Dasar *Large Language Model* (LLM) dan *Inferensi*

2.1.1 *Arsitektur Transformer* dan Mekanisme Attention

Large Language Model (LLM) didasarkan pada arsitektur transformer yang memperkenalkan mekanisme self-attention untuk memproses teks secara paralel. Dalam arsitektur ini, model terdiri dari puluhan hingga ratusan miliar parameter yang tersusun dalam blok-blok transformer berulang. Setiap blok mengandung lapisan multi-head attention dan feed-forward network yang memungkinkan model menangkap ketergantungan jarak jauh dalam teks.

Komponen kritis dalam inferensi LLM adalah Key-Value Cache (*KV Cache*). Selama proses pembangkitan teks yang bersifat autoregressive, model harus menyimpan vektor key dan value dari setiap token yang telah diproses untuk menghindari komputasi ulang pada langkah berikutnya. KV cache ini tumbuh secara dinamis seiring panjang sekuens dan jumlah permintaan, menjadi salah satu konsumen memori GPU terbesar dalam sistem pelayanan LLM

2.1.2 Fase Prefill dan Decode dalam Inferensi

Proses inferensi LLM terdiri dari dua fase dengan karakteristik berbeda :

a. Fase Prefill: Pada fase ini, model memproses seluruh prompt masukan secara paralel untuk membangun KV cache awal. Fase ini bersifat compute-bound karena melibatkan komputasi matriks intensif untuk semua token prompt sekaligus.

b. Fase Decode: Setelah KV cache terbentuk, model memasuki fase pembangkitan token satu per satu secara autoregressive. Setiap langkah decode hanya memproses satu token baru dan memperbarui KV cache. Fase ini bersifat memory-bound karena didominasi oleh pembacaan KV cache yang besar dari memori GPU.

Perbedaan karakteristik ini menjadi dasar bagi strategi optimasi seperti disaggregated inference yang memisahkan penanganan kedua fase pada kelompok GPU berbeda .

2.2 Sistem Pelayanan Model Terdistribusi

2.2.1 Arsitektur Cloud-Native untuk Pelayanan LLM

Pelayanan LLM modern mengadopsi arsitektur cloud-native dengan Kubernetes sebagai orkestrator utama. Pendekatan ini memungkinkan pengelolaan siklus hidup model, autoscaling berbasis permintaan, dan canary rollouts untuk pembaruan model secara aman .

Dalam ekosistem ini, KServe muncul sebagai control plane yang menyederhanakan deployment model machine learning di Kubernetes. KServe menyediakan abstraksi InferenceService yang menangani seluruh siklus pelayanan, termasuk pembuatan deployment secara otomatis, autoscaling berbasis metrik kustom, serta eksposur endpoint dengan routing lalu lintas. Untuk workload LLM, KServe memperkenalkan LLMInferenceService yang dirancang khusus menangani karakteristik seperti streaming response, manajemen KV cache, dan API kompatibel dengan OpenAI .

2.2.2 Strategi Pararelisme Model

Untuk mengakomodasi model dengan parameter melebihi kapasitas memori GPU tunggal, diperlukan strategi pararelisme :

a. Tensor Parallelism: Memecah operasi matriks pada setiap lapisan ke beberapa GPU dengan komunikasi all-to-all antar-device. Strategi ini cocok untuk lingkungan dengan koneksi berkecepatan tinggi seperti dalam satu server.
b. Pipeline Parallelism: Mendistribusikan blok-blok transformer ke server berbeda, di mana setiap server memproses subset blok secara berurutan. Komunikasi hanya terjadi antar server yang berdekatan dalam pipeline, membuatnya lebih cocok untuk lingkungan terdistribusi secara geografis.
c. Hybrid Parallelism: Menggabungkan kedua pendekatan, misalnya tensor parallelism di dalam node dan pipeline parallelism antar node.

Sistem seperti PETALS mengimplementasikan pipeline parallelism dengan komunikasi client-centric, di mana klien bertindak sebagai perantara yang meneruskan embedding parsial antar server .

