EFFICIENT RESOURCE ORCHESTRATION FOR DISTRIBUTED LARGE LANGUAGE MODEL INFERENCE AT THE EDGE

by
SAMA HABIBI

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Doctor of Philosophy

Sabancı University
June 2025

EFFICIENT RESOURCE ORCHESTRATION FOR DISTRIBUTED LARGE LANGUAGE MODEL INFERENCE AT THE EDGE

Approved by:

Prof. Dr. ÖZGÜR ERÇETİN .....
(Dissertation Supervisor)

Asst. Prof. ECE GELAL SOYAK .....

Asst. Prof. MOHANED CHRAITI .....

Prof. Dr. ÖZGÜR GÜRBÜZ .....

Prof. Dr. DİDEM GÖZÜPEK .....

Date of Approval: July 18, 2025

ABSTRACT

EFFICIENT RESOURCE ORCHESTRATION FOR DISTRIBUTED LARGE LANGUAGE MODEL INFERENCE AT THE EDGE

SAMA HABIBI

ELECTRONICS ENGINEERING Ph.D DISSERTATION, June 2025

Dissertation Supervisor: Prof. Dr. ÖZGÜR ERÇETİN

Keywords:Adaptive Scheduling, Distributed AI, Edge Computing, Fair Incentive Mechanism, Large Language Models, Resource Allocation.

The increasing deployment of Large Language Models (LLMs) in real-time and resource-constrained environments has exposed critical limitations of centralized cloud inference, including high latency, cost, and scalability concerns. This thesis addresses these challenges by proposing two integrated solutions for efficient and fair distributed LLM inference at the edge: the *Fair Cost-Efficient Incentive Mechanism* (FCIM) and the *Adaptive Dynamic Scheduling Algorithm* (ADSA). FCIM introduces an auction-based layer allocation framework that ensures truthful participation and fairness among heterogeneous devices, dynamically balancing task latency, reward cost, memory feasibility, and system-wide alignment. ADSA complements FCIM by scheduling layer execution in a deadline-aware and preemption-conscious manner, reducing queuing delay while adapting to fluctuating device availability and network conditions.

Together, FCIM and ADSA offer a scalable, incentive-compatible, and resource-efficient approach for edge-based inference. The mechanisms are extensively evaluated through simulations across diverse model architectures, including GPT-Neo, GPT-3, and BLOOM, under varying GPU configurations and bidding scenarios. Results demonstrate that FCIM reduces communication overhead by up to 54.7% and task processing time by 36.9%, while ADSA decreases queuing delays by 39% compared to conventional schedulers. Fairness is quantitatively validated using Jain's index over both reward and layer distributions, with FCIM consistently outperforming baseline methods. This thesis establishes a principled foundation for deploying

LLMs in federated, latency-sensitive environments and offers insights into future extensions involving reinforcement learning, multi-tenant inference, and real-world edge deployments.

ÖZET

UÇ BILIŞIMDE DAĞITIK BÜYÜK DIL MODELI ÇIKARIMI IÇIN ETKIN KAYNAK ORKESTRASYONU

SAMA HABIBI

ELEKTRONİK MÜHENDİSLİĞİ DOKTORA TEZİ, Haziran 2025

Tez Danışmanı: Prof. Dr. ÖZGÜR ERÇETİN

Anahtar Kelimeler: Uyarlanabilir Zamanlama, Dağıtık Yapay Zekâ, Uç Bilişim, Adil Teşvik Mekanizması, Büyük Dil Modelleri, Kaynak Tahsisi

Gerçek zamanlı ve kaynak kısıtlı ortamlarda Büyük Dil Modelleri'nin (LLM'ler) giderek daha fazla yaygınlaştırılması, merkezi bulut çıkarımının yüksek gecikme süresi, maliyet ve ölçeklenebilirlik gibi temel sınırlamalarını gözler önüne sermiştir. Bu tez, uç bilişim ortamında etkin ve adil bir dağıtık LLM çıkarımı gerçekleştirmek amacıyla iki entegre çözüm önermektedir: *Adil ve Maliyet-Etkin Teşvik Mekanizması (FCIM)* ve *Uyarlamalı Dinamik Zamanlama Algoritması (ADSA)*.

FCIM, doğruluğu teşvik eden ve heterojen cihazlar arasında adil katılımı garanti altına alan, açık artırma tabanlı bir katman atama çerçevesi sunar. Bu yapı, görev gecikmesini, ödül maliyetini, bellek uygunluğunu ve sistem çapında hizalanmayı dinamik şekilde dengelemektedir. FCIM'i tamamlayıcı nitelikteki ADSA ise, katmanların yürütülmesini son teslim tarihlerine duyarlı ve kesintiye açık şekilde planlayarak, cihaz erişilebilirliği ve ağ koşullarındaki değişkenliklere uyum sağlarken kuyruk gecikmelerini azaltmayı hedeflemektedir.

FCIM ve ADSA birlikte, uç ortamda çıkarım için ölçeklenebilir, teşvik uyumlu ve kaynak açısından verimli bir yaklaşım sunar. Bu mekanizmalar, GPT-Neo, GPT-3 ve BLOOM gibi farklı model mimarileri altında çeşitli GPU yapılandırmaları ve teklif senaryolarıyla gerçekleştirilen simülasyonlar aracılığıyla kapsamlı şekilde değerlendirilmiştir. Sonuçlar, FCIM'in iletişim yükünü %54,7'ye kadar, görev işleme süresini ise %36,9 oranında azalttığını; ADSA'nın ise klasik zamanlayıcılara kıyasla kuyruk gecikmelerini %39 oranında düşürdüğünü göstermektedir. Adalet,

hem ödül hem de katman dağılımı üzerinde Jain endeksi ile nicel olarak doğrulanmış ve FCIM'in sürekli olarak temel yöntemlerden daha iyi performans gösterdiği ortaya konmuştur.

ACKNOWLEDGEMENTS

I sincerely thank my advisor, **Prof. Dr. Özgür Erçetin**, for his guidance and support throughout this research. His expertise and feedback were invaluable for the completion of this thesis.

My deepest gratitude goes to my family—my mother, father, and brother—for their constant encouragement and patience. Their unwavering belief in me kept me motivated, even from afar.

I also extend my appreciation to my friends and the Graduate Office of Sabancı University for their support during this journey.

To my mother, father, and brother—
whose love crossed borders to keep me standing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Generative AI models—particularly Large Language Models (LLMs)—have become powerful tools for tasks such as language understanding, translation, decision support, and content generation. Since the introduction of the transformer architecture Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser & Polosukhin (2017), the size and capabilities of LLMs have increased rapidly. Models like GPT-4o, Llama3-400B, and Gemini Pro now exceed 400billion parameters, demonstrate near-human reasoning across a wide range of tasks, and support thousands of real-world applications. However, this impressive performance comes at a high computational cost: each inference can involve trillions of operations and require several terabytes of memory, making real-time deployment both technically and economically challenging Jaiswal, Jain, Simmhan, Parayil, Mallick, Wang, Amant, Bansal, Rühle, Kulkarni & others (2025).

1.1 Centralised Inference: Strengths and Limitations

Centralised cloud platforms—such as Amazon SageMaker, Azure AI, and Baidu PaddleInference—have become the standard choice for deploying Large Language Models (LLMs). These platforms provide virtually unlimited computing power, streamlined deployment tools, and mature infrastructure, making them ideal for serving large-scale models like GPT-3 or LLaMA.

Despite these strengths, centralised cloud inference faces critical limitations when applied to latency-sensitive, cost-constrained, or privacy-regulated applications. Three primary challenges are discussed below:

- **Network-induced latency.** Inference requests must traverse the public Internet, introducing unavoidable round-trip delays. Even under ideal condi-

tions, these delays typically exceed 100 milliseconds—close to the upper limit of acceptable latency for real-time systems such as conversational agents and autonomous robots. Additionally, network variability (e.g., jitter and congestion) can further degrade responsiveness, disrupting systems that rely on precise timing Xu, Nagothu & Chen (2024).

- **High operational and energy costs.** Maintaining a large LLM (e.g., 70B parameters) in a fully available state may require up to 40 A100 GPUs, drawing roughly 30 kilowatts of power and incurring operational costs around $350 per hour. This level of resource usage is often prohibitive for small enterprises or academic labs and contributes significantly to carbon emissions Patterson, Gonzalez, Le, Liang, Munguia, Rothchild, So, Texier & Dean (2021).
- **Data privacy and sovereignty risks.** Many applications involve sensitive or regulated data. Laws such as the EU's GDPR and Canada's CPPA impose strict controls on cross-border data movement. Relying solely on centralised servers—often located in other countries—can raise legal and ethical concerns, particularly in sectors like healthcare, finance, and government.

1.2 From Cloud to Edge: The Case for Decentralised Inference

Edge computing has emerged as a promising solution to the limitations of cloud-based inference by relocating computation closer to where data is generated—at the "edge" of the network. This architectural shift reduces the physical and network distance between the model and the user, which can significantly decrease latency, alleviate backbone network congestion, and help ensure compliance with local data sovereignty requirements Shi, Cao, Zhang, Li & Xu (2016).

Several early efforts have explored this idea in the context of deep learning. For instance, systems like DeepMon Huynh, Lee & Balan (2017), and MoDNN Mao, Chen, Nixon, Krieger & Chen (2017) demonstrated how convolutional neural networks (CNNs) can be split across mobile devices and edge servers. These approaches showed that offloading parts of the model to nearby edge resources could achieve better latency and energy efficiency than relying solely on the cloud.

More recent work has attempted to apply similar ideas to transformer-based architectures. Deng et al. Deng, Zhao, Fang, Yin, Dustdar & Zomaya (2020) studied the

feasibility of distributing parts of a transformer across mobile GPUs, while PipeEdge Yuan, He, Chen, Dou, Jin & Yang (2023) and HexGen Jiang, Yan, Yao, Zhou, Chen & Yuan (2023) introduced mechanisms for statically assigning transformer layers to edge nodes. These methods demonstrated that partitioning large language models like BLOOM-7B or GPT-Neo is technically viable in edge environments.

However, several unresolved challenges limit the effectiveness of these approaches in practice:

- **Assumption of stable device sets.** Many prior systems assume a fixed, reliable set of edge devices—typically tested in controlled lab environments or on stationary infrastructure like base stations. In reality, edge devices can be mobile, intermittently connected, or affected by fluctuating network conditions. These dynamics introduce unpredictability in availability, bandwidth, and reliability, which must be accounted for in any robust deployment Bakhtiarnia, Milošević, Zhang, Bajović & Iosifidis (2023).

- **Lack of incentive alignment.** Existing scheduling algorithms often treat edge nodes as cooperative and compliant, assuming that they will truthfully report their costs and capabilities. This is unrealistic in open environments where device owners may act in self-interest. For example, they might underreport costs to win assignments or exaggerate their capabilities. Without mechanisms to ensure honest participation, such misreporting can degrade system performance Liu, Ge, Li, Chen, Gong & Cui (2021); Xiao, Gao, Yang, Cao, Wang & Feng (2023a).

- **Overlooking layer heterogeneity.** Transformer models are not composed of uniformly demanding layers. Early layers—particularly those involved in attention mechanisms—tend to be more memory-bound, whereas deeper layers involving feedforward operations are compute-intensive Xu, Yang, Yuan & Li (2021). Assigning layers in a static or uniform fashion fails to take these differences into account, leading to inefficient resource use and potential deadline violations in latency-critical applications.

In summary, while decentralised inference offers a promising pathway toward real-time, privacy-preserving, and scalable LLM deployment, current methods fall short when faced with the realities of device heterogeneity, unreliable connectivity, and the need for economic fairness. Addressing these gaps is essential to realizing the full potential of LLMs in edge environments.

1.3 Architectural Techniques for Distributed LLM Inference

Deploying large language models in distributed environments—whether across cloud clusters or edge devices—requires a range of supporting strategies to manage memory, computation, and communication overhead. This section highlights four major areas of research that have shaped the evolution of distributed LLM inference, along with their limitations in edge contexts.

Pipeline and Tensor Parallelism: One of the earliest breakthroughs in distributed training and inference was pipeline parallelism, introduced by GPipe Huang, Cheng, Bapna, Firat, Chen, Chen, Lee, Ngiam, Le, Wu & others (2019). It works by dividing a neural network into sequential stages and processing different data "micro-batches" through each stage in a pipelined fashion. While GPipe laid the groundwork, later systems like PipeDream Narayanan, Harlap, Phanishayee, Seshadri, Devanur, Ganger, Gibbons & Zaharia (2019) improved efficiency by balancing the "pipeline bubble" overhead through hybrid parallelism—combining pipeline and data parallelism.

Other advances, such as tensor parallelism, further accelerated inference by splitting the computation within each layer across multiple GPUs. Examples include Megatron-LM Shoeybi, Patwary, Puri, LeGresley, Casper & Catanzaro (2019), which uses fine-grained model sharding, and Switch Transformers Fedus, Zoph & Shazeer (2022), which route tokens through a sparse subset of model experts to reduce overhead.

Despite their success in scaling model width, these techniques generally assume high-performance, homogeneous GPU clusters with fast interconnects (e.g., NVLink). As a result, they do not translate well to edge environments, where device capabilities vary widely and communication links are unreliable or bandwidth-limited.

Compression and Adaptation: To reduce the resource demands of LLMs, researchers have also explored model compression techniques. Knowledge distillation, as used in DistilBERT Sanh, Debut, Chaumond & Wolf (2019), trains a smaller model to mimic a larger one, capturing its behavior with fewer parameters. Another effective approach is parameter-efficient fine-tuning, such as LoRA Hu, Shen, Wallis, Allen-Zhu, Li, Wang, Wang, Chen & others (2022), which updates only a small subset of model weights, significantly reducing training overhead.

These methods primarily address vertical scaling—i.e., making individual models

lighter. While useful, they are largely orthogonal to the horizontal distribution strategies that this thesis focuses on. In fact, compression techniques can be complementary to our proposed layer-allocation framework and may improve efficiency when used together.

Edge-Aware Runtime Systems: Runtime systems tailored for edge computing attempt to optimize where and how computation is performed. Some works use profiling and adaptive offloading for CNNs, deciding whether computation should happen locally or be pushed to a nearby server based on resource availability. More recently, TLPipe Zuo (2020) examined transformer partitioning by profiling attention layers and running distributed inference across clusters of Raspberry Pi devices.

While these systems represent important progress, they all rely on trusted and preconfigured infrastructures. Critically, they assume devices will honestly report their capabilities and comply with task assignments. In open or semi-trusted environments, such as community edge clouds or federated systems, these assumptions may not hold. Importantly, none of these systems incorporate economic incentives to regulate participation or ensure reliability.

Incentive Mechanisms in Edge Computing: Incentive design has long been a research topic in mobile and edge computing, especially for task offloading in heterogeneous networks. Techniques include Truthful auction-based resource allocation mechanisms Wang, Wu, Wang, Zeng, Ma & Yu (2023), Stackelberg games Xu, Ge & Wu (2020), and blockchain-backed reward systems Wang, Chen & Wu (2023). These models aim to ensure fair and rational participation among devices with varying preferences and constraints.

However, most existing mechanisms are built on the assumption that tasks are divisible or independent. This assumption does not apply to transformer inference, where layers must be executed sequentially and the failure of a single layer can stall the entire pipeline. In this setting, a misbehaving or underperforming node can compromise the end-to-end task, making reliability and strategic behavior even more critical Wang, Zhang, Gholami & others (2022).

1.4 Problem Statement and Research Questions

Despite significant advances in deploying large language models (LLMs) at the edge, the literature reveals a crucial gap: no existing framework simultaneously addresses the challenges of layer-aware heterogeneity, strategic incentives, and real-time deadline constraints. This thesis aims to fill this gap by investigating the following key research questions:

RQ1: Dynamic Incentive-Aligned Allocation How can we design a real-time market mechanism that allocates sequential LLM layers to heterogeneous, self-interested edge devices, while guaranteeing cost efficiency, incentive compatibility (truthfulness), and adherence to memory constraints?

RQ2: Deadline-Aware Scheduling What strategy can be employed to ensure that task scheduling satisfies deadlines while minimizing overall task latency under communication and computational constraints?

RQ3: Empirical Performance Evaluation How do the proposed FCIM and ADSA mechanisms perform in terms of fairness, cost efficiency, queue reduction, and scalability, when compared to current state-of-the-art solutions?

1.5 Proposed Mechanisms and Contributions

To address the research questions outlined above, this thesis proposes two closely integrated mechanisms:

- **Fair Cost-Efficient Incentive Mechanism (FCIM).** A practical, context-aware layer allocation approach that promotes fair and cost-conscious participation among heterogeneous edge devices. FCIM adapts a multi-scenario reward strategy to encourage cooperative behavior under realistic system constraints such as memory limits and latency sensitivity.
- **Adaptive Dynamic Scheduling Algorithm (ADSA).** A scheduling strategy that leverages classical real-time scheduling principles and adapts them for pipeline-parallel inference. ADSA prioritizes deadlines while minimizing preemption, accounting for intra-device computation and inter-device communication latencies.

To validate our techniques, we conduct a detailed experimental assessment using simulations of real-world workloads analyzing: (i) *Inference latency reduction* —mea-

sured as end-to-end task completion time; (ii) *Total inference cost* —the payment made to participating devices; (iii) *Total communication overhead* —intra-device communication within the inference pipeline; (iv) *Memory utilization* —evaluating effective use of device GPU memory; (v) *Cost efficiency analysis* —defined as the number of allocated layers per unit cost; (vi) *Fairness in distribution of layers and rewards* —equitable distribution of layers and rewards across devices; (vii) *Scalability* —performance across LLMs of varying sizes and diverse device capabilities.

1.6 Thesis Roadmap

The remainder of this thesis is organized as follows. Chapter 2 provides the background and literature review necessary for understanding the foundations of distributed large language model (LLM) inference in edge computing environments. It discusses existing LLM deployment frameworks, conventional scheduling methods, and presents a complexity analysis for the proposed mechanisms. Chapter 3 introduces the overall system model, including the network topology, task model, communication and computation delay formulations, pipeline parallelism, and system constraints related to memory and device heterogeneity. Chapter 4 formally defines the joint optimization problem of layer allocation and task scheduling, including detailed model constraints and complexity analysis. Chapter 5 presents the proposed solution methodologies. It first introduces the Fair Cost-Efficient Incentive Mechanism (FCIM) for the assignment subproblem, providing theoretical proofs of its utility, truthfulness, and equilibrium conditions. It then describes the Adaptive Deadline-aware Scheduling Algorithm (ADSA) to address the scheduling subproblem under dynamic constraints. Chapter6 presents an extensive experimental evaluation of both FCIM and ADSA. It analyzes their performance across a range of models and deployment scenarios in terms of cost efficiency, fairness, communication overhead, processing time, and energy consumption. Finally, Chapter 7 concludes the thesis by summarizing key contributions and suggesting directions for future work.

2. BACKGROUND AND LITERATURE REVIEW

The growing computational and latency constraints of centralized inference systems have led to increased interest in deploying LLMs in distributed and edge environments. This section reviews existing frameworks for LLM inference, incentive mechanisms, and task scheduling, highlighting their key limitations. These limitations motivate the development of the proposed FCIM and ADSA methods for task assignment and scheduling in heterogeneous edge settings.

2.1 Distributed LLM Inference and Edge Computing

This subsection examines distributed AI and edge computing as scalable alternatives to centralized LLM inference, which often suffers from latency and resource inefficiencies. Recent frameworks aim to improve task allocation, scalability, and resource utilization in decentralized environments.

AR-Edge Xu et al. (2024) is a recent framework designed for resilient edge-based LLM inference, integrating trust-based task decentralization and fault-tolerant orchestration to improve scalability in smart city deployments. While it introduces important mechanisms for reliability and control-plane resilience, it largely overlooks the computational challenges of transformer-based inference. In particular, AR-Edge does not address how to coordinate fine-grained layer execution across multiple devices or ensure fairness and truthful participation among self-interested nodes—capabilities that are critical in federated edge environments with heterogeneous hardware and limited trust.

HexGen Jiang et al. (2023), in contrast, focuses on performance optimization through tensor and pipeline partitioning strategies tailored for heterogeneous GPU clusters. Although it effectively models compute and communication costs, HexGen

assumes a stable, datacenter-like environment with no consideration for incentive compatibility, fair resource allocation, or decentralized device autonomy. These assumptions limit its applicability in more dynamic edge contexts, where participation is voluntary and resources fluctuate.

These limitations motivate the need for a principled mechanism—such as our proposed FCIM—that is built from the ground up to handle fairness, cost-efficiency, and coordination under real-world edge constraints. FCIM introduces a decentralized, auction-based approach that supports fine-grained, layer-level scheduling. Through a bidding and reward framework, it aligns incentives among heterogeneous devices, promotes truthful participation, and ensures equitable workload distribution. By directly addressing the gaps in fairness, autonomy, and coordination left by systems like AR-Edge and HexGen, FCIM offers a practical and scalable solution for collaborative LLM inference at the edge.

Finally, while prior work, e.g., Xiao, Gao, Yang, Cao, Wang & Feng (2023b), employs VCG-based auction mechanisms to optimize social welfare in general MEC systems, their frameworks are not directly applicable to LLM inference due to the absence of layer granularity, memory constraints, and pipeline parallelism. Our FCIM mechanism addresses these limitations with cost- and memory-aware bidding, and is tailored for heterogeneous edge inference.

2.2 LLM Deployment Frameworks

Various techniques have been proposed to optimize the deployment of LLMs across distributed systems, primarily focusing on reducing latency and improving resource utilization. However, many existing approaches overlook key considerations such as fairness, dynamic adaptability, and cost-effectiveness.

PipeEdge Yuan et al. (2023) introduces a dynamic programming-based framework for optimal DNN layer partitioning across heterogeneous edge devices. While it achieves efficient load balancing under stable conditions, it assumes static device availability and workload profiles. Moreover, it ignores communication overhead—a critical factor for latency and energy efficiency—which limits its practicality for large-scale, real-world deployments.

PipeDream Narayanan et al. (2019) statically assigns DNN layers to GPUs in a fixed

pipeline order, enabling computation-communication overlap to improve throughput. However, its static offline allocation fails to accommodate runtime variability in workload or device performance. Consequently, it performs poorly in heterogeneous environments with fluctuating resource conditions and network bandwidths.

ExeGPT Oh, Kim, Kim, Kim, Lee, Chang & Seo (2024) offers a constraint-aware scheduling mechanism that considers device-specific limitations such as memory, compute, and bandwidth. Despite this, it relies on static scheduling strategies similar to PipeDream, which makes it less suited to dynamic environments where workloads and availability evolve over time.

FastServe He, Fang, Yu & Leung (2024) enhances inference for multi-model serving by adopting a multi-level feedback queue (MLFQ) scheduler. While this improves workload prioritization and latency for high-priority tasks, it can lead to task imbalance, where some devices become overloaded while others remain underutilized—especially in heterogeneous edge settings.

In contrast to these frameworks, FCIM performs layer allocation based on real-time device states and reported workloads. By considering cost, memory, and compute constraints in a dynamically adaptive manner, it enables fairer and more efficient LLM inference across heterogeneous edge environments. A detailed comparison of FCIM with existing frameworks in terms of dynamic adaptability, fairness, and incentive mechanisms is provided in Table 2.1.

Table 2.1 Comparison of FCIM with Existing LLM Deployment Frameworks (Chronological Order)

Method	**Dynamic Adaptability**	**Fairness Support**	**Incentive Alignment**	**Notes**
PipeDream Narayanan et al. (2019)	✗	✗	✗	Static pipeline assignment, offline allocation.
HexGen Jiang et al. (2023)	✓	✗	✗	Tensor partitioning adapts to device heterogeneity but lacks fairness/incentive features.
PipeEdge Yuan et al. (2023)	✓	✗	✗	Doesn't adapt to runtime resource fluctuations.
FastServe He et al. (2024)	✓	✗	✗	Adapts to latency, but not designed for fairness/incentives.
ExeGPT Oh et al. (2024)	✓	✗	✗	Constraint-aware, but fairness not enforced.
FCIM (Proposed)	✓	✓	✓	Promotes fairness and participation via adaptive reward scoring.

2.3 Limitations of Conventional Scheduling Methods

A variety of scheduling policies have been extensively studied to ensure efficient inference, including Least Laxity First (LLF), Shortest Remaining Time First (SRTF), and First-Come, First-Served (FCFS). However, each of these approaches exhibits significant drawbacks:

Least Laxity First (LLF) Yoon, Seo, Lee & Kim (2023) : LLF assigns top priority to tasks with the smallest slack time. While it effectively prioritizes critical tasks, frequent preemption leads to high context-switch overhead and fragmentation of resources, especially under heavy workloads.

Shortest Remaining Time First (SRTF) González-Rodríguez, Otero-Cerdeira, González-Rufino & Rodríguez-Martínez (2024) : SRTF selects the task with the shortest remaining execution time, often minimizing average response time. However, it may cause deadlines to be missed for longer tasks, since shorter tasks are repeatedly given precedence.

First-Come, First-Served (FCFS) Kaur & Saini (2017) : FCFS dispatches tasks in the order they arrive, thus preventing starvation. Unfortunately, it suffers from the convoy effect, where an early long-running task delays all subsequent tasks, leading to inefficient system performance.

These limitations motivate the design of our Adaptive Dynamic Scheduling Algorithm (ADSA), which we describe in detail in Section 4. Table 2.2 compares ADSA with LLF, SRTF, and FCFS.

Table 2.2 Comparison of ADSA with Conventional Scheduling Algorithms (Chronological Order)

Method	**Deadline Awareness**	**Preemption Overhead**	**Queue Management**	**Adaptive Prioritization**	**Robustness**
FCFS Kaur & Saini (2017)	✗	None	High	✗	✓
LLF Yoon et al. (2023)	✓	High	Medium	✓	✗
SRTF González-Rodríguez et al. (2024)	✗	Medium	Low	✓	✗
Optimal Scheduling	✓	Medium	High	✗	✓
ADSA (Proposed)	✓	Low	Low	✓	✓

3. SYSTEM MODEL

3.1 Network Topology

We consider a distributed inference system composed of a set of clients $\mathcal{J}$ and a set of heterogeneous worker devices $\mathcal{I}$. Each client $j \in \mathcal{J}$ generates LLM inference requests, which are forwarded to one or more selected worker devices for execution. Each request corresponds to a task $\tau_{j,m}$ that uses model $m \in \mathcal{M}$.

To handle the computational and memory demands of LLM inference, a pipeline of selected devices is formed, where each device $i \in \mathcal{I}$ processes a subset of the model's layers. After a task completes pipeline execution, the final output is transmitted from the last device in the pipeline back to the client.

The network is assumed to be connected via point-to-point communication links. Devices vary in processing speed, memory capacity, and available communication bandwidth. We assume that the set of available devices and their resource profiles (e.g., compute rate, memory, and bandwidth) are known at scheduling time. Table 4.1 summarizes the key notations used throughout the paper.

3.2 Computation and Communication Time Models

We model inference latency using transformer-specific computation and communication models, following similar approaches in Jiang et al. (2023). The computation time $T^{\text{comp}}_{i,j,m,l}$ required for worker device i to process layer l of model m for task $\tau_{j,m}$ is:

$$T^{\text{comp}}_{i,j,m,l} = \frac{12H_m^2 B_{\text{type}} S^{\text{out}}_{i,j,m,l}}{m_i} + \frac{24b_j(S^{\text{in}}_{i,j,m,l} + S^{\text{out}}_{i,j,m,l})H_m^2}{c_i}, \tag{3.1}$$

The computation time $T^{\text{comp}}_{i,j,m,l}$, defined in Equation (3.1), represents the number of time slots required by worker device i to process layer l of model m for client j. It captures the main computational stages in transformer models, specifically memory bandwidth usage and matrix multiplications.

The first term of equation (3.1), $\frac{12H_m^2 B_{\text{type}} S^{\text{out}}_{i,j,m,l}}{m_i}$, reflects the memory transfer time, where m_i refers to the bandwidth of device i, H_m refers to the hidden dimension size, B_{type} denotes the byte size of the numerical precision (e.g., FP16 or FP32), and $S^{\text{out}}_{i,j,m,l}$ denotes the output sequence length of layer l.

The second term of the equation (3.1), $\frac{24b_j(S^{\text{in}}_{i,j,m,l}+S^{\text{out}}_{i,j,m,l})H_m^2}{c_i}$, models the compute time required for matrix operations in the attention and feedforward layers. Here, b_j refers to the task's batch size, c_i denotes the compute capacity of device i (in FLOP/s), and $S^{\text{in}}_{i,j,m,l}$, $S^{\text{out}}_{i,j,m,l}$ refers to the input and output sequence lengths in bytes. Together, these terms provide a detailed formula to estimate the computation time for inference using transformer-based models on edge devices.

The equation estimates time-requiring transformer-specific parameters. By incorporating model-related factors (H_m^2, B_{type}) and device-specific capabilities (m_i, c_i), the proposed formula offers a practical and accurate approximation of computation time. Although it does not explicitly account for overheads, including memory latency or scheduling delays, these are very small in general compared to the main computational workload.

The two additive terms in (3.1) separately capture the memory-bound and compute-bound costs of processing one transformer layer. In particular,

$$\underbrace{\frac{12H_m^2\, B_{\text{type}}\, S^{\text{out}}_{i,j,m,l}}{m_i}}_{\text{memory-bound}}$$

models the time to move activation tensors of size $12H_m^2\, B_{\text{type}}\, S^{\text{out}}$ bytes through the device's memory subsystem at bandwidth m_i (bytes/s), thus isolating the data-transfer bottleneck.

The second term

$$\underbrace{\frac{24\, b_j\, (S^{\text{in}}_{i,j,m,l} + S^{\text{out}}_{i,j,m,l})\, H_m^2}{c_i}}_{\text{compute-bound}}$$

counts the total number of floating-point operations (multiplies plus adds) required—proportional to the sum of input and output token lengths, batch size b_j, and hidden dimension H_m—and divides by the device's peak compute rate c_i (FLOP/s), thereby isolating the arithmetic bottleneck. S^{in} and S^{out} are the input and output sequence lengths in bytes. Together, these two terms yield a tight estimate of T_{comp} by adding the dominant memory-transfer delay and the dominant compute delay.

The total number of time slots required by device i to process the task of the client j based on the number of layers assigned from model m is given by:

$$T_{i,j,m}^{comp} = \sum_{l \in \mathcal{L}m} T_{i,j,m,l}^{comp}, \tag{3.2}$$

The communication time $T_{i-i',j,m}^{\text{comm}}$ represents the total time required to transmit intermediate results between two consecutive devices i and i' in the pipeline while performing client j's task using model m. It is expressed as:

$$T_{i-i',j,m}^{\text{comm}} = \frac{b_j H_m B_{\text{type}} S_{i,j,m,l}^{\text{in}}}{\beta_{i-i'}(t)} + \frac{b_j H_m B_{\text{type}} S_{i,j,m,l}^{\text{out}}}{\beta_{i-i'}(t)}, \tag{3.3}$$

where the first term accounts for the transmission time during the prefill phase, transmitting input data, and the second term corresponds to the decoding phase, transmitting intermediate results. Here, $\beta_{i-i'}(t)$, the time-varying transmission rate, accounts for channel dynamics and is modeled under Rayleigh fading as:

$$\beta_{i-i'}(t) = B_i \cdot W \cdot \log_2 \left(1 + \frac{P_i h_{i-i'}(t)}{W\left(N_0 + I_{i-i'}\right)}\right), \tag{3.4}$$

where $B_i = \frac{B^S}{I}$ is the portion of resource blocks allocated to device i, W represents the bandwidth associated with each resource block, P_i denotes the transmit power of device i, N_0 is the noise power spectral density, $I_{i-i'}$ is the interference power on the link between i and i', and $h_{i-i'}(t)$ is the channel gain at time t, modeled as a Rayleigh random variable.

The Rayleigh fading model provides an excellent fit for wireless network channel behavior changes thus making it valuable for edge computing systems. The model enables researchers to properly replicate the realistic variations that occur in signal communication within dense and heterogeneous networks. Real-time operations in

such conditions benefit from the use of expected transmission rates under Rayleigh fading channel conditions.

$$\mathbb{E}[\beta_{i-i'}] = B_i \cdot W \cdot \mathbb{E}\left[\log_2\left(1 + \frac{P_i h_{i-i'}(t)}{W\left(N_0 + I_{i-i'}\right)}\right)\right]. \tag{3.5}$$

This method detects variations which occur dynamically in communication channels to give edge-based LLM inference its realistic communication model. The method utilizes predicted channel trends as a means to achieve realistic wireless performance replication through simplified processing requirements.

3.3 Task Model and Client Requests

Each inference request from client j for model m is denoted by task $\tau_{j,m}$ and characterized as a tuple:

$$\tau_{j,m} = (S_{j,m}, P_{j,m}, T_{j,m})$$

where:

- $S_{j,m}$: input size of the request, which affects both compute and memory usage;
- $P_{j,m}$: budget constraint imposed by the client for completing the inference;
- $T_{j,m}$: deadline (maximum tolerable latency) for receiving the completed inference result.

Tasks may arrive dynamically and are assumed to be non-preemptive once execution begins. However, our scheduling approach allows adaptive reordering prior to execution based on deadline and resource availability.

3.4 Pipeline Parallelism Model

When the full LLM cannot be processed by a single device due to memory or compute constraints, the task is partitioned across multiple devices using pipeline parallelism.

Each device i is assigned a segment of the model layers—denoted $l_{i,j,m}$—to process for task $\tau_{j,m}$. Devices operate sequentially on these partitions: the output of one stage becomes the input to the next.

The computation time of each stage depends on the number of assigned layers and device characteristics. The communication time between stages is governed by the intermediate output size and the bandwidth between devices.

Pipeline parallelism introduces a tradeoff between concurrency (multiple stages can process different tasks simultaneously) and communication overhead (as data must be forwarded between devices). These tradeoffs are considered in both layer allocation and scheduling decisions. Refer to Figure 3.1 for a visualization of parallel pipeline execution, where heterogeneous worker devices process layer subsets ($l_{i,j,m}$) of model m for concurrent client requests under the timing constraints defined in Equations 3.2–3.3.

3.5 Resource Heterogeneity and Constraints

Each device i is characterized by its memory capacity M_i, processing rate, and link bandwidths to other devices. Devices report their current load and availability, and may be selectively assigned tasks depending on whether their available memory can accommodate $l_{i,j,m}$ layers and whether the cumulative cost falls within the client's budget $P_{j,m}$.

All scheduling and allocation decisions must satisfy constraints on:

- *Computation time* — based on the number of layers and device-specific processing rates;
- *Communication time* — due to intermediate data transfer between pipeline stages;
- *Memory usage* — determined by the layer-specific memory model (Eq. (4.8));
- *Task deadlines and budgets.*

These constraints are formally defined in Section 4.2.

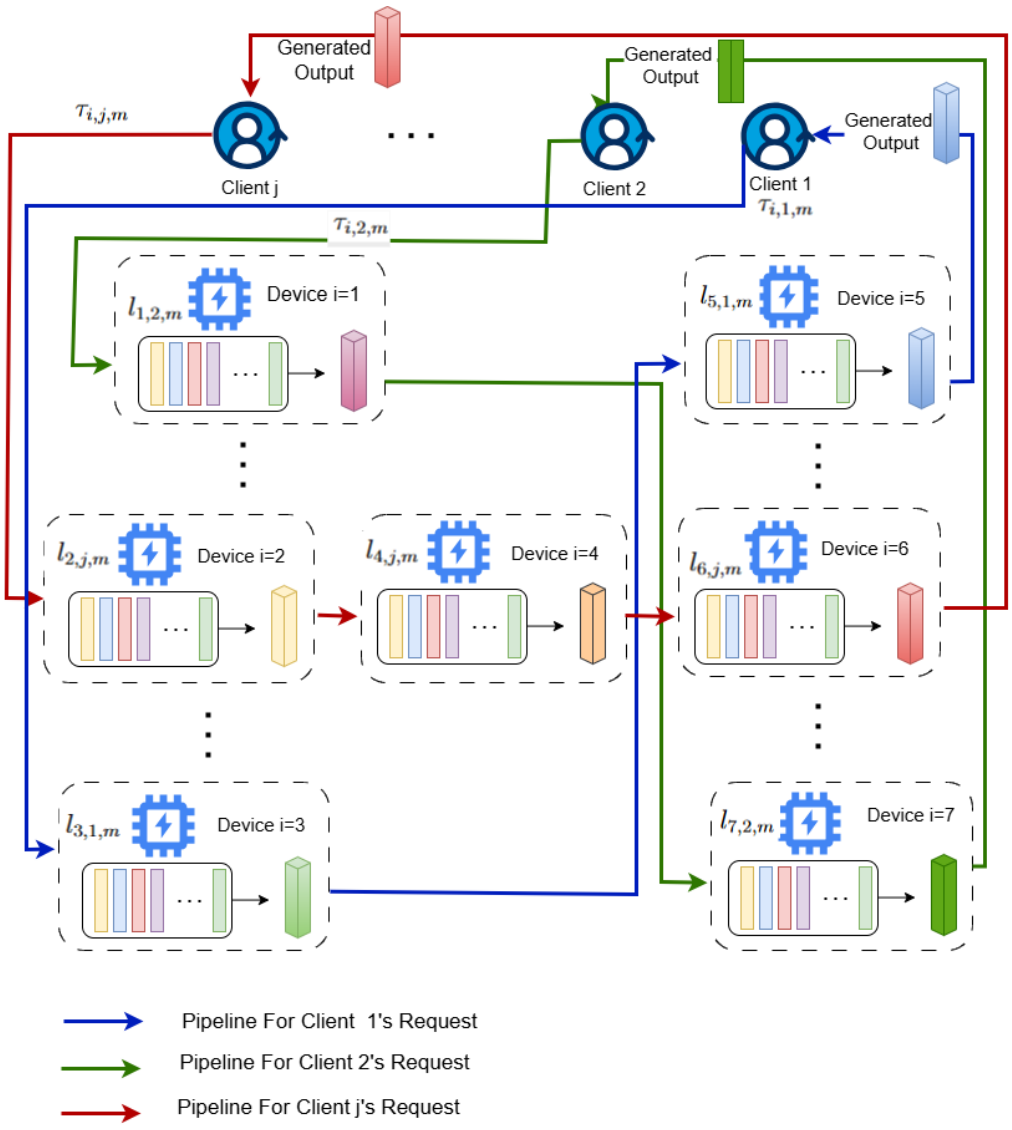

Figure 3.1 Parallel pipeline execution across heterogeneous devices for multiple clients. Each client's request (e.g., Client j) is processed through a dedicated pipeline of worker devices (e.g., Devices 2,4,6 for Client j), where each device i is allocated $l_{i,j,m}$ layers of model m to process task $\tau_{j,m}$.

4. PROBLEM FORMULATION

4.1 Problem Formulation

A time-slotted model, as detailed in Appendix B, enables task distribution to worker devices through defined variables. A primary goal of our system is to decrease the selection of devices across pipelines together with completion times while controlling the expenses the client must bear.

We define binary decision variables $y_m = (y_{i,j,m} \in \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J})$, where $y_{i,j,m} = 1$ if device i is selected to execute the task requested client j. Additionally, we introduce slot-indexed variables $x_m = (x_{i,j,m,t} \in \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, t \in \mathcal{T})$, where $x_{i,j,m,t} = 1$ if the inference task for client j is processed at worker device i during time slot S_t. The key notation used in the problem formulation is summarized in Table 4.1.

A time-slotted model, as detailed in Appendix B, enables task distribution to worker devices through defined variables. A primary goal of our system is to minimize device selection across pipelines and overall completion times while controlling the client's associated costs.

We define binary decision variables $y_m = (y_{i,j,m} \in \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J})$, where $y_{i,j,m} = 1$ if device i is selected to execute the task requested by client j. Additionally, we introduce slot-indexed variables $x_m = (x_{i,j,m,t} \in \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, t \in \mathcal{T})$, where $x_{i,j,m,t} = 1$ if the inference task for client j is processed at worker device i during time slot S_t.

We adopt a slotted-time framework in which each $x_{i,j,m,t}$ corresponds to a slot of duration S_t. To ensure that each transformer layer completes within a single slot

without introducing idle time on faster devices, we define

$$S_t = \max_{i,j,m} \left\{ T^{\text{comp}}_{i,j,m} \right\},$$

i.e., the longest per-layer execution time observed across all devices, clients, and models. This conservative choice ensures synchronization across heterogeneous devices and avoids stalling due to layer fragmentation.

The implications of this time discretization are twofold. On one hand, smaller S_t values improve temporal resolution and allow for finer-grained scheduling decisions that better adapt to dynamic workloads and device availability. On the other hand, smaller slots increase the number of time indices T, expanding the optimization space by introducing more binary decision variables and constraints, which increases the computational burden on the solver. Conversely, a larger S_t simplifies the optimization problem but may lead to underutilization of device capacity due to coarse scheduling granularity.

In our simulations, we fixed the slot duration to $S_t = 110\,\text{ms}$, which corresponds to the typical per-layer execution time observed on the slowest participating edge device. This ensures feasibility while maintaining manageable optimization complexity.

4.2 Model Constraints

The optimization problem is subject to a set of constraints governing task assignment, execution timing, memory usage, and task completion. We categorize and explain these below for clarity.