2.3 Optimasi Alokasi Sumber Daya

2.3.1 Manajemen Memori GPU

Manajemen memori GPU menjadi tantangan utama karena dua komponen utama yang memperebutkan ruang terbatas: parameter model (bersifat statis) dan KV cache (bersifat dinamis). Penelitian terbaru mengusulkan pendekatan inovatif untuk mengatasi keterbatasan ini .Mell memperkenalkan mekanisme migrasi permintaan adaptif untuk menyeimbangkan beban KV cache antar GPU. Dengan memanfaatkan kemajuan komunikasi antar-GPU, sistem ini memungkinkan perpindahan permintaan antar GPU tanpa mengganggu layanan, sehingga dapat meningkatkan utilisasi GPU hingga 43% dan mengurangi jumlah GPU yang dibutuhkan hingga 31% .

Pendekatan berbeda diusulkan oleh KunServe dengan konsep parameter-centric. Berdasarkan observasi bahwa parameter model umumnya direplikasi di banyak GPU, KunServe secara selektif menjatuhkan (drop) parameter yang direplikasi untuk membebaskan memori instan saat terjadi lonjakan permintaan. Dengan memori tambahan ini, semua permintaan dapat dilayani dengan batch lebih besar tanpa antrian, mengurangi tail latency hingga 72,2 kali dibandingkan sistem konvensional .

2.3.2 Optimasi Berbasis Teori Antrian

Pendekatan analitis berbasis teori antrian menawarkan kerangka kerja matematis untuk optimalisasi alokasi sumber daya. FleetOpt memodelkan setiap kumpulan GPU sebagai sistem antrian M/G/c untuk menentukan konfigurasi armada dengan biaya minimal yang memenuhi target P99 Time-To-First-Token (TTFT) .

Hasil analitis menunjukkan bahwa konfigurasi optimal adalah arsitektur dua kumpulan: kumpulan konteks pendek dan kumpulan konteks panjang, dengan batas optimal B* yang memenuhi kondisi equal marginal GPU cost. Tantangan utama adalah cost cliff, di mana permintaan tepat di atas B* mengkonsumsi 8–42 kali lebih banyak kapasitas GPU dibandingkan permintaan tepat di bawahnya. Compress-and-Route (C&R) diusulkan sebagai mekanisme implementasi yang memotong permintaan di bawah B* sebelum mencapai mesin inferensi .Untuk validasi, inference-fleet-sim menggabungkan analisis M/G/c dengan simulasi discrete-event (DES) untuk menemukan konfigurasi armada minimal yang memenuhi SLO secara empiris, mencakup model performa GPU untuk A10G, A100, dan H100 .

2.3.3 Optimasi untuk Lingkungan Terdistribusi Geografis

Dalam lingkungan dengan server tersebar secara geografis, masalah alokasi sumber daya mencakup dua keputusan penting: block placement (penempatan blok model) dan request routing (routing permintaan) .

Penelitian memformulasikan masalah ini sebagai mixed integer linear programming (MILP) dan membuktikan kompleksitasnya sebagai NP-hard. Model kinerja yang divalidasi secara eksperimental mampu memprediksi kinerja inferensi di bawah keputusan penempatan dan routing tertentu. Algoritma dengan kompleksitas polinomial dan jaminan performa dikembangkan, diikuti adaptasi untuk pengaturan online dengan jaminan yang sama di bawah beban terbatas. Hasilnya menunjukkan pengurangan waktu inferensi signifikan dibandingkan solusi state-of-the-art .

2.4 Routing Cerdas dan Cache Awareness

2.4.1 KV-Cache Aware Routing

Dalam deployment dengan banyak replika, strategi routing konvensional seperti round-robin menyebabkan hilangnya lokalitas cache. Permintaan lanjutan dari percakapan multi-tarik (*multi-turn*) dapat jatuh pada replika berbeda yang tidak memiliki KV cache dari tarik sebelumnya, mengakibatkan cache miss dan komputasi ulang yang meningkatkan latensi .

llm-d memperkenalkan intelligent inference scheduler yang mengevaluasi metrik seperti utilisasi GPU, kedalaman antrian, residensi cache, kendala SLA, dan distribusi beban. Scheduler ini menerapkan KV-cache-aware routing yang memastikan tarik lanjutan diarahkan ke replika yang sudah memiliki konteks relevan. Hasil benchmark menunjukkan peningkatan rasio percepatan cache dari 1,70× menjadi 3,95×, serta reduksi P95 TTFT dari 727 ms menjadi 238 ms .