4.2.1 Task Assignment Constraints

Each worker device can process at most one task at any given time:

$$\sum_{j \in \mathcal{J}} x_{i,j,m,t} \leq 1, \quad \forall i \in \mathcal{I}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}. \tag{4.1}$$

Table 4.1 Notation Summary

Symbol	Description
$\mathcal{I}$	Set of worker devices
$\mathcal{J}$	Set of clients
$\mathcal{M}$	Set of models
$\mathcal{T}$	Set of time slots
$y_{i,j,m}$	Binary variable indicating task-device assignment
$x_{i,j,m,t}$	Task execution indicator over time
$T^{\text{comp}}_{i,j,m}$	Computation time of task j on device i
$T^{\text{comm}}_{i,j,m}$	Communication time for data transfer
L_m	Total number of layers in model m
$l_{i,j,m}$	Layers assigned to device i for client j
$M_{l_{i,j,m}}$	Memory required for layers $l_{i,j,m}$
M_i	Available memory of device i
$\phi_{j,m}$	Completion time for client j on model m
$T^{\text{total}}_{j,m}$	Total task completion time
$R^{\text{total}}_{j,m}$	Total reward cost
$P_{i,j,m}$	Payment to device i for executing j's task
$P_{j,m}$	Budget allocated by client j
α, β, γ	Optimization weights

Each client's task can be assigned to only one worker at any time:

$$\sum_{i \in \mathcal{I}} x_{i,j,m,t} \leq 1, \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}. \tag{4.2}$$

The total time slots assigned for a given task must cover its computation and communication durations:

$$\sum_{t \in \mathcal{T}} x_{i,j,m,t} = y_{i,j,m} \cdot T^{\text{comp}}_{i,j,m}, \quad \forall i, j, m, \tag{4.3}$$

$$\sum_{t \in \mathcal{T}} x_{i,j,m,t} = y_{i,j,m} \cdot T^{\text{comm}}_{i-i',j,m}, \quad \forall i, j, m. \tag{4.4}$$

4.2.2 Layer Allocation Constraint

Each model's layers must be fully distributed across the devices assigned to that pipeline:

$$\sum_{i \in \mathcal{I}} y_{i,j,m} \cdot l_{i,j,m} = L_m, \quad \forall j, m. \tag{4.5}$$

4.2.3 Completion Time Constraint

The overall completion time of a task is determined by the latest finishing device in the pipeline:

$$\phi_{j,m} \geq (t+1) \cdot x_{k,j,m,t}, \quad \text{where } k = \max\{i \mid y_{i,j,m} = 1\}. \tag{4.6}$$

The final task completion time includes downstream communication:

$$T^{\text{total}}_{j,m} = \phi_{j,m} + T^{\text{comm}}_{k-J,j,m}. \tag{4.7}$$

4.2.4 Memory Constraints

The memory required to process $l_{i,j,m}$ layers is given by:

$$\begin{aligned} M_{l_{i,j,m}} = &\left(12H_m^2 B_{\text{type}} + 2b_j(S^{\text{in}}_{i,j,m} + S^{\text{out}}_{i,j,m})H_m B_{\text{type}}\right) \cdot l_{i,j,m} \\ &+ 4b_j(S^{\text{in}}_{i,j,m} + S^{\text{out}}_{i,j,m})H_m B_{\text{type}}. \end{aligned} \tag{4.8}$$

This memory usage must not exceed the capacity of device i:

$$M_{l_{i,j,m}} \leq M_i. \tag{4.9}$$

The memory requirement $M_{l_{i,j,m}}$ for executing $l_{i,j,m}$ layers of model m on device i is derived to encompass three critical components, capturing all aspects of memory utilization in transformer-based computations. The first term, $12H_m^2 B_{\text{type}}$, represents the memory required to store the model parameters for each transformer layer. These parameters include weight matrices for the self-attention mechanism (query, key, and value matrices) and the feedforward network, both of which scale with the hidden dimension H_m. The quadratic dependence on H_m arises because transformer

layers perform matrix multiplications that involve the $H_m \times H_m$ parameter matrices.

The byte size variable B_{type} determines numeric computation precision (each value contains 4 bytes for FP32 and 2 bytes for FP16) to generate precise parameter memory estimates in each layer according to Jiang et al. (2023).

This second term describes the memory space required for storing intermediate results (activations) during forward and backward passes through the expression $2b_j(S^{\text{in}}_{i,j,m} + S^{\text{out}}_{i,j,m})H_m B_{\text{type}}$. The memory consumption of this step is defined by the multiplication of sequence sizes $S^{\text{in}}_{i,j,m}$ and $S^{\text{out}}_{i,j,m}$ with the product of the batch size b_j and hidden dimension H_m.

The entire layer memory needs amount to $12H_m^2 B_{\text{type}} + 2b_j(S^{\text{in}}_{i,j,m} + S^{\text{out}}_{i,j,m})H_m B_{\text{type}}$ because this shows how parameters and activation memory requirements combine for efficient variable-length LLM workload management.

The third memory term has the value $4b_j(S^{\text{in}}_{i,j,m} + S^{\text{out}}_{i,j,m})H_m B_{\text{type}}$ and is dedicated to activation caches that store intermediate results while conducting the inference forward pass. The activation caches require a set amount of memory to operate independently of the number of layers in order to maintain data flow in the execution pipeline.

The memory consumption analysis of transformer-based models for inference is represented in Equation (7). The memory system consists of three main elements where parameters and intermediate results and a set of fixed overhead costs are stored. The memory consumption model borrows concepts from HexGen Jiang et al. (2023) to evaluate edge inference needs precisely. Training features including checkpointing together with gradient accumulation are not part of the analysis because they are exclusive to training operations.

4.3 Optimization Problem

We formulate a multi-objective optimization problem to jointly determine: (i) the layer allocation $l_{i,j,m}$, (ii) task scheduling variables $x_{i,j,m,t}$, and (iii) device assignment indicators $y_{i,j,m}$, for executing LLM inference tasks across heterogeneous worker devices.

The goal is to minimize a weighted sum of three system-level costs: (1) the number

of selected devices, (2) total task completion time, and (3) reward payments to participating devices.

Problem 1: Multi-Objective Task Scheduling and Allocation

$$\min_{l,x,y} \quad \sum_{j\in\mathcal{J}} \left\{ \alpha \sum_{i\in\mathcal{I}} y_{i,j,m} + \beta T^{\text{total}}_{j,m} + \gamma R^{\text{total}}_{j,m} \right\},$$
$$\forall m \in \mathcal{M}$$

subject to:

$$\text{System constraints: } (4.1) - (4.7),$$
$$R^{\text{total}}_{j,m} = \sum_{i\in\mathcal{I}} y_{i,j,m} \cdot P_{i,j,m}, \quad \forall j, m, \tag{4.10}$$
$$R^{\text{total}}_{j,m} \leq P_{j,m}, \quad \forall j, m, \tag{4.11}$$
$$T^{\text{total}}_{j,m} \leq T_{j,m}, \quad \forall j, m, \tag{4.12}$$
$$x \in \{0,1\}^{|\mathcal{I}|\times|\mathcal{J}|\times|\mathcal{M}|\times|\mathcal{T}|}, \tag{4.13}$$
$$\phi_{j,m}, T^{\text{total}}_{j,m} \in [0,T], \quad \forall j, m, \tag{4.14}$$
$$y \in \{0,1\}^{|\mathcal{I}|\times|\mathcal{J}|\times|\mathcal{M}|}. \tag{4.15}$$

Here, α, β, and γ are tunable weighting coefficients that balance the trade-offs between device utilization, latency, and cost. Constraints (4.10)–(4.12) enforce per-task budget and deadline limits, while (4.13)–(4.15) define variable domains and feasibility bounds.

4.3.1 Complexity of the Allocation and Scheduling Problem

The joint optimization of task allocation, scheduling, and reward minimization is NP-hard.

We reduce from the Parallel Machine Scheduling Problem (PMSP), where n jobs are assigned to p identical machines with unit processing time to minimize makespan. We assume unit task durations and zero communication delay in Problem 1, making it equivalent to PMSP. Therefore, Problem 1 is NP-hard.

NP-Hardness Proofs for Problem 1: Problem 1 derives its computational complexity via a polynomial-time reduction from the known NP-hard Parallel Machine Scheduling Problem (PMSP). PMSP involves assigning n jobs across p parallel machines, where each machine processes only one job at a time and each job requires a

fixed processing time of $q = 1$. The main goal of PMSP is to minimize the makespan, i.e., the total time needed to complete all jobs, as discussed in the work by Pang et al. Pang, Chou & Tsai (2017).

To align Problem 1 with PMSP, we introduce two simplifying assumptions: (1) zero communication delays between worker devices, and (2) uniform task processing time of one unit for all clients. These assumptions reduce the problem scope to pure task allocation, mirroring the job distribution process in PMSP. Under these settings, the number of tasks J equals the number of devices I, and the processing time for each is uniformly $q = 1$.

This transformation shows that solving Problem 1 is at least as hard as solving PMSP since each PMSP instance can be reduced to a corresponding instance of Problem 1 in polynomial time. As PMSP is already proven to be NP-hard Pang et al. (2017); Zhao & Tang (2016), Problem 1 inherits this intractability. Consequently, the allocation and scheduling problem for LLM inference in heterogeneous edge environments remains computationally hard in general, highlighting the need for heuristic or approximation-based approaches.

Additionally, the objective of Problem 1, which is to minimize $\alpha \sum_{i \in \mathcal{I}} y_{i,j,m} + \beta T_{j,m}^{\text{total}} + \gamma R_{j,m}^{\text{total}}$, aims to jointly optimize the number of worker devices utilized $\sum_{i \in \mathcal{I}} y_{i,j,m}$, the total task completion time $T_{j,m}^{\text{total}}$, and the total rewards $R_{j,m}^{\text{total}}$. This combined optimization further underscores the computational complexity of Problem 1, reinforcing its NP-hardness, analogous to PMSP.

5. PROPOSED METHODS

5.1 Assignment Subproblem

The assignment subproblem seeks to optimize y_m (worker device assignment) and l_m (LLM layer allocation) to minimizes the worker device usage alongside task completion time and total rewards cost given to selected devices. This optimization is subject to constraints on memory capacity, budget, task allocation, and task completion time. The subproblem is formally defined as:

Subproblem 1:

$$\begin{aligned} \min_{l_m, y_m} \quad & \sum_{j \in \mathcal{J}} \left\{ \alpha \sum_{i \in \mathcal{I}} y_{i,j,m} + \beta T_{j,m}^{\text{total}} + \gamma R_{j,m}^{\text{total}} \right\}, \\ \text{Subject to:} \quad & \forall m \in \mathcal{M}, \\ & \text{Constraints:} \quad (4.5), (4.8), (4.10), (4.11), (4.15). \end{aligned}$$

5.1.1 NP-Hardness Proofs for Subproblem 1

Subproblem 1 can be shown to be NP-hard by a reduction from the well-known *Set Cover Problem*, which is a classical NP-hard problem in combinatorial optimization Iwata & Nagano (2009). The Set Cover Problem is defined as follows: Given a finite universe U and a collection $\mathcal{S}$ of subsets of U, the goal is to identify the smallest sub-collection of $\mathcal{S}$ whose union equals U.

This reduction is achieved by mapping the components of the Set Cover Problem to

the elements of Subproblem 1 as follows:

- The universe U corresponds to the set of model layers $\mathcal{L}_m$ that must be processed.
- The subsets $\mathcal{S}$ represent the available worker devices $i \in \mathcal{I}$.
- Each subset in $\mathcal{S}$ denotes the set of layers $l_{i,j,m}$ that a device i can handle for the task $\tau_{j,m}$, based on its memory and computation constraints.

The goal in Subproblem 1 is to cover all layers of the requested model m using a subset of devices, while minimizing a combination of task completion time and total reward cost. This is analogous to selecting the minimum number of subsets in Set Cover that collectively cover all elements in U.

Since any instance of the Set Cover Problem can be reduced in polynomial time to an instance of Subproblem 1, and Set Cover is known to be NP-hard, it follows that Subproblem 1 is also NP-hard.

The computational complexity of Subproblem 1 further increases due to its combinatorial nature. As the number of tasks $|\mathcal{J}|$, devices $|\mathcal{I}|$, and model layers L_m grows, the space of possible allocations expands exponentially. Additionally, optimizing over multiple performance metrics—such as task completion time $T_{j,m}^{\text{total}}$ and total reward $R_{j,m}^{\text{total}}$—makes finding exact solutions infeasible for large-scale deployments. This justifies the use of heuristic or approximation algorithms in practice.

5.2 Proposed solution for Subproblem 1: Fair Cost-Efficient Incentive Mechanism (FCIM)

FCIM is a practical two-phase auction mechanism designed to jointly allocate model layers and determine reward payments for heterogeneous edge devices. For each client request $\tau_{j,m}$, devices respond with bid tuples $b_{i,j,m} = (P_{i,j,m}, l_{i,j,m}, T_{i,j,m}^{\text{comp}})$, where $P_{i,j,m}$ is the proposed price, $l_{i,j,m}$ is the number of layers the device can process, and $T_{i,j,m}^{\text{comp}}$ is the expected compute time based on local resource availability.

Based on their capabilities, devices are categorized into two groups:

- **Full-model candidates** ($\mathcal{W}'$): Devices for which $l_{i,j,m} \geq L_m$.

- **Partial contributors** ($\mathcal{W}$"): Devices offering $l_{i,j,m} < L_m$, requiring collaborative assignment.

Each client j broadcasts its task $\tau_{j,m}$ to a subset of neighboring devices $\mathcal{W} \subseteq \mathcal{I}$, which respond with their respective bids. To evaluate bids, FCIM uses a normalized scoring function that combines cost, latency, and allocation efficiency. Specifically, scores are computed using per-layer cost, latency, and load efficiency metrics. Each term is normalized across the bidding devices using min-max scaling to $[0,1]$ to ensure comparability across heterogeneous bids.

The reward mechanism adapts to system priorities. In cost-sensitive scenarios (Case 1: $\gamma > \alpha + \beta$), the $\mathcal{W}'$ device is paid based on the second-lowest price-per-layer bid, ensuring incentive compatibility akin to a second-price auction. In latency- or memory-dominated regimes (Case 2), the reward is adjusted via a capped penalty that discourages overbidding while guaranteeing non-negative utility.

Because the score uses the ratio $C_i = P_{i,j,m}/l_{i,j,m}$ (Eqs. (5.1) and (5.2)), any deviation from the true layer capacity shifts C_i just as changing the price would. Appendix A-C proves that *in both regimes* ($\gamma > \alpha + \beta$ and $\gamma \leq \alpha + \beta$) raising C_i (by under-reporting layers) strictly reduces expected utility, while lowering C_i (by over-reporting layers) either lowers the winning probability or, if selected, yields no utility gain after the second-price or penalty-capped reward is applied. Moreover, bids that exceed a device's physical capacity violate the memory constraint $M_{l_{i,j,m}} \leq M_i$ (Eq. (4.9) and are discarded. Hence truthful reporting of both price and layer capacity is a dominant strategy, establishing a Nash equilibrium in FCIM.

Algorithm Overview

1.1 **Initialization:** Initialize selected device set $SW_{j,m} = \emptyset$, and total reward $R_{j,m}^{\text{total}} = 0$.

1.2 **Task Advertisement & Bidding:** Client j announces task $\tau_{j,m}$. Each device i responds with $(P_{i,j,m}, l_{i,j,m}, T_{i,j,m}^{\text{comp}})$.

1.3 **Device Categorization:** Place devices in $\mathcal{W}'$ or $\mathcal{W}$" based on whether $l_{i,j,m} \geq L_m$.

1.4 **Single-Device Assignment (if $\mathcal{W}' \neq \emptyset$):**

- Compute score for each $i \in \mathcal{W}'$:

$$\text{Score}_i = \beta \left(\frac{T_{i,j,m}^{\text{comp}}}{l_{i,j,m}} \right)_{\text{norm}} + \gamma \left(\frac{P_{i,j,m}}{l_{i,j,m}} \right)_{\text{norm}}. \tag{5.1}$$

- Select device i^* with minimum score and compute reward using the second-lowest bid.
- Assign task if $R_{j,m}^{\text{total}} \leq P_{j,m}$.

1.5 **Multi-Device Assignment (if $\mathcal{W}' = \emptyset$):**

- Compute scores for $i \in \mathcal{W}$":

$$\begin{aligned}\text{Score}_i = &\alpha \left(\frac{L_m}{l_{i,j,m}} \right)_{\text{norm}} + \beta \left(\frac{T_{i,j,m}^{\text{comp}}}{l_{i,j,m}} \right)_{\text{norm}} \\ &+ \gamma \left(\frac{P_{i,j,m}}{l_{i,j,m}} \right)_{\text{norm}}. \end{aligned} \tag{5.2}$$

- Iteratively select devices with lowest scores until all L_m layers are covered.
- Adjust selections if budget or deadline constraints are violated.

1.6 **Final Validation:** Ensure all constraints are satisfied; update assignments and compute total reward.

Please refer to Algorithm 1 and Algorithm 2 for details of the FCIM algorithm.

FCIM enables both *cost-awareness* and *fairness* by rewarding devices based on competition (second-best bid) and incorporating compute-time, memory limits, and task size into the bidding strategy. It ensures *individual rationality*, supports *partial participation* from diverse devices, and adapts to real-time pricing dynamics across heterogeneous edge environments.

5.2.1 Rewarding Mechanism and Utility Analysis of FCIM

To promote truthful bidding and ensure positive utility for participating devices, FCIM incorporates a reward mechanism tailored to realistic system priorities—whether dominated by cost, latency, or memory efficiency. While FCIM is not directly derived from dominant-strategy incentive-compatible (DSIC) or Bayesian incentive-compatible (BIC) auction frameworks such as VCG or Myerson mechanisms, it is pragmatically motivated by second-price logic and includes safeguards for positive utility under practical edge constraints.

We present the reward function and analyze the resulting utility under two representative configurations of the scoring weights α, β, and γ.

Algorithm 1 Fair Cost-Efficient Incentive Mechanism – Part 1: Single-Device Selection

Input: Set of Worker devices $\mathcal{I}$, Clients $\mathcal{J}$, Requested LLMs $\mathcal{M}$, Task details $\tau_{j,m} = \{S_{j,m}, P_{j,m}, T_{j,m}\}$.

Output: Selected devices $SW_{j,m}$, Assigned layers l^*_m, Device map y^*_m

Initialize $\mathcal{W}' \leftarrow \emptyset$, $\mathcal{W}" \leftarrow \emptyset$, $SW_{j,m} \leftarrow \emptyset$, $L \leftarrow 0$, $R^{\text{total}}_{j,m} \leftarrow 0$
for $j \in \mathcal{J}$ **do**
  Announce task $\tau_{j,m}$ to devices $\mathcal{W} \subseteq \mathcal{I}$; collect bids $b_{i,j,m} = \{P_{i,j,m}, l_{i,j,m}, T^{comp}_{i,j,m}\}, \forall i \in \mathcal{W}$
  for $i \in \mathcal{W}$ **do**
    if $l_{i,j,m} \geq L_m$ **then**
      $\mathcal{W}' \leftarrow \mathcal{W}' \cup \{i\}$
    else
      $\mathcal{W}" \leftarrow \mathcal{W}" \cup \{i\}$
    end if
  end for
  if $\mathcal{W}' \neq \emptyset$ **then**
    Compute score for each device using Equation (19), $\forall i \in \mathcal{W}'$
    Select $i^* = \arg\min\{\text{Score}_i\}$
    if $\gamma > \alpha + \beta$ **then**
      Set $R_{i^*,j,m}$ using Equation (22)
    else
      if $\frac{P_{i^*,j,m}}{l_{i^*,j,m}} = \min\left\{\frac{P_{i,j,m}}{l_{i,j,m}}\right\}$ **then**
        Set $R_{i^*,j,m}$ using Equation (22)
      else
        Set $R_{i^*,j,m}$ using Equation (23)
      end if
    end if
    $R^{\text{total}}_{j,m} \leftarrow R^{\text{total}}_{j,m} + R_{i^*,j,m}$
    $SW_{j,m} \leftarrow SW_{j,m} \cup \{i^*\}$, $L \leftarrow L + l_{i^*,j,m}$
    if $R^{\text{total}}_{j,m} < P_{j,m}$ **then**
      Set $y_{i^*,j,m} = 1$, $y_{i,j,m} = 0$, $\forall i \in \mathcal{W}' \setminus \{i^*\}$
    end if
  end if
end for

Algorithm 2 Fair Cost-Efficient Incentive Mechanism – Part 2: Multi-Device Selection

Input: Set of Worker devices $\mathcal{I}$, Clients $\mathcal{J}$, Requested LLMs $\mathcal{M}$, Task details $\tau_{j,m} = \{S_{j,m}, P_{j,m}, T_{j,m}\}$, Device responses $\{P_{i,j,m}, l_{i,j,m}, T_{i,j,m}^{comp}\}$ for all $i \in \mathcal{W}$", Allocation state from Algorithm 1

Output: Remaining allocations for $SW_{j,m}$, y_m^*, $R_{j,m}^{\text{total}}$

1: **for** $j \in \mathcal{J}$ where $\mathcal{W}' = \emptyset$ **do**
2: Compute score for each device using Equation (20), $\forall i \in \mathcal{W}$"
3: Sort $\mathcal{W}$" by ascending Score_i
4: Assign devices from sorted $\mathcal{W}$" to $SW_{j,m}$ until $L \geq L_m$
5: **for** each selected $z \in \mathcal{W}$" **do**
6: **if** $\gamma > \alpha + \beta$ **then**
7: Set $R_{z,j,m}$ using Equation (22)
8: **else**
9: **if** $\frac{P_{z,j,m}}{l_{z,j,m}} = \min\left\{\frac{P_{i,j,m}}{l_{i,j,m}}, \forall i \in \mathcal{W}"\right\}$ **then**
10: Set $R_{z,j,m}$ using Equation (22)
11: **else**
12: Set $R_{z,j,m}$ using Equation (23)
13: **end if**
14: **end if**
15: Update $L \leftarrow L + l_{z,j,m}$, $R_{j,m}^{\text{total}} \leftarrow R_{j,m}^{\text{total}} + R_{z,j,m}$
16: Compute $T_{j,m}^{\text{total}}$
17: **if** $T_{j,m}^{\text{total}} > T_{j,m}$ **or** $R_{j,m}^{\text{total}} > P_{j,m}$ **then**
18: Remove z from $SW_{j,m}$ and revert to line 3
19: **else**
20: Set $y_{z,j,m} = 1$, $y_{i,j,m} = 0$, $\forall i \in \mathcal{W}" \setminus \{z\}$
21: **end if**
22: **end for**
23: **end for**

Reward Function and Device Utility Recall that device scores are computed using (5.2), where:

- α emphasizes layer-handling efficiency,
- β prioritizes computation time efficiency,
- γ captures cost-per-layer efficiency.

The utility of a winning device i^* is defined as:

$$U_{i^*,j,m} = R_{i^*,j,m} - P_{i^*,j,m}, \tag{5.3}$$

where $R_{i^*,j,m}$ is the reward received and $P_{i^*,j,m}$ is the device's declared cost.

Case 1: Cost-Dominated Settings ($\gamma > \alpha + \beta$)

When cost-per-layer is the dominant priority, the score is minimized primarily through the term $\frac{P_{i,j,m}}{l_{i,j,m}}$, and the $\mathcal{W}'$ device i^* is the lowest-cost bidder. The reward follows a second-price structure:

$$R_{i^*,j,m} = \frac{P_{v,j,m} \cdot l_{i^*,j,m}}{l_{v,j,m}}, \tag{5.4}$$

where v denotes the second-best bidder. This formulation guarantees positive utility since the $\mathcal{W}'$ device's payment is benchmarked to a higher or equal cost-per-layer from another bidder.

Case 2: Latency- or Memory-Dominated Settings ($\gamma \leq \alpha + \beta$)

In this regime, scoring is influenced more heavily by layer-handling and computation time. As a result, the winning device may not be the cheapest, and the second-price formula in Equation (5.4) can result in negative utility.

To address this, we modify the reward function to incorporate both a protective lower bound and a penalization mechanism that discourages overbidding:

$$
\begin{aligned}
R_{i^*,j,m} = &\max\left(\frac{P_{v,j,m}\cdot l_{i^*,j,m}}{l_{v,j,m}},\ P_{i^*,j,m}\right) \\
&-\max\left(0,\ \min\left(\delta\cdot l_{i^*,j,m}\cdot\left(\frac{P_{i^*,j,m}}{l_{i^*,j,m}}-P_{\text{avg}}\right),\ R_{\text{safe}}\right)\right)
\end{aligned}
\tag{5.5}
$$

where P_{avg} is the average cost-per-layer of all other bidders:

$$
P_{\text{avg}} = \frac{1}{|\mathcal{W}|-1}\sum_{z\neq i^*}\frac{P_{z,j,m}}{l_{z,j,m}},
$$

and R_{safe} is an upper bound on the penalty, ensuring non-negative utility:

$$
R_{\text{safe}} = \max\left(\frac{P_{v,j,m}\cdot l_{i^*,j,m}}{l_{v,j,m}},\ P_{i^*,j,m}\right) - P_{i^*,j,m}.
$$

This reward formula ensures:

- A lower bound at the device's declared cost,
- A penalty only if the proposed cost-per-layer exceeds P_{avg},
- A capped penalty to preserve individual rationality.

In (5.5) δ parameter appears within the penalty component:

$$
\delta\cdot l_{i^*,j,m}\cdot\left(\frac{P_{i^*,j,m}}{l_{i^*,j,m}}-P_{\text{avg}}\right),
$$

which quantifies how much more (per-layer) a winning device has requested compared to the system-wide average. The parameter $\delta\in\mathbb{R}_{\geq 0}$ is a *dimensionless* penalization coefficient that linearly scales this deviation. In other words, one unit of δ increases the per-layer penalty by one credit for every unit of cost inflation. Conceptually, δ represents the fraction of cost inflation that the system chooses to penalize.

We emphasize that δ only affects the *magnitude* of the penalty and does not alter the structure of the reward formulation or the truthfulness guarantees. That is, the mechanism remains incentive-compatible for any nonnegative value of δ, so long as the total penalty remains bounded by R_{safe}.

Please refer to Chapter 6 for the impact of the penalty parameter δ on performance metrics, including total cost, fairness indices for rewards and layers, and total processing time.

5.2.2 Proof of Positive Utility

5.2.2.1 Proof of Positive Utility in FCIM for Case1

When cost efficiency is prioritized, and γ dominates $\alpha+\beta$, the optimization problem gives more importance to the cost-per-layer term $\frac{P_{i,j,m}}{l_{i,j,m}}$. The winning device i^* is therefore selected based on its ability to provide the lowest cost-per-layer, denoted as:

$$C_{i^*} = \frac{P_{i^*,j,m}}{l_{i^*,j,m}}. \tag{5.6}$$

The reward for the winner is determined by the second winner's price-per-layer:

$$C_v = \frac{P_{v,j,m}}{l_{v,j,m}}, \tag{5.7}$$

Since the winner i^* has the lowest cost-per-layer C_{i^*}, it follows that $C_v > C_{i^*}$ for the second winner v. Consequently:

$$\frac{P_{v,j,m}}{l_{v,j,m}} > \frac{P_{i^*,j,m}}{l_{i^*,j,m}}. \tag{5.8}$$

When scaled to the layers handled by the winner, the reward ensures:

$$R_{i^*,j,m} = \frac{P_{v,j,m} \cdot l_{i^*,j,m}}{l_{v,j,m}} > P_{i^*,j,m}. \tag{5.9}$$

This inequality ensures that the reward always exceeds the cost of the winner, and consequently, the utility is always positive.

5.2.2.2 Proof of Positive Utility for Case2

The rewarding formula proposed in Equation (5.5) provides a guaranteed positive reward value to the winner device. The first term provides the device with a minimum payment that exceeds its proposed or the scaled value of the second winner's price, thus establishing a minimum payment threshold. The penalty term penalizes the given reward when the winner's proposed price per layer surpasses the average

proposed price per layer of the participants. The deliberate penalty enforces R_{safe}, defined as the difference between the winner's baseline reward and its own proposed price, which works as an upper threshold on the penalty amounts and ensures even in the worst-case scenario, the penalty cannot reduce the reward below the winner's bid since the utility (reward minus bid) remains positive. This structure guarantees that the winning device always receives at least as much as its bid, providing positive utility and preserving the incentive for honest participation.

5.2.3 Truthfulness and Nash Equilibrium in FCIM

5.2.3.1 Proof of Truthfulness for Case1

In Case 1 when $\gamma > \alpha + \beta$ cost efficiency dominates the optimization problem, the score is dominated by the cost-per-layer term $\frac{P_{i,j,m}}{l_{i,j,m}}$, and devices with the lowest cost-per-layer have a higher likelihood of winning. Consider a device i with a truthful bid $P_{i,j,m}$ and an alternative strategy of overbidding $P'_{i,j,m} > P_{i,j,m}$ or underbidding $P'_{i,j,m} < P_{i,j,m}$:

- *Overbidding:* When the device overbids, its cost-per-layer increases as $C'_i = \frac{P'_{i,j,m}}{l_{i,j,m}} > \frac{P_{i,j,m}}{l_{i,j,m}}$. Since the score is primarily influenced by the cost-per-layer, the increased C'_i reduces the likelihood of winning. Even if the device wins, the reward $R_{i^*,j,m}$ remains tied to the second winner's price and the utility $U_{i^*,j,m} = R_{i^*,j,m} - P'_{i,j,m}$ is lower than truthful bidding because $R_{i^*,j,m}$ is unaffected by $P'_{i,j,m}$.

- *Underbidding:* When the device underbids, $C'_i = \frac{P'_{i,j,m}}{l_{i,j,m}} < \frac{P_{i,j,m}}{l_{i,j,m}}$, which improves the score and increases the chances of winning. However, the reward $R_{i^*,j,m}$ remains tied to the second winner's price, and there is no additional gain in utility. Hence, underbidding does not provide any benefit.

Truthful bidding maximizes the utility and is therefore a dominant strategy.

5.2.3.2 Proof of Nash Equilibrium for Case1

For truthful bidding to be a Nash equilibrium, no device can unilaterally improve its utility by deviating. When $\gamma > \alpha + \beta$, the reward ensures $R_{i^*,j,m} \geq P_{i^*,j,m}$ and any deviation from truthful bidding (overbidding or underbidding) as mentioned earlier either reduces the chances of winning or does not increase the utility. Thus, truthful bidding satisfies Nash equilibrium.

Truthful bidding ensures a Nash equilibrium when no device can gain more utility by changing its bid alone. When $\gamma > \alpha + \beta$, the reward guarantees that the winner earns at least its bid. Any deviation (overbidding or underbidding) either decreases the chance of winning or does not improve the utility. Therefore, truthful bidding remains the best strategy, satisfying the Nash equilibrium condition.

5.2.3.3 Proof of Truthfulness for Case2

In this section, the truthfulness of the rewarding mechanism using (20) will be proved as follows:

Truthful Bidding If a device proposes its true cost-per-layer and if it wins, its utility is obtained as:

$$U_{i^*,j,m} = R_{i^*,j,m} - P_{i^*,j,m}$$

Since the first term in (20) guarantees the reward is at least the device's bid, and the second term only applies a penalty if the bid is above average, a truthful bid receives either no penalty (if proposed cost-per-layer $\leq P_{\text{avg}}$), resulting in full reward, or it will receive a bounded penalty (if cost-per-layer $> P_{\text{avg}}$), that still guarantees non-negative utility due to the R_{safe} cap.

Therefore, truthful bidding either results in maximum or fair reward with safe utility.

Underbidding Now assume the device reduces its proposed bid: $P'_{i^*,j,m} < P_{i^*,j,m}$. Then its score increases, improving its chances of winning. If it wins, it receives a reward at least equal to its proposed bid. If $P'_{i^*,j,m} > P_{\text{avg}}$, the penalty still applies. If $P'_{i^*,j,m} < P_{\text{avg}}$, the penalty is zero, and the reward is determined based on the first term and it receives at a maximum according to the second winner's proposed price. Hence, underbidding offers no gain.

Overbidding Suppose a device overbids by $P'_{i^*,j,m}$, where $P'_{i^*,j,m} > P_{i^*,j,m}$. Then its score decreases, reducing its chances of winning, and if it still wins, its proposed

price $P'_{i^*,j,m}$ potentially exceeds P_{avg}, causing a larger penalty.

Hence, overbidding will not be beneficial.

Thus, in all three analyzed cases, including truthful, overbidding, and underbidding, truthful bidding either gives the highest or safest utility for the winning device. However, dishonest bidding lowers the chance of winning or the utility. So, the reward mechanism encourages truthful bids, proving it is truthful. Hence, the equation (5.5) incentivizes truthful bidding.

5.2.3.4 Proof of Nash Equilibrium for Case 2

To prove that Equation (5.5) satisfies Nash equilibrium, we must show that any winner can not receive higher utility by unilaterally deviating from its truthful bidding strategy.

Nash Equilibrium Definition: A Nash equilibrium exists in bidding when each device maintains its current bid strategy because no one can achieve higher utility from altering their price while other devices remain static.

As shown in the proof of truthfulness, the reward formula ensures that:

- The device receives the complete reward without any penalty when its price-per-layer proposal is equal or lower than the average value of P_{avg}.
- The device receives a bounded penalty which is limited by R_{safe} when its price-per-layer exceeds P_{avg}.

In both cases, bidding truthfully gives the highest or at least a safe, non-negative utility. A winning device that places a higher bid than its competitors will experience a reduction in its score because of which it faces lower chances of winning. When a device underbids, it earns better score to win and if it wins, a penalty remains active when the underbid extends beyond P_{avg} threshold. The reward provided with such underbids becomes so small that utility can turn out to be less suitable than honest bidding competition.

No device following a rational strategy will deviate from their truthful bid since any deviation leads to either a reduced utility or lower chances of winning in the mechanism. In Case 2 the truthful bidding strategy establishes a Nash equilibrium.

5.3 Scheduling Subproblem

Given the optimal device assignment y_m^* and layer allocation l_m^* obtained via the FCIM mechanism, we now aim to determine an execution schedule x_m that minimizes total completion time while respecting resource, communication, and deadline constraints.

Subproblem 2: Task Scheduling Optimization

$$\begin{aligned} \min_{x_m} \quad & \sum_{j \in \mathcal{J}} \left\{ \beta T_{j,m}^{\text{total}} + \gamma R_{j,m}^{\text{total}} \right\}, \quad \forall m \in \mathcal{M} \\ \text{Subject to:} \quad & (4.1)\text{–}(4.4), (4.6), (4.9), (4.12)\text{–}(4.14) \end{aligned}$$

Note that since the assignments y_m^* and l_m^* are fixed, the term involving $\sum_i y_{i,j,m}$ in the original objective becomes constant and is excluded from this subproblem.

Subproblem 2 is a generalization of classical multiprocessor scheduling with added complexity from:

- inter-device communication delays,
- deadline constraints,
- and layer-wise dependencies across a distributed pipeline.

The scheduling subproblem with communication and computational constraints resembles classical dependent task scheduling with communication delays, shown to be NP-hard Ali & El-Rewini (1993); Eisenbrand & Rothvoß (2008).

5.3.1 NP-Hardness Proofs for Subproblem 2

The computational difficulty of Subproblem 2 can be determined through established scheduling theory findings. Multiple journal articles demonstrate that scheduling systems consisting of communication and computational constraints remains a computationally challenging problem. The incorporation of non-uniform execution times and communication delays turns dependent task scheduling into an NP-hard problem according to Ali and El-Rewini Ali & El-Rewini (1993). The calculation of response time becomes an NP-hard problem when performed on systems that

have communication overhead according to Eisenbrand and Rothvoss Eisenbrand & Rothvoß (2008) particularly when completion times optimization is pursued. Norman et al. Norman, Pelagatti & Thanisch (1995) established that the scheduling process across multiple devices transforms into NP-hard problems when the objective is to minimize makespan despite incorporating processing time together with inter-device communication.

Subproblem 2 contains several aspects from previous models while adding features for pipeline parallel execution and integrated computation and communication delays across disparate devices. The research studied static scheduling frameworks but Subproblem 2 introduces additional complexity because it requires handling dependent relationships among sequential layers that spread across several edge devices. The architecture requires each device to process a model segment followed by passing its output to the subsequent device thus creating intricate relations among task scheduling.

Extensive search becomes impossible due to the rapidly multiplying number of feasible scheduling options when tasks and devices rise in numbers. Additionally the problem becomes more challenging with dynamic task arrival patterns and diverse device capabilities and strict deadline constraints. Large-scale implementations prevent the use of exact solutions to address Subproblem 2. The evaluation demonstrates that Subproblem 2 belongs to the NP-hard classification which highlights why practical deployment requires approximation methods instead of exhaustive search.

To address this, we propose a preemptive scheduling algorithm—ADSA—that dynamically assigns time slots to tasks while minimizing the overall completion time. ADSA accounts for task arrival times, device availability, execution durations, and communication delays. By prioritizing tasks nearing their deadlines and limiting preemptions, the algorithm balances efficiency and deadline compliance in real-time edge inference scenarios.

5.3.2 Computing the Arrival Time for Selected Worker Device

Consider $SW_{j,m} = \{d_1, d_2, \ldots, d_h, \ldots, d_w\}$ as the set of sekectet worker devices using FCIM for client j. This set ordered such that device d_i precedes d_{i+1}. The computation time at each selected device d_i, $T^{\text{comp}}_{d_i,j,m}$, is calculated using (3.2) and the transmission time between consecutive devices d_i and d_{i+1}, $T^{\text{comm}}_{d_i-d_{i+1},j,m}$, is obtained

using (3.3) The arrival time of the task from client j processed by LLM m at device d_h can be calculated as follows:

$$T_{h,j,m}^{\text{arrival}} = T_{j-d_1,j,m}^{\text{comm}} + \sum_{i=1}^{h-1} \left(T_{d_i,j,m}^{\text{comp}} + T_{d_i-d_{i+1},j,m}^{\text{comm}} \right),$$

where:

- $\sum_{i=1}^{h-1} T_{d_i,j,m}^{\text{comp}}$ represents the total computation time of devices from d_1 to d_{h-1} in $SW_{j,m}$,
- $\sum_{i=1}^{h-1} T_{d_i-d_{i+1},j,m}^{\text{comm}}$ represents the total transmission time between consecutive devices from d_1 to d_h,
- $T_{j-d_1,j,m}^{\text{comm}}$ denotes the transmission time from client j to the first selected device d_1 in the set $SW_{j,m}$.

Given a set of tasks $\tau_m = \{\tau_{1,m}, \tau_{2,m}, \ldots, \tau_{J,m}\}$ from J clients, each task $\tau_{j,m}$ is defined by the triple

$$\tau_m = (T_{i,j,m}^{\text{arrival}}, T_{i,j,m}^{\text{comp}}, T_{i,j,m}^{\text{comm}}).$$

The objective is to schedule tasks on worker device i to minimize total task completion time. To achieve this, the Adaptive Dynamic Scheduling Algorithm (ADSA) is proposed, which optimally schedules tasks by dynamically adapting to varying computational and communication demands. While based on classical real-time scheduling principles (e.g., LLF, SRTF), ADSA integrates them in a task-aware, memory- and transmission-conscious way tailored for pipeline-parallel LLM inference. ADSA prioritizes deadline-critical tasks, while defaulting to FCFS for non-critical tasks under uniform arrivals. In contrast, when the arrival tasks follow a non-uniform distribution, it starts with the task that has the longest transmission time (LTT) to guarantee that tasks with higher transmission time requirements meet their deadlines. ADSA shows excellent suitability for real-time edge computing due to its ability to work with various workload patterns.

The ADSA algorithm is provided in Algorithm 3.

5.3.3 Overcoming the Limitations of Conventional Scheduling Methods

The Adaptive Dynamic Scheduling Algorithm (ADSA) addresses the limitations of conventional scheduling approaches by combining their strengths while mitigating

Algorithm 3 Adaptive Dynamic Scheduling Algorithm (ADSA)

Require: Task list $\tau_i = \{\tau_{i,j}\}$ for worker i, where each task $\tau_{i,j} = (T^{\text{arrival}}, T^{\text{comp}}, T^{\text{comm}}, T_j)$ includes arrival time, computation time, communication time, and deadline.

Ensure: Optimal schedule x_m^*, total completion time T_j^{total}.

Initialize $t \leftarrow 0$, $Q \leftarrow \emptyset$, $x_m^* \leftarrow \emptyset$, $T_j^{\text{total}} \leftarrow 0$
while τ_i or Q is not empty **do**
  Add tasks with $T^{\text{arrival}} \leq t$ to Q
  Compute completion times $C_{i,j}$ and urgency $\Delta_{i,j} = T_j - C_{i,j}$
  Select Task:
  if $\exists \Delta_{i,j} < 0$ **then**
    $\tau_{\text{selected}} \leftarrow \arg\min(\Delta_{i,j})$ {Prioritize urgent task}
  else if τ_{current} is running **then**
    $\tau_{\text{selected}} \leftarrow \tau_{\text{current}}$
  else
    Compute $\text{variance}_{\text{comp}}, \text{variance}_{\text{comm}}$
    $\tau_{\text{selected}} \leftarrow$ FCFS if uniform, else LTT strategy
  end if
  Execute:
  Update $t \leftarrow \max(t, T_{\text{selected}}^{\text{arrival}})$
  Set start time $t_{\text{start}} \leftarrow t$
  Execute τ_{selected} and update $t \leftarrow t + T_{\text{selected}}^{\text{comp}}$
  Remove task if computation is complete
  if Computation is finished **then**
    Start transmission and update T^{total}
    Compute waiting time w_{selected} and store in W
  end if
end while
return x_m^*, T^{total}

key drawbacks—particularly in the context of edge-based LLM inference.

Unlike LLF, which incurs high preemption overhead due to frequent context switches, ADSA employs a low-preemption strategy that significantly reduces switching costs. It features an adaptive prioritization mechanism that dynamically adjusts task execution based on real-time task arrivals and system state, ensuring robustness under variable loads and device availability.

ADSA further improves efficiency by aligning task execution with inter-task communication delays, enabling effective overlap of computation and communication. This makes it well-suited for resource-constrained, high-load edge environments.

Task selection in ADSA is based on:

Slack Time : For each task $\tau_{i,j}$, it computes $\Delta_{i,j} = T_j - (t + T_{i,j}^{\text{comp}} + T_{i,j}^{\text{comm}})$. If any $\Delta_{i,j} < 0$, the most urgent task is prioritized.

Variance-Based Strategy Switching : If no task is urgent, ADSA computes the variance of compute and communication times across the queue. If the variance is below a threshold ϵ, it applies First-Come-First-Serve (FCFS); otherwise, it switches to Longest Transmission Time (LTT) scheduling.

In contrast to SRTF, which may starve long-duration tasks, ADSA maintains fairness by balancing the scheduling of both short and long tasks.