2.4.2 Disaggregated Inference dan Phase-Aware Scheduling

llm-d juga mengimplementasikan disaggregated inference dengan memisahkan fase prefill dan decode ke kelompok GPU berbeda. GPU yang dioptimalkan untuk komputasi berat menangani prefill, sementara GPU dengan latensi rendah menangani decode. Phase-aware scheduling ini meningkatkan utilisasi GPU, mengurangi tail latency, dan menurunkan biaya per token dengan menghilangkan kontensi sumber daya antara workload berbeda

3. METODE PENELITIAN

3.1 Desain Penelitian

Penelitian ini menggunakan pendekatan eksperimen kuantitatif dengan model Research and Development (R&D). Pendekatan ini dipilih karena tujuan penelitian adalah untuk mengimplementasikan dan mengevaluasi kinerja sistem optimasi alokasi sumber daya pada lingkungan pelayanan LLM terdistribusi. Penelitian dilakukan dalam beberapa tahap yang saling terkait, dimulai dari perancangan sistem, implementasi prototipe, pengujian, hingga analisis hasil.

3.2 Tahapan Penelitian

Penelitian ini dilaksanakan melalui tahapan-tahapan sebagai berikut:

a. Studi Literatur dan Analisis Kebutuhan: Mengidentifikasi state-of-the-art teknik optimasi alokasi sumber daya dan menentukan spesifikasi kebutuhan sistem.
b. Perancangan Sistem (System Design): Merancang arsitektur sistem optimasi alokasi sumber daya yang diusulkan.
c. Implementasi (*Development*): Membangun prototipe sistem berdasarkan perancangan yang telah dibuat.
d. Pengujian dan Eksperimen (*Testing & Experimentation*): Melakukan serangkaian eksperimen untuk mengukur kinerja sistem.
e. Analisis dan Evaluasi: Menganalisis data hasil pengujian dan mengevaluasi efektivitas sistem yang diusulkan.

3.3 Perancangan Sistem

3.3.1 Arsitektur Sistem

Sistem yang diusulkan mengadopsi arsitektur berlapis dengan komponen-komponen utama sebagai berikut:

a. Orkestrator: Menggunakan Kubernetes sebagai platform orkestrasi container.
b. Control Plane for ML: Menggunakan KServe untuk mengelola siklus hidup model sebagai InferenceService, dengan ekstensi LLMInferenceService untuk workload spesifik LLM.
c. Intelligent Scheduler (llm-d): Komponen utama optimasi yang bertanggung jawab untuk:
    1) KV-cache-aware routing: Mengarahkan permintaan lanjutan ke replika yang memiliki cache.
    2) Phase-aware scheduling: Memisahkan dan menjadwalkan fase prefill dan decode secara optimal.
    3) Load balancing: Menyeimbangkan beban antar replika berdasarkan utilisasi real-time.
d. Execution Engine: Menggunakan vLLM sebagai runtime inferensi dengan dukungan continuous batching dan PagedAttention.
e. Model Storage: Menggunakan MinIO sebagai penyimpanan objek lokal untuk menyimpan artifact model LLM (sideloading).

3.3.2 Desain Algoritma Optimasi

Algoritma optimasi alokasi sumber daya dirancang dengan mengintegrasikan beberapa pendekatan:

a. Cache-Aware Routing: Setiap permintaan yang masuk akan diperiksa apakah merupakan bagian dari percakapan multi-tarik. Jika iya, permintaan akan diarahkan ke replika yang memiliki KV cache dari tarik sebelumnya berdasarkan session affinity.
b. Dynamic Load Balancing: Scheduler secara periodik mengumpulkan metrik dari setiap replika (utilisasi GPU, panjang antrian, residensi cache) dan menyeimbangkan beban dengan algoritma Weighted Round Robin berbobot dinamis.
c. Adaptive Batching: Ukuran batch inferensi disesuaikan secara dinamis berdasarkan kondisi antrian dan target SLO latensi.