5.4 Complexity Analyze

The FCIM+ADSA framework unifies two elements that combine the Fair Cost-Efficient Incentive Mechanism (FCIM) for resource allocation with the Adaptive Dynamic Scheduling Algorithm (ADSA) for task scheduling on each device.

5.4.1 FCIM Complexity

The Fair Cost-Efficient Incentive Mechanism's function involves assessing devices along with sorting them while performing checks on allocation limits. FCIM + ADSA requires $O(J \cdot I^3)$ complexity for dominant costs that include the execution

time of sorting and scoring measures when J represents the client numbers and I stands for edge device numbers.

5.4.2 ADSA Complexity

End devices manage their workload through ADSA which performs live task scheduling operations for each device along with queue administration functions as well as deadline enforcement and scheduling priority system. Sorting the task queue consumes the greatest amount of overhead and generates $O(N^2)$ complexity when there are N tasks.

5.4.3 Combined Complexity

FCIM+ADSA delivers combined complexity of $O(J \cdot I^3 + N^2)$ which constitutes a real-time feasible polynomial function. The heuristic nature of the framework allows near-optimal performance with scalability and efficiency when run on dynamic edge systems because it bypasses the exponential complexity of ILP/MILP ($O(N^3 \cdot 2^N)$).

FCIM+ADSA functions differently from static models including Megatron-LM Shoeybi et al. (2019) or PipeDream Narayanan et al. (2019) because it uses real-time resource availability to dynamically allocate layers and tasks. Dynamic allocation functions in the algorithm lower the delays caused by workload imbalances and staleness in distributed processing. Static workload distribution within FCIM+ADSA system enables improved performance of heterogeneous pipelines across devices.

Genetic algorithms make dynamic scheduling more popular due to their ability to produce high-quality near-optimal solutions when using crossover and mutation patterns Akbari, Rashidi & Alizadeh (2017); Page & Naughton (2005). GAs require substantial computational resources due to their complexity $O(N \times g)$ which includes g generations and therefore are not ideal for dynamic systems that need immediate adjustments.

The FCIM+ADSA system performs near-optimal resource allocation and task scheduling through its hybrid auction protocol and real-time scheduling approach at decreased computational complexity rating of $O(J \cdot I^3 + N^2)$. Because of its design FCIM+ADSA provides fast reaction to changing workload requirements together

with rapid convergence speed superior to GAs. The FCIM mechanism provides fair pricing benefits for devices alongside cost efficiency then ADSA maintains real-time task execution deadlines. Real-time LLM inference with FCIM+ADSA operates as a scalable practical solution for distributed edge systems.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

This subsection outlines the experimental environment used to evaluate the FCIM and ADSA components. We describe the hardware infrastructure, the LLM configurations, the nature of task arrivals, and the evaluation parameters.

6.1.1 Hardware Setup

Experiments were conducted on:

- **6 NVIDIA GPUs** (3x RTX 4090 with 24GB VRAM, 3x T4 with 16GB VRAM),
- **CUDA Version**: 12.1, **PyTorch Version**: 2.0,
- **Connection**: We assumed a point-to-point connection topology among the GPUs. The communication time between each GPU pair was empirically estimated as the average of several experiments conducted under Rayleigh fading channel conditions.

Energy usage was monitored using `nvidia-smi` with 1-second sampling intervals. GPU memory utilization and compute time were recorded during inference.

6.1.2 Model Setup

We used three different LLMs in our experiments:

- **BLOOM-176B** (BigScience): For evaluating energy consumption and inference time using the FCIM bidding framework. The model was loaded with 70 layers.
- **GPT-Neo-1.3B** (EleutherAI): Used for testing the FCIM and ADSA scheduling algorithm under various deadlines and arrival patterns.
- **GPT-3 (13B)**: Used as another reference model in comparative tests in FCIM.

6.1.3 Task Types and Arrival Patterns

We simulated five representative tasks with diverse sizes, arrival intervals, transmission costs, and deadlines. Each task corresponds to a unique LLM prompt such as:

```
"Analyze the implications of quantum computing on
cybersecurity."
```

Tasks varied from short summaries (200–300 words) to extended essays (up to 2200 words). Arrival times ranged from 0 to 8 units, and deadlines were set between 20 and 100 time units to emulate real-time constraints.

6.1.4 Parameter Table

Table 6.1 summarizes the key parameters used in FCIM and ADSA evaluations.

Table 6.1 Experimental Parameters

Parameter	Value / Description
Number of GPUs	6 (3x RTX 4090, 3x T4)
Task count	5
Model size (FCIM)	BLOOM-167B with 70 layers, GPT3 (13B) with 40 layers, and GPT-Neo (1.3B) with 24 layers
Model size (ADSA)	GPT-Neo 1.3B
Arrival times	[0, 3, 4, 6, 8]
Output words per task	[1200, 300, 1800, 2200, 200]
Transmission cost	[1–7 units]
Deadlines	[20–100 units]
FCIM weights	$\alpha = 0.1, \beta = 0.3, \gamma = 0.6$

6.1.5 Simulation Structure

This chapter presents simulation results in two main parts:

- **FCIM Evaluation:** The first part evaluates the proposed Fair Cost-Efficient Incentive Mechanism by comparing it with benchmark methods from the literature. These include classic allocation heuristics for load distribution and LLM inference across heterogeneous devices under budget constraints.
- **ADSA Evaluation:** The second part focuses on task scheduling. We demonstrate how the Adaptive Deadline-aware Scheduling Algorithm (ADSA) outperforms both static and dynamic baselines in terms of deadline satisfaction and memory efficiency.

6.2 Effectiveness of FCIM in Assignment Tasks

We evaluate FCIM under two bidding scenarios. In **Scenario S1**, GPU prices reflect real-world availability, simulating market-driven bids based on resource capacity. **Scenario S2** introduces a more aggressive setting, emphasizing competitive bidding by high-end devices such as RTX 4090 GPUs, to assess FCIM's robustness under skewed participation conditions.

6.2.1 Experimental Setup for FCIM

The experimental evaluation of FCIM resource efficiency involves three prepared transformer language models including GPT-Neo (1.3B), GPT-3 (13B), and BLOOM (176B). EleutherAI created GPT-Neo which stands as a free-to-use lightweight foundation model. The mid-tier GPT-3 (13B) exists through API-limited access but BLOOM from BigScience functions as a high-resource benchmark because it contains extensive parameter count and layer configuration.

- Task Characteristics: Five subtasks appear in the simulation which contains clear output specifications for word count demands as well as network time requirements and time constraints. Longer output texts for tasks need additional processing time and memory that affects scheduling decisions. The simulation includes deadlines and transmission times to replicate the real-time conditions that occur during LLM inference operations.

- GPU Bidding and Allocation Framework: A system with six GPUs incorporates both RTX 4090 and T4 models which vary by their memory resources and processing capabilities. GPU memory availability determines the calculation from total available memory minus the amount of memory being used at present.

- Bidding Strategy and Proposed Price Calculation:

  To generate bids, each GPU calculates its price per model layer, denoted as $P_{i,j,m}$, based on its available memory and computational capacity. The pricing considers the following parameters:

  - Total Task Size $S_{j,m}$ (in bytes): Represents the full workload for task j using model m.
  - Available Memory M_i (GB):The remaining memory on GPU i after deducting current usage.
  - Computational Bandwidth B_i (TFLOP/s): Processing performance of GPU i.
  - Scaling Factor C: A tunable constant to adjust the sensitivity of pricing.

  The formula for computing the price per layer is:

$$P_{i,j,m} = \frac{S_{j,m}}{M_i \times B_i} \times C \tag{6.1}$$

In this equation, $S_{j,m}$ reflects the size of the workload, M_i indicates how much memory the device has available, B_i refers to its compute speed, and C controls the pricing granularity.

Based on this model, GPUs with less available memory tend to propose higher prices because limited memory increases the cost. For example, an RTX 4090 with lower available memory bids higher than one with more memory. Likewise, T4 GPUs, which generally have lower compute power, also submit higher bids. However, within any GPU type, a device with less memory results in a higher price per layer.

This pricing approach favors GPUs with more resources, making them appear more cost-efficient, which fits the intended behavior of Scenario S1.

To test how well this pricing model works under different conditions, Scenario S2 is introduced. It applies a more competitive strategy, particularly for high-performance GPUs like the RTX 4090. By comparing the outcomes of S1 and S2, we evaluate how FCIM handles various market conditions and resource constraints, focusing on both adaptability and fairness.

In our experimental setup, we used the scoring weights $\alpha = 0.1$, $\beta = 0.3$, and $\gamma = 0.6$. These parameters define the priority balance between layer-handling efficiency, latency, and cost in the FCIM scoring function in (5.2).

These values were selected based on empirical performance profiling and reflect a regime where cost-per-layer (γ) is prioritized without fully ignoring latency and task fairness. Specifically, the configuration satisfies $\gamma > \alpha + \beta$, classifying it as a cost-dominated scenario and invoking the reward formulation of Equation (5.4). This prioritization aligns with typical federated edge environments, where resource-constrained systems often favor budget efficiency.

To determine these values, we conducted a grid search across several configurations of $\alpha + \beta + \gamma = 1$. We observed that:

- Raising γ above 0.6 consistently skewed allocation toward the cheapest (often slowest) devices, increasing deadline violations.
- Increasing β above 0.4 led to excessive emphasis on fast devices, marginalizing low-cost participants and reducing overall fairness.
- Assigning α above 0.2 caused unstable task splitting, fragmenting layer allocations across too many GPUs.

The chosen values $(0.1, 0.3, 0.6)$ yielded a balanced trade-off: meeting deadlines,

preserving fairness (as shown by fairness indices in Section VII), and maintaining cost-efficiency. These outcomes are visible in Figures 6.3–6.7.

We emphasize that these values are not fixed in the FCIM design. The scoring weights are intentionally tunable, allowing system administrators to shift priorities depending on real-time constraints — such as minimizing latency in live inference or maximizing cost-efficiency in batch deployments.

In our experiments FCIM is compared against three representative baseline methods: PipeEdge Yuan et al. (2023), PipeDream Narayanan et al. (2019), and HexGen Jiang et al. (2023). PipeEdge employs dynamic programming for task partitioning but lacks fairness and cost-awareness. PipeDream adopts a static pipeline parallelism approach that limits flexibility under dynamic conditions. HexGen improves inference latency through asymmetric tensor parallelism, yet does not address fairness or incentive compatibility.

The comparison focuses on several key performance metrics: total task completion time, total rewards distributed to participating devices, inter-device communication overhead, average memory usage, cost-effectiveness (layers per unit reward), and fairness is quantified using a normalized Jain's fairness index over allocated layers and cumulative rewards across devices. This captures the degree of equitable task and payment distribution in heterogeneous environments.

The communication overhead and total processing time metrics are computed using the following formulas across all selected GPUs:

$$\text{Comm. Overhead} = \frac{T_{\text{total}}^{\text{comm}}}{T_{\text{total}}^{\text{comm}} + T_{\text{total}}^{\text{comp}}},$$

$$T_{\text{total}}^{\text{proc}} = T_{\text{total}}^{\text{comm}} + T_{\text{total}}^{\text{comp}}.$$

Here, $T_{\text{total}}^{\text{comm}}$ is the total inter-device and client communication time, and $T_{\text{total}}^{\text{comp}}$ is the cumulative computation time across selected GPUs. These metrics are averaged over multiple simulation runs with different task sets and seeds.

To quantify FCIM's advantage, we compute the processing time reduction relative to each baseline using:

$$\text{Reduction } (\%) = \left(\frac{T_{\text{strategy}}^{\text{proc}} - T_{\text{FCIM}}^{\text{proc}}}{T_{\text{strategy}}^{\text{proc}}} \right) \times 100.$$

6.2.2 Evaluating FCIM: Efficiency, Fairness, and Adaptability Under Different Scenarioes

As illustrated in Figures 6.1 and 6.2, FCIM efficiently allocates LLM layers by selecting a smaller number of GPUs with higher memory capacity and lower bid costs. This strategy leads to tighter pipeline formation and better resource utilization across both bidding scenarios. In Scenario S1, FCIM prefers RTX 4090 GPUs due to their lower cost-per-layer, especially for larger models like BLOOM and GPT-3. In Scenario S2, where cost dynamics are more aggressive, FCIM adaptively shifts to lower-cost T4 GPUs for smaller models like GPT-Neo, balancing performance and budget constraints.

In contrast, PipeEdge consistently favors high-performance GPUs regardless of cost, leading to suboptimal budget utilization. HexGen exhibits fragmented and imbalanced layer allocations, particularly under aggressive bidding (S2). PipeDream maintains a balanced allocation strategy, but its rigid pipeline design results in excessive inter-GPU communication in both scenarios.

Figure 6.3 shows that FCIM achieves the lowest total cost across all models and bidding configurations by dynamically adapting to bid profiles and model requirements. This cost-efficiency stems from minimizing both the number of engaged GPUs and the cost-per-layer trade-off.

Furthermore, FCIM reduces communication overhead and processing time, as demonstrated in Figures 6.4 and 6.5.

We emphasize that the results for Scenarios S1 and S2 are calculated and presented independently. No aggregation or stacking is used; each scenario's communication overhead and processing time are computed separately using the defined metrics and plotted using distinct bars (dark for S1 and light for S2). While overlapping values may visually appear stacked, the y-axis always starts from zero and no summation occurs.

Although processing time slightly increases in S2 due to reliance on lower-end GPUs, the trade-off remains favorable under budget constraints.

Compared to HexGen, PipeDream, and PipeEdge, FCIM reduces average communication overhead by 18.8%, 34.6%, and 54.7%, respectively. It also achieves average reductions in total processing time of 11.6%, 36.9%, and 27.8%. These improvements reflect FCIM's ability to maintain cost-effectiveness, fairness, and adaptability across varying model sizes, pricing conditions, and deployment scenarios.

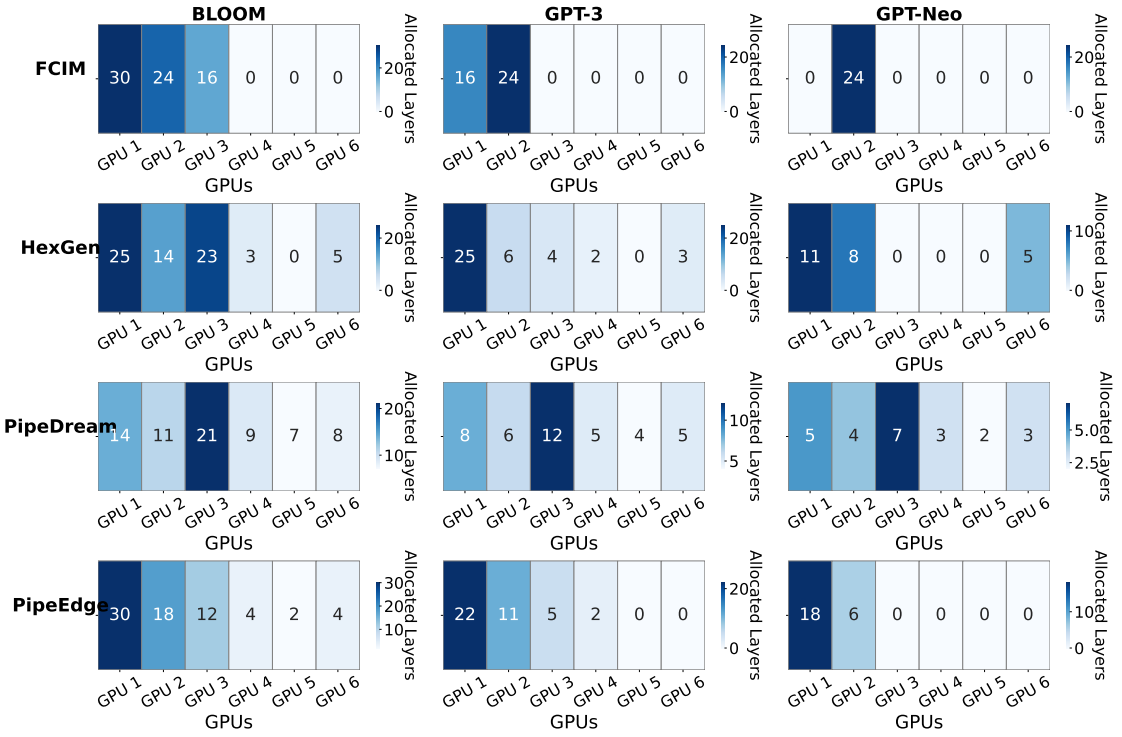

Figure 6.1 Layer allocation heatmap (Scenario S1): RTX 4090 (GPUs 1–3), T4 (GPUs 4–6).

FCIM demonstrates superior fairness performance by achieving the maximum fairness indexes for reward distribution and layer assignments according to Fig.6.6 and Fig.6.7.

The adaptive incentive mechanism of FCIM achieves superior performance in Scenario S1 because it distributes rewards fairly when RTX 4090 GPUs offer bids that are lower than those of T4 GPUs. The uneven distribution of workloads between high-performance GPU devices makes PipeEdge achieve the lowest fairness metric through its biased allocation system. HexGen achieves reasonable execution compared to FCIM since its predetermined allocation methodology offers restricted flexibility.

The adoption of higher bid submissions from RTX 4090 GPUs in S2 scenario leads to moderate deterioration of fairness levels in FCIM. The competitive environment leads GPUs to request more tasks since their reduced numbers accept bidding jobs. FCIM maintains superior fairness functionality when compared to PipeEdge and HexGen because it combines worker selection dynamics with both bidding information and available memory measurements. The system supports balanced layer and reward distribution through these mechanisms regardless of changes in the operating

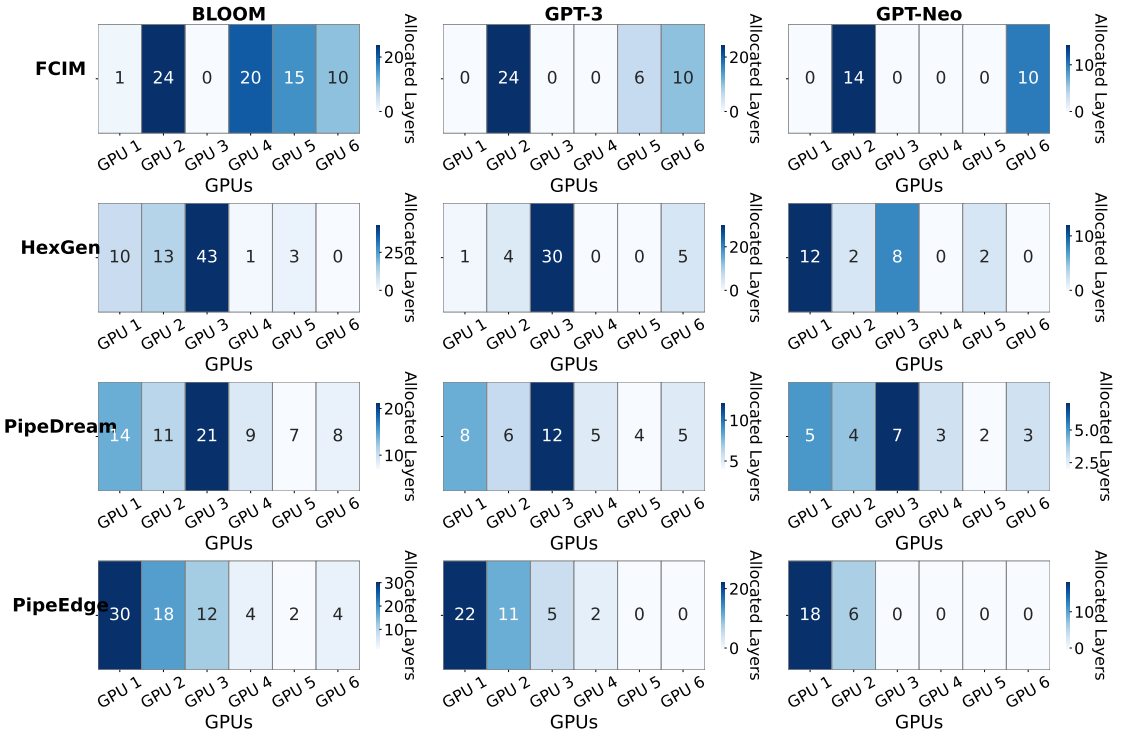

Figure 6.2 Layer allocation heatmap (Scenario S2): RTX 4090 (GPUs 1–3), T4 (GPUs 4–6).

environment.

Memory optimization for GPUs effectively leverages their available resources preventing underutilization as well as overload that may occur on specific devices. Fig. 6.8 demonstrates that FCIM produces the best average memory utilization performance when compared to other examined techniques. The selection process of FCIM chooses GPUs according to available memory capacity but only when their pricing falls within budget constraints. The bottleneck reduction capabilities of FCIM become superior to other benchmarks because it deploys model layers to GPUs which contain sufficient memory resources.

The dynamic task management approach of FCIM selects computations for more powerful GPUs that have extra memory capacity alongside cost considerations in scenario S1. In contrast, PipeDream and PipeEdge struggle with memory efficiency due to their rigid allocation strategies. The pipeline parallelism approach in PipeDream distributes computations between GPUs yet introduces performance problems because GPUs need to exchange data with each other. The GPU selection methods of PipeEdge favor elite hardware choices thus restricting users from exploiting memory resources in lower-end or middle-class GPU models. HexGen maintains

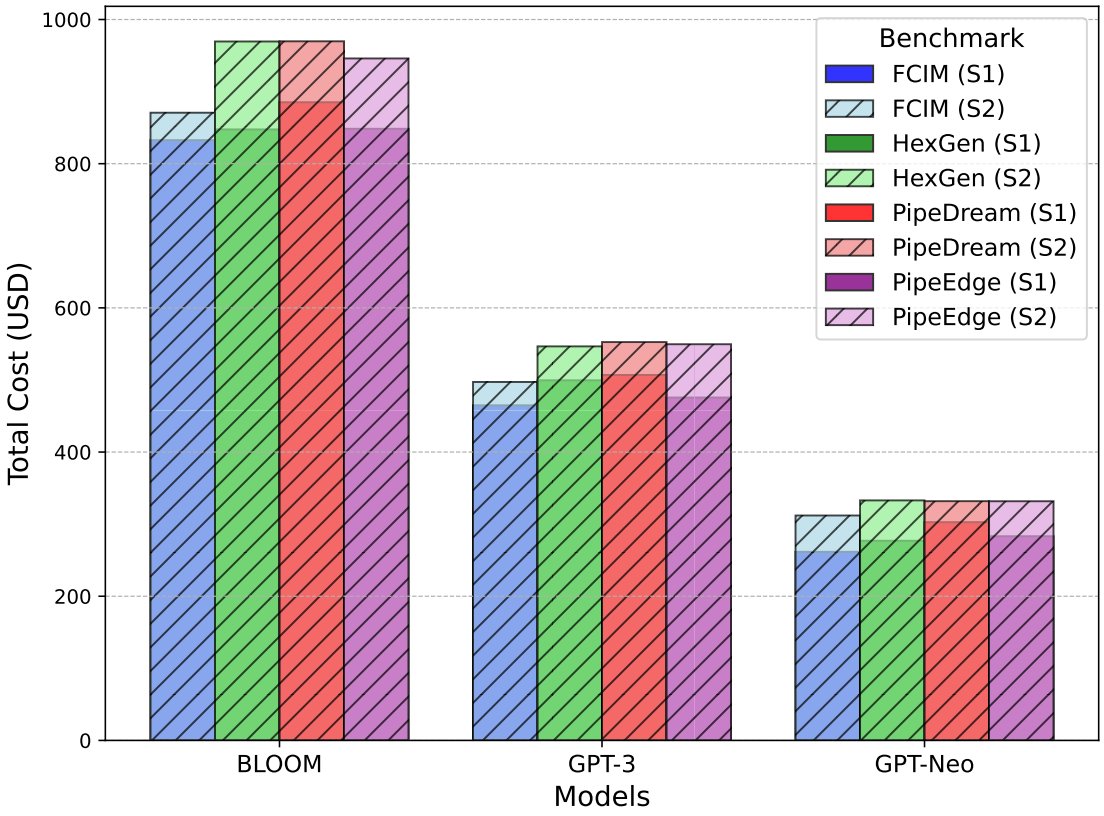

Figure 6.3 Total cost comparison across models and bidding scenarios.

its memory allocation through a pre-defined hierarchical system that stays disconnected from actual GPU environmental changes. When operated as described layer assignments become spread across GPUs thus leading to reduced memory efficiency.

FCIM manages the extended bid format of scenario S2 by maintaining efficient memory consumption and operational costs. PipeDream alongside HexGen distribute memory resources in an inconsistent way thus negatively affecting load processing time. HexGen maintains a fixed strategy that does not respond to market price changes thus resulting in a mismatch of layer assignments where GPU resources are either overloaded or underutilized. The method proves unproductive since it fails to optimize the use of memory resources alongside compute resources. FCIM offers the optimal solution for cost efficiency and memory optimization and performance by selecting GPUs according to their combined memory capacity and cost needs.

The cost efficiency measurement expresses how well computing system funds are utilized based on the layer allocation against total expenses. Figure 6.9 shows that FCIM demonstrates superior cost performance than other frameworks in Scenario S1 because it uses its cost-aware selection approach. S1 demonstrates the maximum cost efficiency through FCIM which selects high-memory RTX 4090 GPUs because their pricing level is the lowest. FCIM benefits from these powerful GPUs with

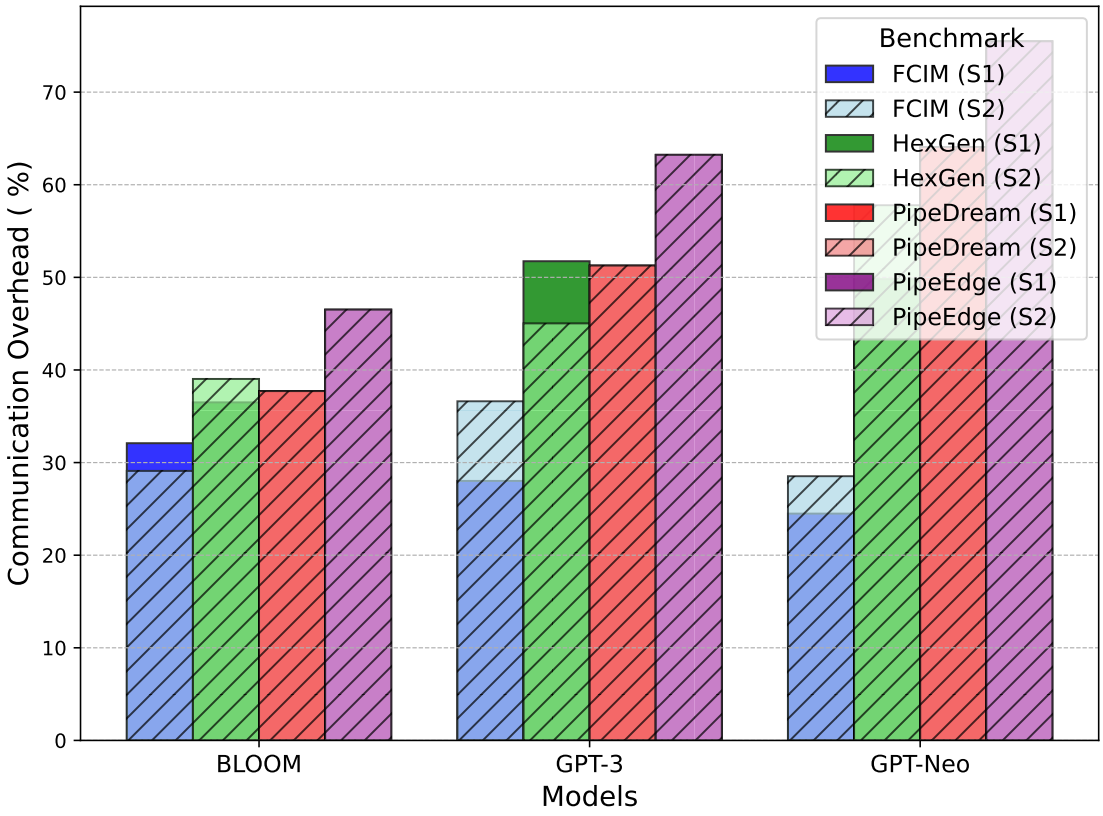

Figure 6.4 Communication overhead comparison across benchmarks.

low price-per-layer offers through layer assignment so it decreases total expenses along with optimizing workload distribution. The inability of PipeEdge to adapt GPU pricing makes it less competitive than the FCIM framework. The current implementation makes its cost determination dependent on fixed expenses which generates higher expenses per layer than FCIM.

The layer distribution policies of HexGen and PipeDream fail to minimize costs therefore leading to their inferior performance. A dynamic bidding system must be utilized to maximize the selection of affordable GPUs since non-dynamic approaches create execution costs that rise higher than desirable levels.

The cost efficiency of FCIM declines a little in Scenario S2 for BLOOM and GPT-3 because of structural changes in the bidding system. The cost per layer rises for GPT-Neo because its GPU selection is reduced compared to other frameworks. The cost efficiency ratio of FCIM remains superior to other benchmarks despite the evaluation changes.

We conduct an empirical energy analysis of the four distributed inference strategies by measuring real-time GPU power draw using `nvidia-smi` with a 1-second sampling interval. The total energy consumption E is computed as:

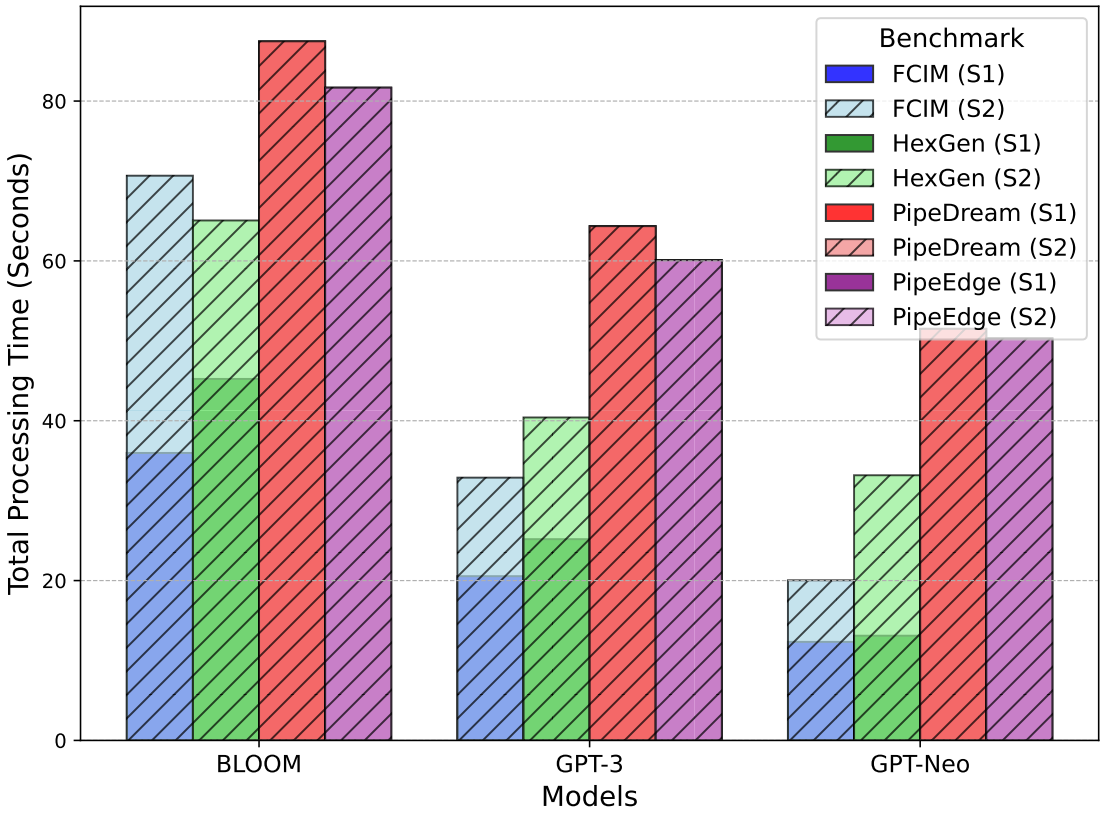

Figure 6.5 Total processing time comparison across benchmarks.

$$E = \int_0^T P(t)\,\mathrm{d}t \approx \sum_{i=1}^{N} P(t_i) \cdot \Delta t, \tag{6.2}$$

where $P(t_i)$ is the instantaneous power draw at timestamp t_i, $\Delta t = 1\,\mathrm{s}$ is the sampling interval, and T is the total runtime.

The observed energy consumption differences between Scenarios S1 and S2 in Figure 6.10 directly reflect the impact of device heterogeneity, bidding configurations, and layer allocation strategies. Scenario S1 follows Equation (33) from Appendix$\tilde{\text{E}}$, where the price per layer is inversely related to GPU memory and compute capacity. In this scenario, RTX 4090 GPUs propose lower prices due to their higher resources, making them favorable for cost-optimized selection. As a result, FCIM and PipeEdge allocated more layers to the high-performance RTXs in S1, yielding higher total energy usage for BLOOM (15.76 kJ for FCIM and 14.94 kJ for PipeEdge). However, PipeDream, which distributes layers more evenly across all devices, achieved lower energy at 12.97 kJ. HexGen, which assigned fewer layers to RTXs, resulted in the lowest among them (12.26 kJ). In contrast, Scenario S2 revised pricing to reflect realistic power-performance trade-offs, making T4 GPUs more attractive due to their lower consumption. This shift allowed FCIM to reduce

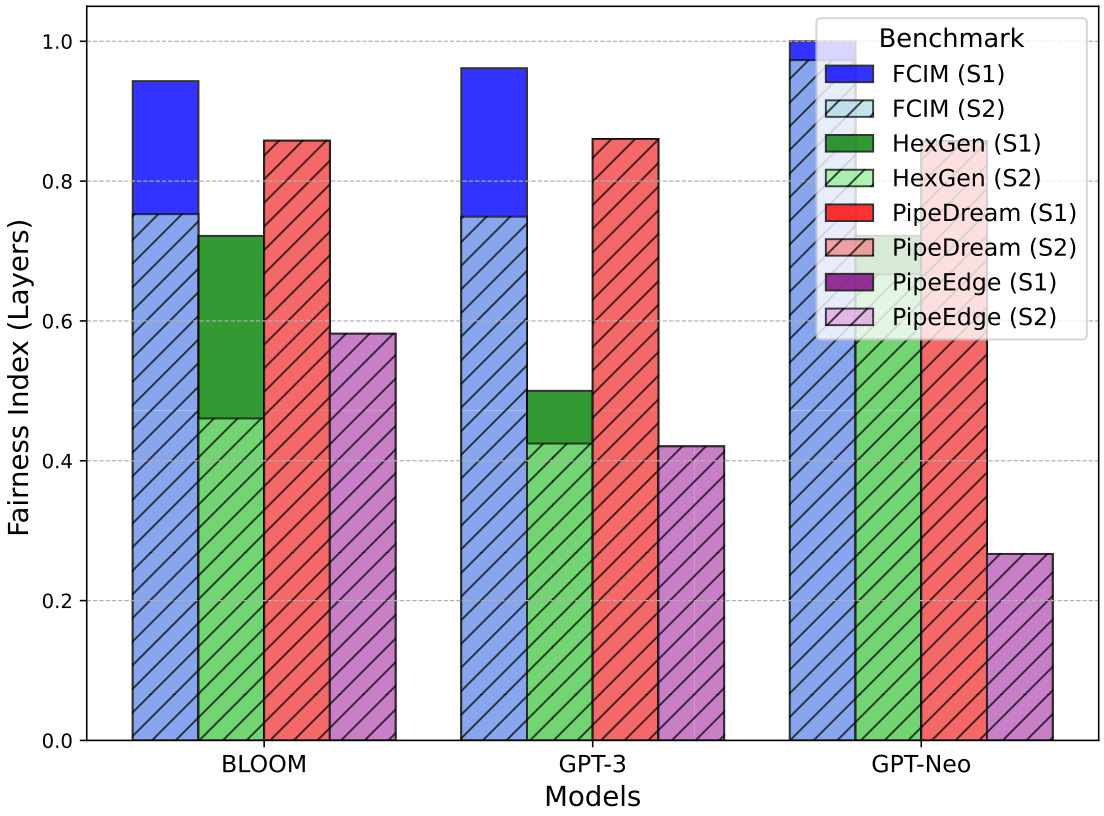

Figure 6.6 Fairness index layers comparison for benchmarks across different models and bidding scenarios.

energy to 6.46 kJ and HexGen to 8.2 kJ by leveraging more T4s, while PipeDream and PipeEdge remained unchanged due to their fixed strategies. Overall, FCIM demonstrates key advantages across multiple metrics. It achieves the best fairness in both layer and reward distribution, superior cost-efficiency (0.0841 in S1), and the lowest communication overhead (24.49%) and processing time (12.33s) for BLOOM in S1. In S2, FCIM continues to outperform by adapting to the new pricing, maintaining balance and reducing energy and total cost, underscoring its flexibility and optimization potential in dynamic, heterogeneous environments.

To better understand the influence of the penalty parameter δ, we conduct a sensitivity analysis under Scenario S2 with fixed weight values $\alpha = 0.4$, $\beta = 0.3$, and $\gamma = 0.3$, as shown in Fig. 6.11.The results indicate that as δ increases, the total cost of executing the task consistently declines. This trend reflects the model's increasing willingness to penalize overpriced bids more aggressively. The underlying mechanism is defined in (23), where the penalty grows proportionally with δ, reducing the final reward allocated to a winning GPU when its unit cost exceeds the average of its competitors.

However, this cost reduction comes at a trade-off: fairness in terms of reward dis-

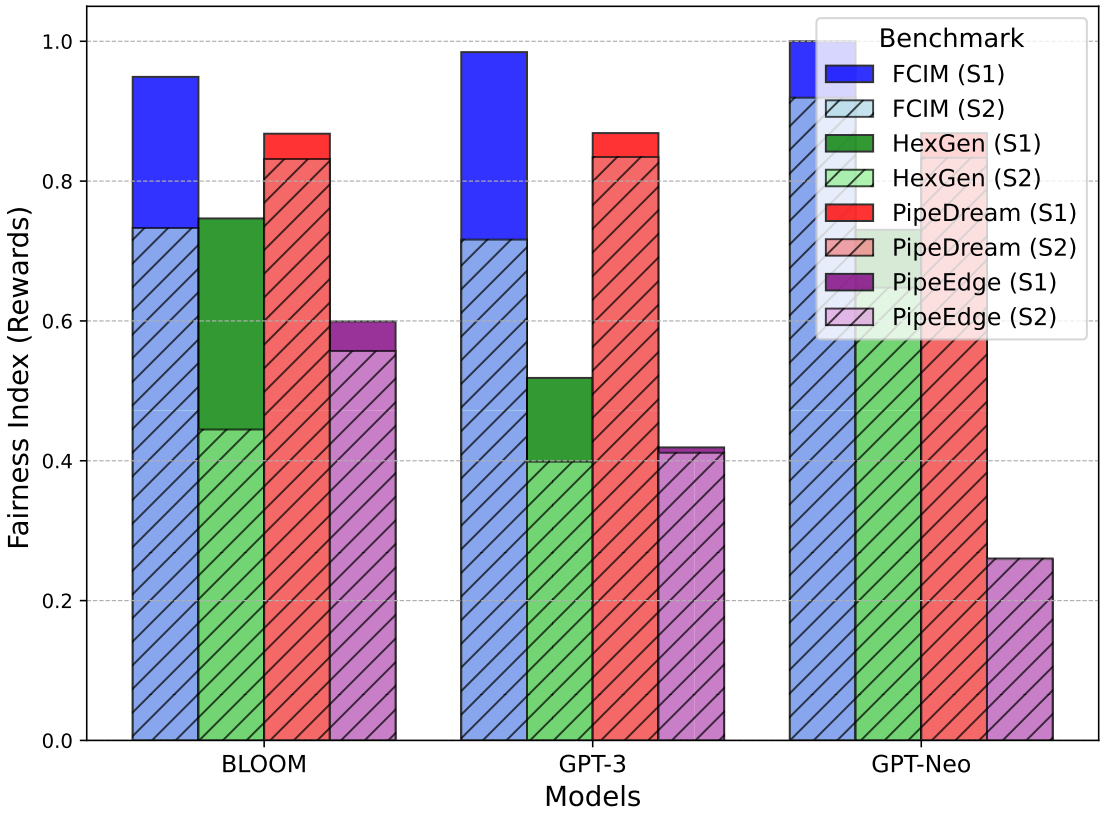

Figure 6.7 Fairness index rewards comparison for benchmarks across different models and bidding scenarios.

tribution begins to degrade significantly once δ exceeds 3. In contrast, fairness with respect to layer allocation and the total processing time remains largely unaffected. This is because changes in δ only influence the final reward calculation—not the selection of winner GPUs or the number of layers assigned to them. Furthermore, after $\delta = 13$, both cost and time metrics plateau. This behavior occurs because the penalty reaches its maximum allowable limit, known as R_{safe}, beyond which further increases in δ no longer affect the outcome.

Overall, this analysis highlights the importance of tuning δ carefully. While higher values can improve cost efficiency, they may also compromise fairness—underscoring the need to balance these competing objectives in distributed GPU auction-based scheduling.

To provide insights into parameter tuning, we conducted a sensitivity analysis on the scoring weights α, β, and γ under Scenario S2 using the GPT-3 model. These weights respectively influence the importance of task-to-layer efficiency, computation time, and cost in the GPU selection mechanism. We evaluate their impact on four key performance metrics: total cost, fairness in layer distribution, fairness in reward distribution, and total processing time.