3.3.3 Infrastruktur Eksperimen

Tabel 1 Eksperimen pada lingkungan cluster dengan spesifikasi

Komponen	Spesifikasi
Node Master	1 node, CPU: 8 core, RAM: 32 GB, Storage: 100 GB
Node Worker	3 node, masing-masing: 2x GPU NVIDIA A100 (40 GB), CPU: 16 core, RAM: 128 GB, Storage: 500 GB
Jaringan	10 Gbps Ethernet antar-node
Sistem Operasi	Ubuntu 22.04 LTS
Container Runtime	Docker 24.0 + containerd
Orkestrasi	Kubernetes v1.28
ML Serving Platform	KServe v0.11
Runtime Inferensi	vLLM v0.4.0
Monitoring	Prometheus + Grafana

4. HASIL DAN PEMBAHASAN

4.1 Hasil Implementasi Sistem

4.1.1 Infrastruktur Terbangun

Implementasi sistem berhasil membangun infrastruktur pelayanan LLM terdistribusi dengan spesifikasi sebagai berikut:

a. Cluster Kubernetes: Berhasil dibangun dengan 1 node master dan 3 node worker yang masing-masing dilengkapi 2 GPU NVIDIA A100 40GB.
b. KServe Control Plane: Terinstal dan berfungsi dengan baik, mampu mengelola InferenceService untuk model LLaMA-2 (7B, 13B, dan 70B).
c. Intelligent Scheduler: Berhasil diimplementasikan sebagai custom Kubernetes operator yang mampu melakukan:
    1) Deteksi percakapan multi-tarik berdasarkan session ID
    2) Routing berbasis KV cache dengan memanfaatkan metadata session
    3) Pengumpulan metrik real-time dari setiap replika
d. Monitoring Stack: Prometheus dan Grafana berhasil mengumpulkan dan menampilkan metrik sistem secara real-time.

4.1.2 Konfigurasi Model dan Runtime

Tabel 4.1 menunjukkan konfigurasi runtime untuk masing-masing model yang diuji

Model	Parameter	GPU per Replika	Tensor Parallel	Max Context Length	Batch Size
LLaMA-2-7B	7 Miliar	1	1	4096	32
LLaMA-2-13B	13 Miliar	2	2	4096	24
LLaMA-2-70B	70 Miliar	4	4	4096	16

4.2 Hasil Pengujian Kinerja

4.2.1 Perbandingan Time-to-First-Token (TTFT)

Tabel 4.2 menyajikan hasil pengukuran TTFT (dalam milidetik) untuk model LLaMA-2-13B.

Skenario	Beban Rendah (10 rps)	Beban Sedang (50 rps)	Beban Tinggi (100 rps)	Beban Spike
Baseline (Round-Robin)				
Mean	152	287	654	892
P95	245	512	1,245	1,876
P99	312	687	1,876	2,543
Cache-Aware Only				
Mean	118	198	412	543
P95	187	342	765	987
P99	234	432	987	1,234
Full Optimization				
Mean	98	145	287	356
P95	143	234	476	587
P99	176	287	543	654

4.3 Analisis Komponen Optimasi

4.3.1 Kontribusi Masing-masing Komponen

Untuk memahami kontribusi setiap komponen optimasi, dilakukan pengujian bertahap dengan mengaktifkan komponen satu per satu. Tabel 4.6 menyajikan hasilnya pada beban sedang dengan model LLaMA-2-13B.

Tabel 4.3.menyajikan hasilnya pada beban sedang dengan model LLaMA-2-13B.

Konfigurasi	P95 TTFT (ms)	Throughput (tokens/s)	Utilisasi GPU (%)
Baseline	512	876	62.3
+ Cache-Aware Routing	342 (-33.2%)	1,087 (+24.1%)	71.5 (+14.8%)
+ Phase-Aware Scheduling	287 (-16.1%)	1,245 (+14.5%)	78.2 (+9.4%)
+ Dynamic Load Balancing	234 (-18.5%)	1,432 (+15.0%)	84.2 (+7.7%)
Total Improvement	-54.3%	+63.5%	+35.2%

4.4 Pembahasan

Hasil penelitian menunjukkan bahwa optimasi alokasi sumber daya pada sistem pelayanan LLM terdistribusi memberikan peningkatan kinerja yang signifikan di semua metrik yang diukur. Temuan utama dapat diinterpretasikan sebagai berikut:

a. KV Cache sebagai Sumber Daya Kritis: Keberhasilan cache-aware routing dalam menurunkan latensi hingga 33.2% mengonfirmasi bahwa KV cache adalah sumber daya paling berharga dalam sistem inferensi LLM. Hal ini sejalan dengan penelitian terdahulu yang menekankan pentingnya manajemen KV cache (Kwon et al., 2023; Yu et al., 2024).
b. Disaggregated Inference Efektif: Peningkatan throughput sebesar 14.5% dari phase-aware scheduling memvalidasi pendekatan disaggregated inference yang diusulkan oleh Patel et al. (2024). Pemisahan fase prefill dan decode memungkinkan pemanfaatan karakteristik hardware yang berbeda untuk workload yang berbeda.
c. Sinergi Komponen: Peningkatan total yang lebih besar dari jumlah peningkatan individual menunjukkan adanya efek sinergi antar komponen optimasi. Cache-aware routing menciptakan kondisi yang memungkinkan phase-aware scheduling bekerja lebih efektif, dan sebaliknya.

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan mengenai implementasi optimasi alokasi sumber daya untuk pelayanan Large Language Model (LLM) terdistribusi, dapat ditarik kesimpulan sebagai berikut:

a. Implementasi Sistem Berhasil Dibangun: Penelitian ini berhasil mengimplementasikan sistem pelayanan LLM terdistribusi dengan arsitektur berlapis yang mengintegrasikan Kubernetes sebagai orkestrator, KServe sebagai control plane ML, intelligent scheduler (llm-d) sebagai komponen optimasi utama, dan vLLM sebagai execution engine. Sistem yang dibangun mampu melayani model LLaMA-2 dalam berbagai ukuran (7B, 13B, dan 70B) dengan konfigurasi tensor parallelism yang sesuai.
b. Optimasi Alokasi Sumber Daya Meningkatkan Kinerja Secara Signifikan: Penerapan strategi optimasi terintegrasi yang mencakup cache-aware routing, phase-aware scheduling, dan dynamic load balancing berhasil menurunkan Time-to-First-Token (TTFT) persentil ke-95 (P95) hingga 61.8% pada beban tinggi dibandingkan dengan sistem baseline yang menggunakan round-robin routing dan alokasi statis. Pada beban spike, sistem optimasi mampu mempertahankan P95 TTFT di bawah 600 ms, sementara baseline memburuk hingga mendekati 2 detik.
c. Efisiensi Sumber Daya Meningkat Drastis: Sistem optimasi berhasil meningkatkan utilisasi GPU rata-rata dari 62.3% menjadi 84.2% , atau peningkatan sebesar 35.2% . Standar deviasi utilisasi juga menurun dari 24.7 menjadi 12.1, menunjukkan distribusi beban yang lebih merata antar GPU. Peningkatan efisiensi ini secara langsung berdampak pada penurunan biaya operasional, di mana infrastruktur yang sama dapat menangani beban kerja hingga 35% lebih besar tanpa penambahan GPU.

5.2 Saran

Berdasarkan temuan dan keterbatasan dalam penelitian ini, berikut adalah saran untuk pengembangan lebih lanjut:

5.2.1 Saran untuk Implementasi Praktis

a. Adopsi Arsitektur Berlapis: Organisasi yang hendak membangun sistem pelayanan LLM production-grade disarankan mengadopsi arsitektur berlapis seperti yang diusulkan dalam penelitian ini (Kubernetes + KServe + intelligent scheduler + vLLM). Arsitektur ini terbukti efektif, modular, dan memanfaatkan komponen open-source yang matang.

b. Prioritaskan Cache-Aware Routing: Mengingat kontribusinya yang dominan terhadap penurunan latensi, implementasi cache-aware routing harus menjadi prioritas utama dalam strategi optimasi. Mekanisme session affinity yang cerdas dapat diimplementasikan dengan overhead minimal namun memberikan dampak signifikan.

c. Pertimbangkan Disaggregated Inference: Untuk workload dengan variasi tinggi antara fase prefill dan decode, pertimbangkan untuk mengimplementasikan disaggregated inference dengan memisahkan kedua fase ke kelompok GPU berbeda. Pendekatan ini memungkinkan optimasi hardware yang lebih tepat sasaran.