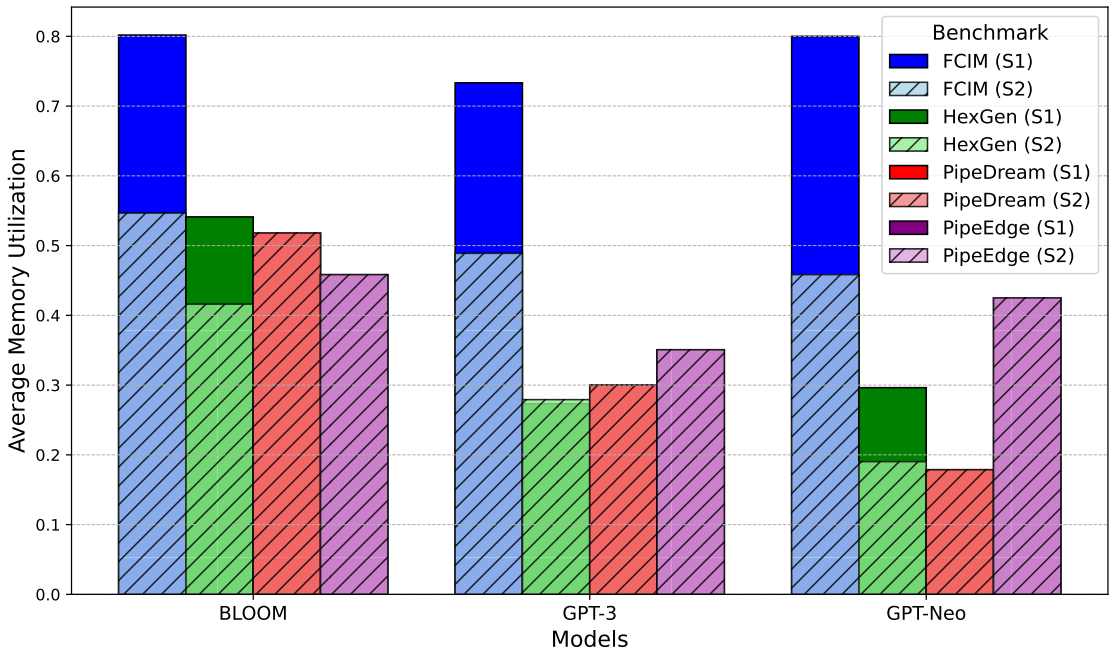

Figure 6.8 Average memory utilization comparison for benchmarks across different models and bidding scenarios.

- **Total Cost (USD):** As shown in Fig. 6.12, the total cost increases when γ is small and α/β is low, indicating an overemphasis on speed. In contrast, higher α/β values prioritize layer-efficient GPUs, reducing the cost. The cost is relatively stable for moderate γ (e.g., 0.7) but increases again for extreme values.

- **Fairness Index (Layers):** Fig. 6.13 shows that fairness in layer allocation remains high (close to 1) across most configurations. However, slight decreases are observed for $\gamma = 0.6$ and 0.51 when α/β is low, suggesting that excessive focus on speed or cost can skew the distribution.

- **Fairness Index (Rewards):** Fig. 6.14 indicates that reward fairness improves as α/β increases, especially under lower γ values. This implies that scoring strategies that emphasize task efficiency promote more balanced compensation across GPUs.

- **Total Processing Time:** As illustrated in Fig. 6.15, total processing time drops with increasing α/β, particularly when γ is small. This suggests that emphasizing layer efficiency results in quicker task completion, while cost-prioritized selections can lead to higher latency.

This analysis provides guidance for tuning α, β, and γ depending on the system's optimization goal—whether to minimize cost, balance load, ensure fairness, or reduce

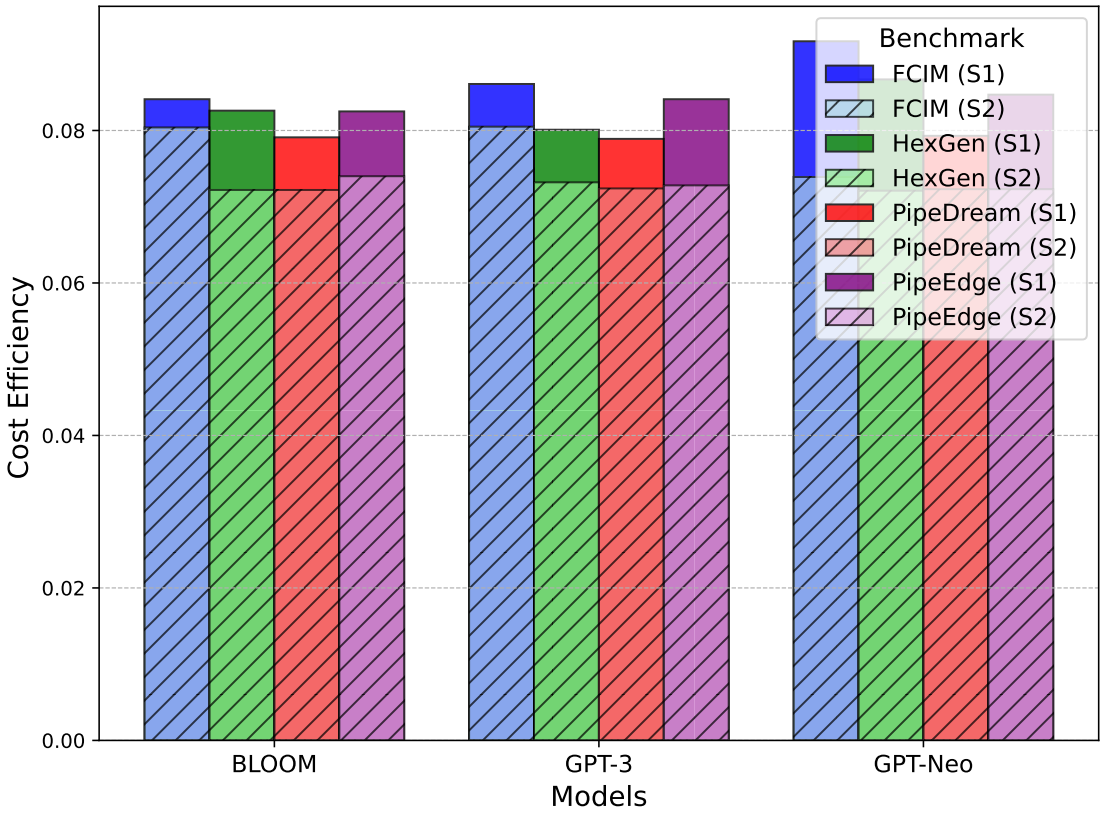

Figure 6.9 Cost efficiency comparison for benchmarks across different models and bidding scenarios.

latency.

6.3 Effectiveness of ADSA in Scheduling Tasks

This subsection evaluates the performance of ADSA against standard scheduling baselines, including Least Laxity First (LLF) Arvindhan, Reetu & Kajol (2018); Cheng, Zhang, Tan & Lim (2017); Yoon et al. (2023), Shortest Remaining Time First (SRTF) González-Rodríguez et al. (2024); Zhao, Li, Qu, Zhan & Zhang (2021), and First-Come, First-Served (FCFS) Kaur & Saini (2017). LLF prioritizes tasks with minimal slack time to satisfy real-time constraints but incurs high preemption overhead. SRTF minimizes average turnaround time by prioritizing shorter tasks, yet often results in starvation for longer ones. FCFS ensures fairness in task order but suffers from the convoy effect and inefficient resource usage.

To establish an upper bound, an optimal scheduler is implemented using the PULP-

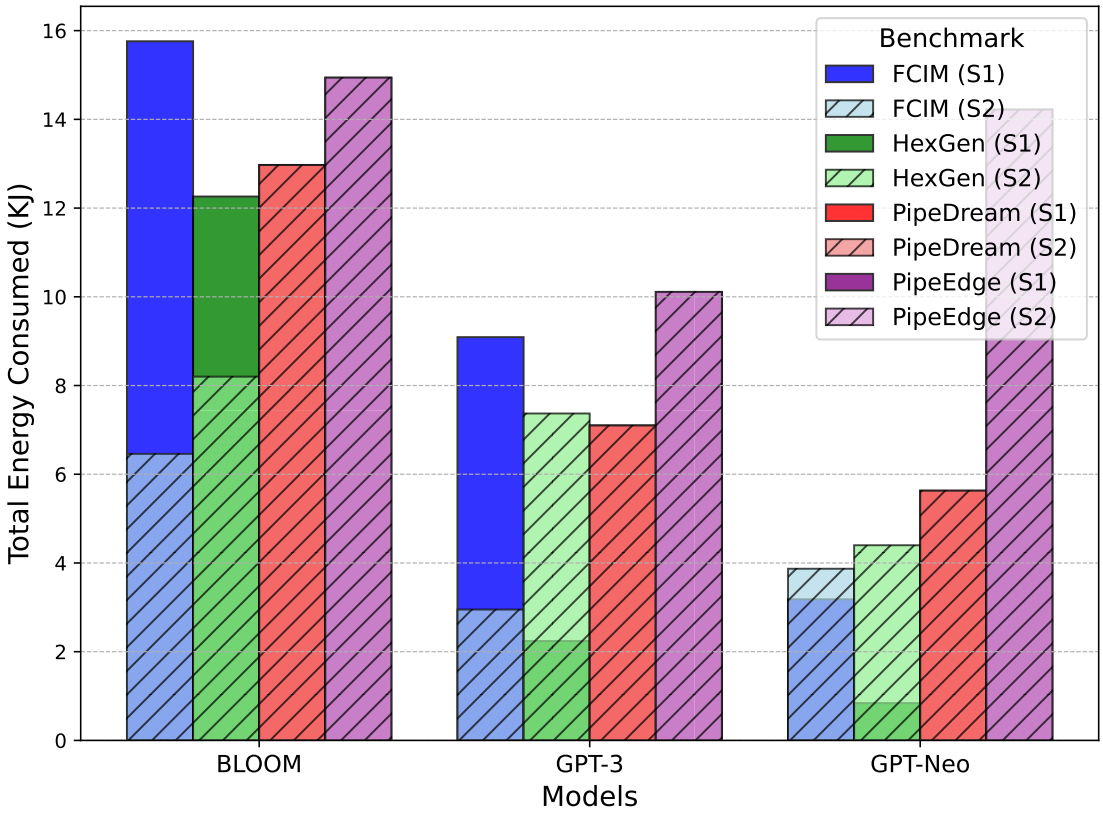

Figure 6.10 Total Energy Consumed comparison for benchmarks across different models and bidding scenarios.

CBC solver, where tasks are treated as atomic units, and constraints on deadline, execution time, and post-computation transmission are explicitly enforced. This allows comparison against a theoretically ideal, deadline-respecting scheduler under simplified conditions.

6.3.1 Experimental Setup and Evaluation Framework for ADSA

To evaluate the Adaptive Dynamic Scheduling Algorithm (ADSA) researchers used one single device which operated as a real-time scheduling prototype. GPT-Neo 1.3B served as the workload generator which executed its tasks on an RTX 4090 graphics processing unit. The setup enabled the creation of dynamic task loads to test the scheduler's capacity to handle limited resources effectively.

- Computation Workloads: Diverse computational requirements emerged from the tasks which had input prompts of different sizes and output length specifications.

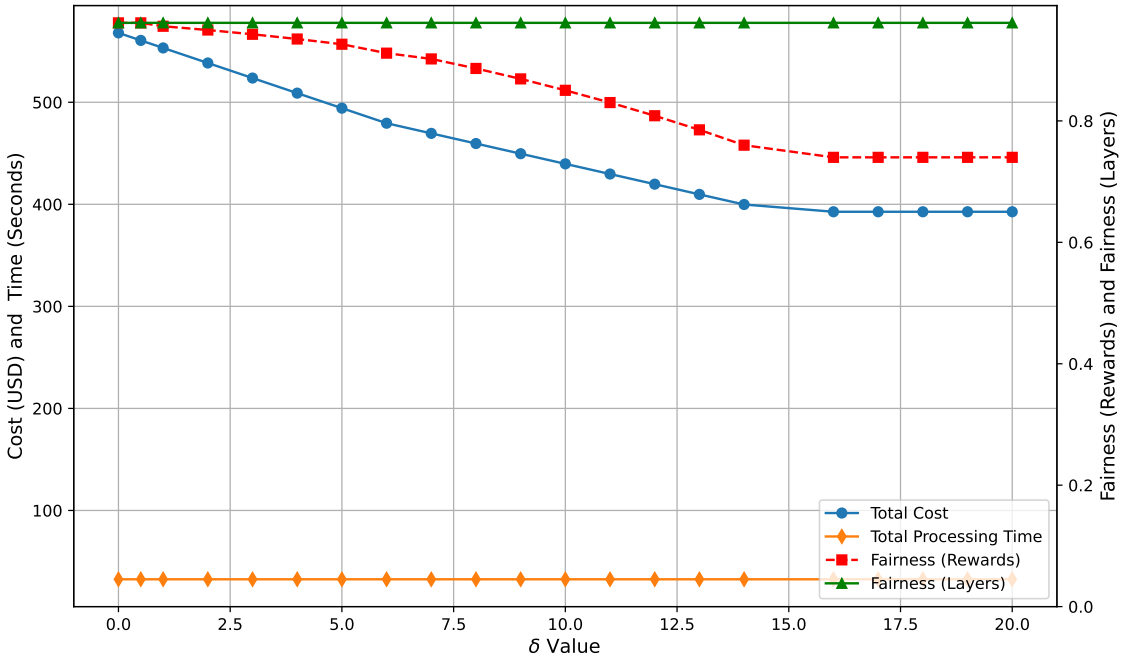

Figure 6.11 Sensitivity of cost, time, and fairness metrics to δ in Scenario S2 with $\alpha = 0.4$, $\beta = 0.3$, and $\gamma = 0.3$.

- Dynamic Arrival Times and Deadlines: The time schedule for task arrivals included different waiting times between tasks in addition to unique deadline requirements which simulated actual service operations.
- Client-Specified Output Requirements: Selection of custom output sizes for each job increased the expansive range of scheduling responsibilities.

The test focused on how ADSA operated against subproblem2 for assessing single-device management of real-time heterogeneous tasks. The experimental setup reveals ADSA's capabilities regarding constrained resource scheduling by effectively handling time-sensitive tasks measurement alongside load management and deadline performance.

Task Definition and Simulation Workflow The system contained five tasks that followed these specifications:

- Prompt Complexity: The output lengths of the tasks varied between 200 words and 2200 words for prompt complexity.
- Deadline Constraints: Each task carried set deadline times that determined its arrival time requirements.
- Transmission Overheads: Showing transmission delays through simulated network delays composed a major part of the simulation.

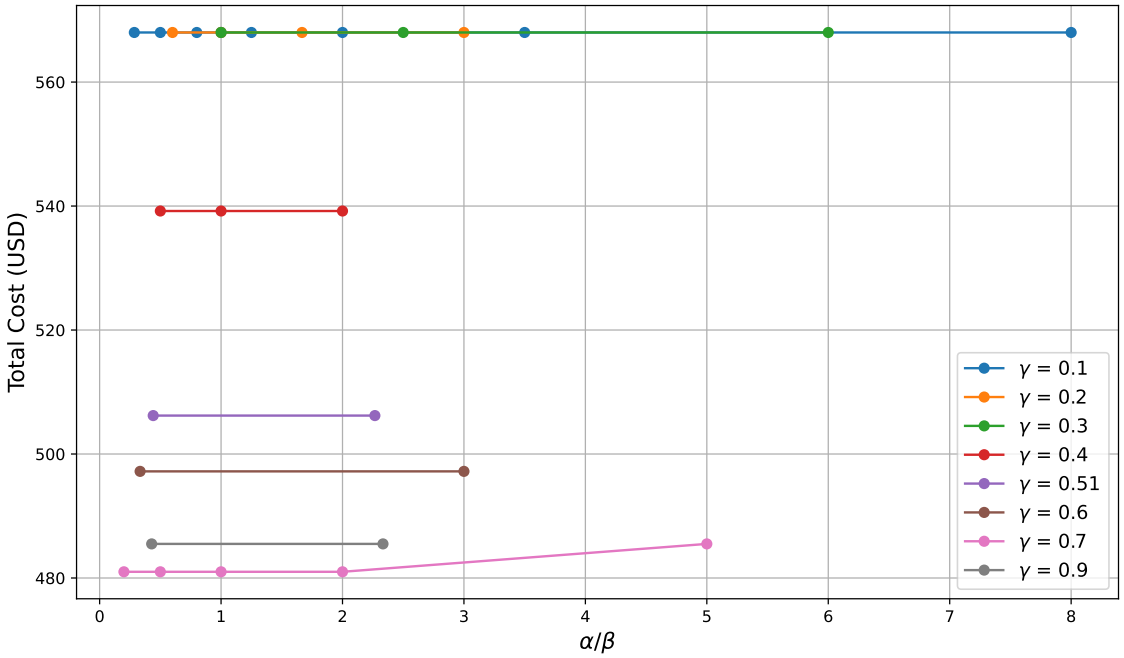

Figure 6.12 Sensitivity analysis of Total Cost vs α/β at fixed values of γ

Under the ADSA algorithm the task queue received automatic reordering which considered deadlines and transmission times together with processing priorities. The GPU received input tokens which enabled task scheduling that included potential preemptive and resumptive activities. Evaluating resource usage together with efficiency required the tracking of performance metrics including computation time and deadline adherence and memory usage and waiting time.

6.3.2 Comparative Analysis of ADSA and Baseline Scheduling Methods

Figures 6.16–6.18 provide a comprehensive evaluation of ADSA compared to four baseline schedulers: FCFS, LLF, SRTF, and an Optimal solver-based schedule.

Figure 6.16 illustrates the task execution timelines for each scheduling method. ADSA maintains compact, non-overlapping schedules with minimal idle time and no deadline violations. In contrast, FCFS and LLF introduce idle gaps and frequent preemptions, respectively. SRTF improves task turnaround but causes starvation for longer tasks—Task 4, for instance, misses its deadline. The Optimal schedule, while meeting all deadlines, assumes idealized conditions not easily reproducible in dynamic systems.

Figure 6.17 shows the deadline margins (i.e., deadline minus transmission comple-

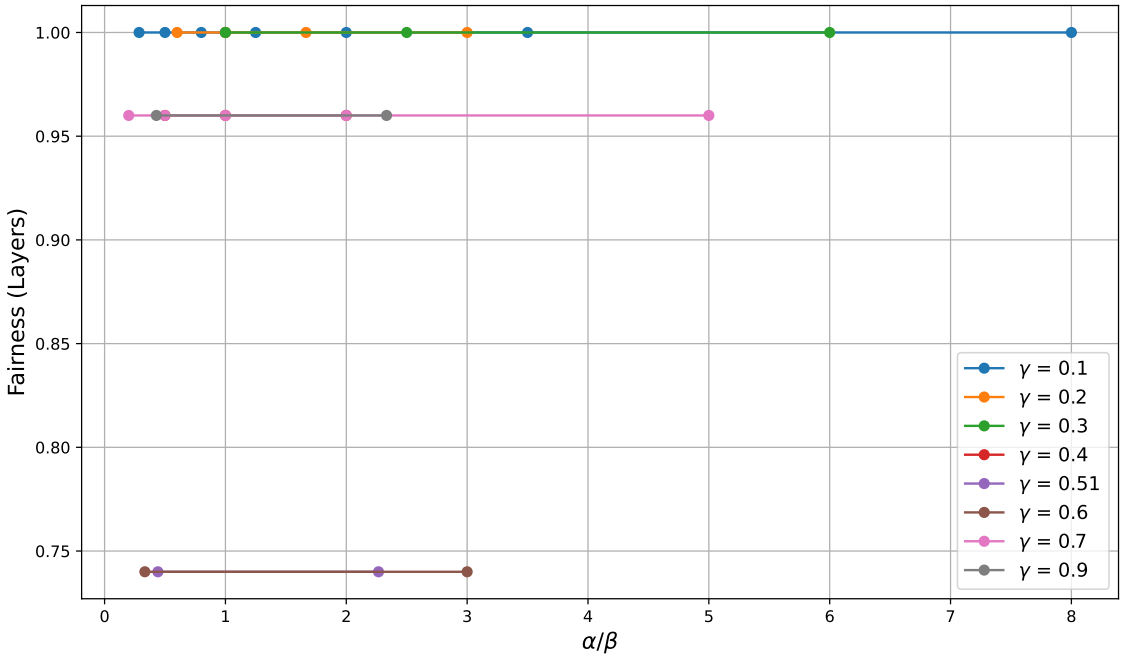

Figure 6.13 Sensitivity analysis of Fairness (Layers) vs α/β at fixed values of γ

tion) for each task under different scheduling policies. ADSA consistently ensures non-negative margins, confirming deadline compliance across all tasks. SRTF and FCFS exhibit deadline violations in several cases, most notably for Task 4 and Task 5, reinforcing their unreliability under time-constrained workloads.

Figure 6.18 compares average waiting times. While SRTF yields slightly lower queue times, it does so at the cost of fairness and deadline violations. ADSA strikes a better balance—reducing waiting time by 39.57% and 38.59% compared to FCFS and LLF, respectively—while preserving deadline satisfaction.

ADSA's design—combining deadline awareness, low preemption overhead, and adaptive prioritization—enables it to maintain both efficiency and robustness under dynamic and heterogeneous scheduling conditions.