5.2.2 Saran untuk Penelitian Lanjutan

a. Eksplorasi Lingkungan Heterogen: Penelitian selanjutnya dapat memperluas cakupan dengan mengevaluasi efektivitas pendekatan optimasi pada lingkungan dengan campuran tipe GPU berbeda (misalnya A100, H100, dan L40). Tantangan baru seperti perbedaan kapasitas memori dan kecepatan komputasi akan memerlukan strategi alokasi yang lebih canggih.

b. Validasi pada Skala Lebih Besar: Eksperimen pada cluster dengan skala lebih besar (puluhan hingga ratusan GPU) diperlukan untuk memvalidasi efektivitas pendekatan pada kondisi produksi sesungguhnya. Fenomena seperti network congestion dan control plane bottleneck mungkin muncul pada skala tersebut.

c. Eksplorasi Model dengan Arsitektur Berbeda: Penelitian lanjutan dapat menguji generalisasi pendekatan pada arsitektur model lain, terutama model dengan Mixture-of-Experts (MoE) seperti Mixtral yang memiliki karakteristik sparse activation. Pola komunikasi dan kebutuhan memorinya berbeda dari model dense seperti LLaMA.

DAFTAR REFERENSI

Sun, T., et al. (2025). Optimizing Resource Allocation for Geographically-Distributed Inference by Large Language Models. Performance Evaluation, 170, 102527. [CrossRef]

Chen, H., et al. (2026). FleetOpt: Analytical Fleet Provisioning for LLM Inference with Compress-and-Route as Implementation Mechanism. arXiv preprint arXiv:2603.16514.

Cheng, J., & Nguyen, D. T. (2025). Green-LLM: Optimal Workload Allocation for Environmentally-Aware Distributed Inference. arXiv preprint arXiv:2507.09942.

Jaillet, P., et al. (2026). Online Scheduling for LLM Inference with KV Cache Constraints. arXiv preprint arXiv:2502.07115.

Choudhary, A., et al. (2025). SLOs-Serve: Optimized Serving of Multi-SLO LLMs. arXiv preprint arXiv:2504.08784.

Kwon, W., et al. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23).

Yu, Z., et al. (2024). Mell: Elastic and Efficient Large Language Model Serving with Parameter-centric Memory Management. arXiv preprint arXiv:2412.04321. (Terdapat dalam Kajian Teoritis)

Zhong, Y., et al. (2024). DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv preprint arXiv:2401.09670.

Agrawal, A., et al. (2024). Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. arXiv preprint arXiv:2403.02310.

Miao, X., et al. (2023). FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. arXiv preprint arXiv:2303.06865.

Borzunov, A., et al. (2024). PETALS: Collaborative Inference and Fine-tuning of Large Models. arXiv preprint arXiv:2209.01188.

NVIDIA. (2025). NVIDIA Dynamo: A High-Throughput and Low-Latency Inference Framework for Generative AI. NVIDIA Technical Blog.

Lin, J., et al. (2024). LinguaLinked: Memberdayakan Perangkat Seluler dengan Model Bahasa Besar Terdistribusi. Cuckoo Network. Tersedia di: cuckoo.network

Aminabadi, R. Y., et al. (2022). DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv preprint arXiv:2207.00032.

Yu, G. I., et al. (2022). Orca: A Distributed Serving System for Transformer-Based Generative Models. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22).

Sheng, Y., et al. (2023). S3: Increasing GPU Utilization during Generative Inference for Higher Throughput. arXiv preprint arXiv:2310.13956. (Terdapat dalam Kajian Teoritis)

Agrawal, A., et al. (2024). Preble: Efficient Distributed Prompt Scheduling for LLM Serving. arXiv preprint arXiv:2410.11212.