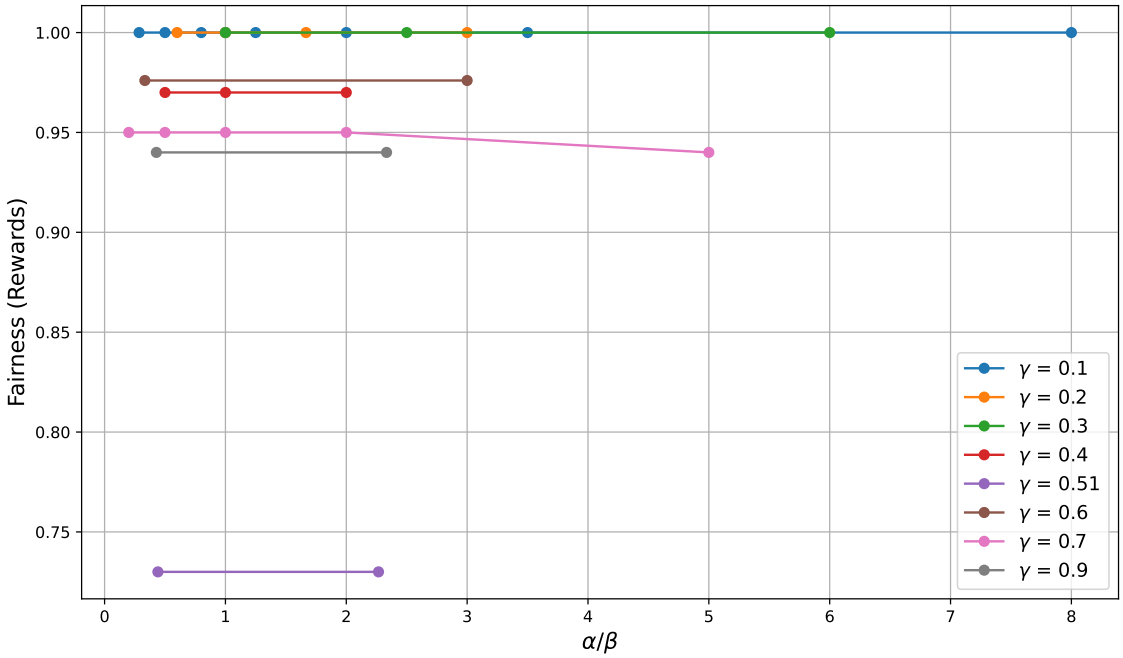

Figure 6.14 Sensitivity analysis of Fairness (Rewards) vs α/β at fixed values of γ

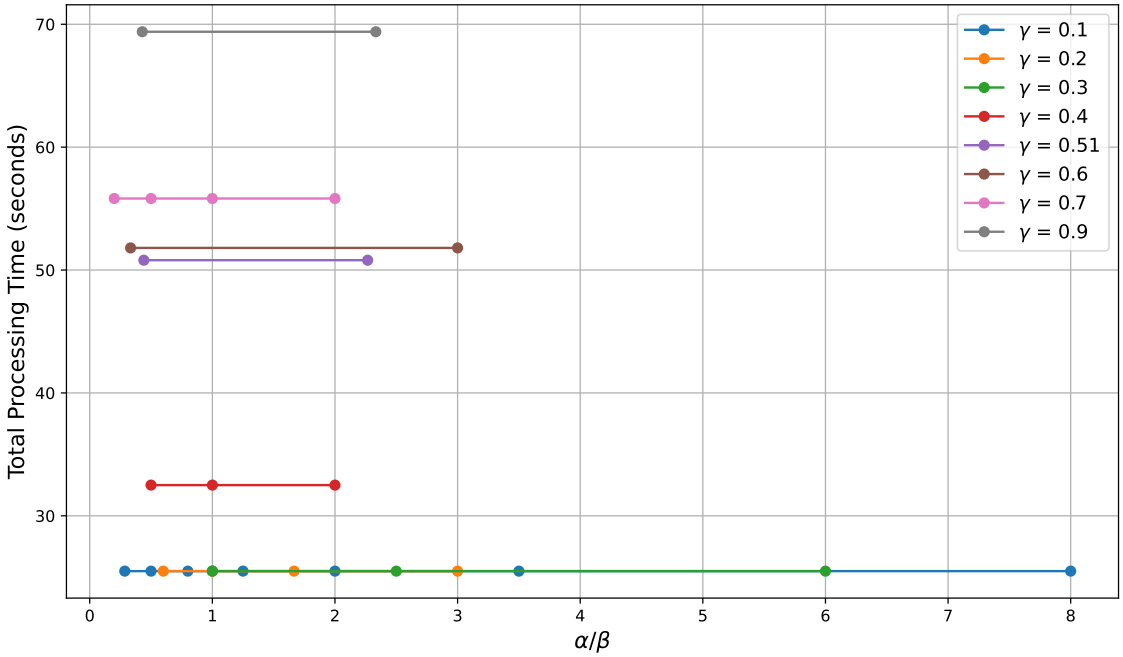

Figure 6.15 Sensitivity analysis of Total Processing Time vs α/β at fixed values of γ

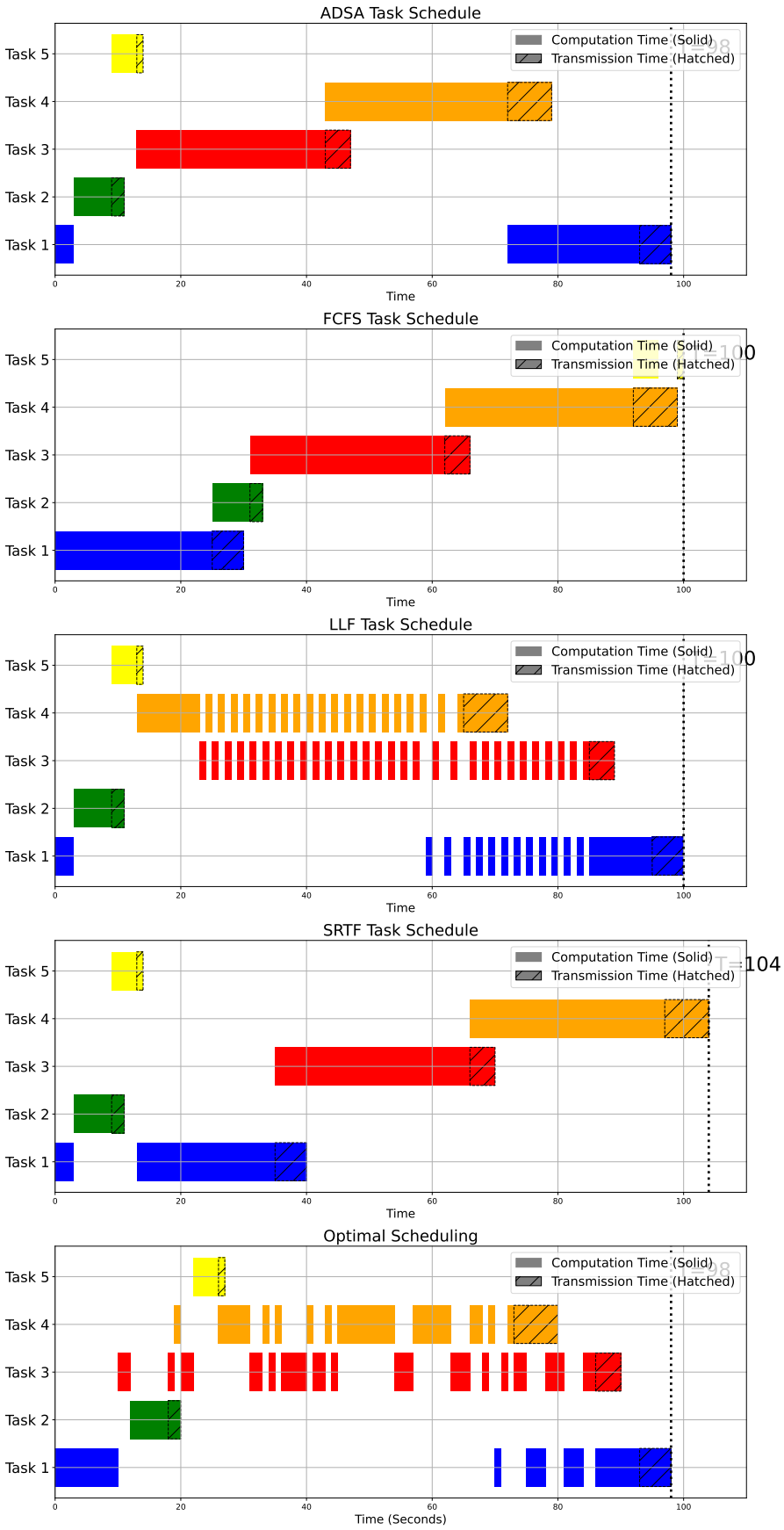

Figure 6.16 Comparison of Task Schedules for ADSA, FCFS, SRTF, LLF , and Optimal Scheduling Methods.

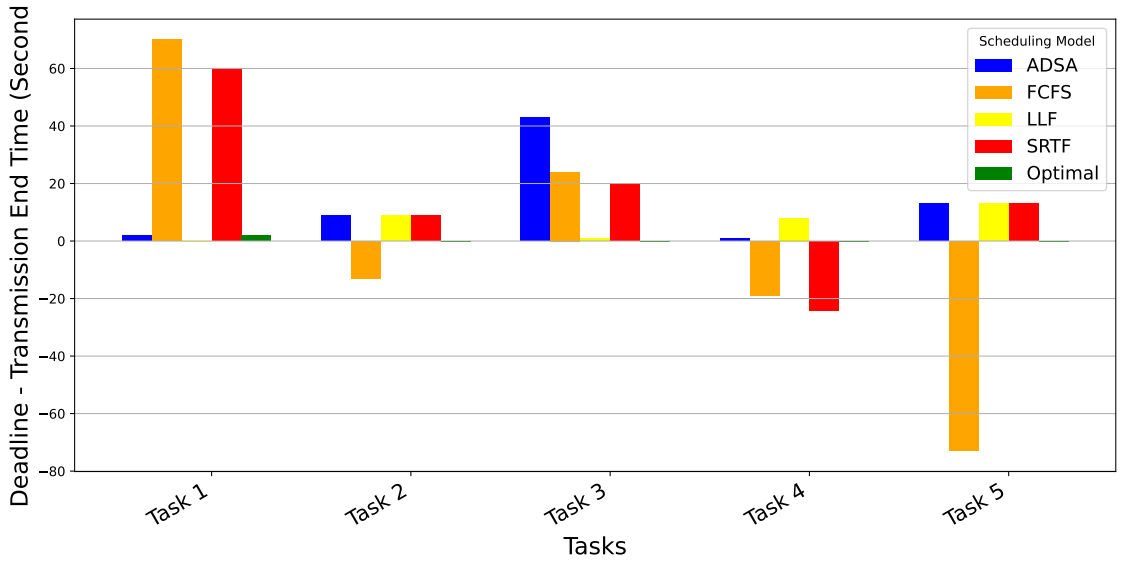

Figure 6.17 Comparison of Deadline Margins for ADSA, FCFS, LLF, SRTF, and Optimal Scheduling Methods

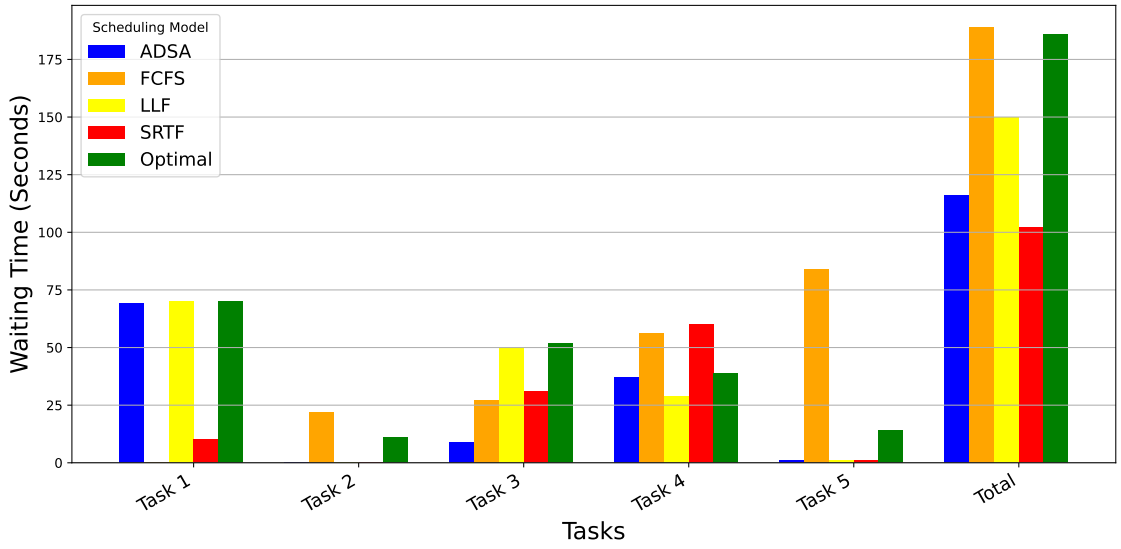

Figure 6.18 Comparison of waiting time in queue for ADSA, FCFS, LLF, SRTF and Optimal scheduling methods through all sections of model.

7. CONCLUSION

This thesis addresses two core challenges in edge-based LLM inference: (1) designing a cost-efficient, fairness-aware incentive mechanism for task assignment, and (2) developing a deadline-aware, low-overhead scheduling strategy for dynamic execution environments.

To this end, we introduced the Fair Cost-Efficient Incentive Mechanism (FCIM), which adaptively allocates model layers based on device availability, memory constraints, and bidding costs. FCIM minimizes a weighted objective that balances task completion time, total reward cost, and the number of devices used. It significantly reduces communication overhead and achieves up to 54.7% improvement in total cost compared to state-of-the-art methods such as HexGen, PipeDream, and PipeEdge.

We also proposed the Adaptive Dynamic Scheduling Algorithm (ADSA), which combines deadline sensitivity with low preemption and efficient queue management. ADSA outperforms classical schedulers like FCFS and LLF by reducing waiting time by up to 39.6% without violating task deadlines—a weakness observed in SRTF.

Overall, the proposed methods demonstrate reliable, flexible, and scalable performance across diverse models, hardware configurations, and bidding scenarios. Future work may extend these frameworks to multi-tenant environments and consider reinforcement learning-based adaptive bidding strategies for more dynamic edge participation.

In future research, we aim to enhance the security of the pipeline-based inference process by establishing secure communication protocols between selected devices. Additionally, future extensions may explore reinforcement learning-based adaptive bidding strategies, privacy-preserving task allocation, and support for multi-tenant edge environments with isolated resource guarantees.

BIBLIOGRAPHY

Akbari, M., Rashidi, H., & Alizadeh, S. H. (2017). An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems. *Engineering Applications of Artificial Intelligence*, *61*, 35–46.

Ali, H. H. & El-Rewini, H. (1993). The time complexity of scheduling interval orders with communication is polynomial. *Parallel Processing Letters*, *3*(01), 53–58.

Arvindhan, M., Reetu, S., & Kajol, K. (2018). An efficient technique for scheduling algorithm in real time environment for optimizing the operating system. *International Journal of Computer Science & Wireless Security*, *2*(1), 276–278.

Bakhtiarnia, A., Milošević, N., Zhang, Q., Bajović, D., & Iosifidis, A. (2023). Dynamic split computing for efficient deep edge intelligence. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 1–5). IEEE.

Cheng, Z., Zhang, H., Tan, Y., & Lim, Y. (2017). Smt-based scheduling for overloaded real-time systems. *IEICE TRANSACTIONS on Information and Systems*, *100*(5), 1055–1066.

Deng, S., Zhao, H., Fang, W., Yin, J., Dustdar, S., & Zomaya, A. Y. (2020). Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, *7*(8), 7457–7469.

Eisenbrand, F. & Rothvoß, T. (2008). Static-priority real-time scheduling: Response time computation is np-hard. In *2008 Real-Time Systems Symposium*, (pp. 397–406). IEEE.

Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.

González-Rodríguez, M., Otero-Cerdeira, L., González-Rufino, E., & Rodríguez-Martínez, F. J. (2024). Study and evaluation of cpu scheduling algorithms. *Heliyon*, *10*(9).

He, Y., Fang, J., Yu, F. R., & Leung, V. C. (2024). Large language models (llms) inference offloading and resource allocation in cloud-edge computing: An active inference approach. *IEEE Transactions on Mobile Computing*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, *1*(2), 3.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, *32*.

Huynh, L. N., Lee, Y., & Balan, R. K. (2017). Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, (pp. 82–95).

Iwata, S. & Nagano, K. (2009). Submodular function minimization under covering constraints. In *2009 50th annual IEEE symposium on foundations of computer science*, (pp. 671–680). IEEE.

Jaiswal, S., Jain, K., Simmhan, Y., Parayil, A., Mallick, A., Wang, R., Amant,

R. S., Bansal, C., Rühle, V., Kulkarni, A., et al. (2025). Serving models, fast and slow: optimizing heterogeneous llm inferencing workloads at scale. *arXiv preprint arXiv:2502.14617*.

Jiang, Y., Yan, R., Yao, X., Zhou, Y., Chen, B., & Yuan, B. (2023). Hexgen: Generative inference of large language model over heterogeneous environment. *arXiv preprint arXiv:2311.11514*.

Kaur, S. & Saini, P. (2017). Deadline-aware mapreduce scheduling with selective speculative execution. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, (pp. 1–5). IEEE.

Liu, L., Ge, H., Li, S., Chen, X., Gong, H., & Cui, Y. (2021). Resource allocation strategy based on improved auction algorithm in mobile edge computing environment. In *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, (pp. 355–363). IEEE.

Mao, J., Chen, X., Nixon, K. W., Krieger, C., & Chen, Y. (2017). Modnn: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, (pp. 1396–1401). IEEE.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., & Zaharia, M. (2019). Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, (pp. 1–15).

Norman, M. G., Pelagatti, S., & Thanisch, P. (1995). On the complexity of scheduling with communication delay and contention. *Parallel Processing Letters*, *5*(03), 331–341.

Oh, H., Kim, K., Kim, J., Kim, S., Lee, J., Chang, D.-s., & Seo, J. (2024). Exegpt: Constraint-aware resource scheduling for llm inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, (pp. 369–384).

Page, A. J. & Naughton, T. J. (2005). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *19th IEEE international parallel and distributed processing symposium*, (pp. 8–pp). IEEE.

Pang, J., Chou, F.-D., & Tsai, Y.-C. (2017). Minimizing the makespan on parallel machines with flexible maintenances and dynamic release times of jobs. *DEStech Transactions on Social Science, Education and Human Science*.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network training.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, *3*(5), 637–646.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, (pp. 5998–6008).

Wang, B., Zhang, Z., Gholami, A., et al. (2022). On-device inference with trans-

formers: Optimization and scheduling. *NeurIPS*, *35*, 23202–23215.

Wang, Q., Chen, S., & Wu, M. (2023). Incentive-aware blockchain-assisted intelligent edge caching and computation offloading for iot. *Engineering*, *31*, 127–138.

Wang, X., Wu, D., Wang, X., Zeng, R., Ma, L., & Yu, R. (2023). Truthful auction-based resource allocation mechanisms with flexible task offloading in mobile edge computing. *IEEE Transactions on Mobile Computing*, *23*(5), 6377–6391.

Xiao, J., Gao, Q., Yang, Z., Cao, Y., Wang, H., & Feng, Z. (2023a). Multi-round auction-based resource allocation for edge computing: Maximizing social welfare. *Future Generation Computer Systems*, *140*, 365–375.

Xiao, J., Gao, Q., Yang, Z., Cao, Y., Wang, H., & Feng, Z. (2023b). Multi-round auction-based resource allocation for edge computing: Maximizing social welfare. *Future Generation Computer Systems*, *140*, 365–375.

Xu, L., Ge, M., & Wu, W. (2020). Edge computing based incentivizing mechanism for mobile blockchain in iot. *arXiv preprint arXiv:2006.08915*.

Xu, L., Yang, Y., Yuan, B., & Li, Y. (2021). Accelerating bert inference on gpus via architecture-aware layer optimization. *IEEE Transactions on Parallel and Distributed Systems*, *33*(5), 1013–1025.

Xu, R., Nagothu, D., & Chen, Y. (2024). Ar-edge: Autonomous and resilient edge computing architecture for smart cities. In *Edge Computing Architecture-Architecture and Applications for Smart Cities*. IntechOpen.

Yoon, D.-H., Seo, H., Lee, J., & Kim, Y. (2023). Online electric vehicle charging strategy in residential areas with limited power supply. *IEEE Transactions on Smart Grid*, *15*(3), 3141–3151.

Yuan, L., He, Q., Chen, F., Dou, R., Jin, H., & Yang, Y. (2023). Pipeedge: A trusted pipelining collaborative edge training based on blockchain. In *Proceedings of the ACM Web Conference 2023*, (pp. 3033–3043).

Zhao, C. & Tang, H. (2016). Scheduling deteriorating jobs with availability constraints to minimize the makespan. *Asia-Pacific Journal of Operational Research*, *33*(06), 1650048.

Zhao, L., Li, F., Qu, W., Zhan, K., & Zhang, Q. (2021). Aiturbo: Unified compute allocation for partial predictable training in commodity clusters. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, (pp. 133–145).

Zuo, S. (2020). tlpipe: Data processing pipeline for the tianlai experiment. *Astrophysics Source Code Library*, ascl–2011.