Equinox: Holistic Fair Scheduling in Serving Large Language Models

Zhixiang Wei
Shanghai Jiao Tong University
China
tonywei_sjtu@sjtu.edu.cn

James Yen
Shanghai Jiao Tong University
China
jamesyen2202002@gmail.com

Jingyi Chen
Shanghai Jiao Tong University
China
2744234012@qq.com

Ziyang Zhang
Shanghai Jiao Tong University
China
functioner@sjtu.edu.cn

Zhibai Huang
Shanghai Jiao Tong University
China
paynqueller@sjtu.edu.cn

Chen Chen
Shanghai Jiao Tong University
China
chenchen825@sjtu.edu.cn

Xingzi Yu
Shanghai Jiao Tong University
China
edittriendl@sjtu.edu.cn

Yicheng Gu
Shanghai Jiao Tong University
China
guyicheng98@sjtu.edu.cn

Chenggang Wu
Shanghai Jiao Tong University
China
wuchenggang@sjtu.edu.cn

Yun Wang
Shanghai Jiao Tong University
China
yunwang94@sjtu.edu.cn

Mingyuan Xia
UltraRISC Shanghai
China
xiamy@ultrarisc.com

Jie Wu
Cloud Computing Research Institute,
China Telecom
China
wujie@chinatelecom.cn

Hao Wang
Stevens Institute of Technology
United State
hwang9@stevens.edu

Zhengwei Qi
Shanghai Jiao Tong University
China
qizhwei@sjtu.edu.cn

Abstract

Large Language Model serving faces unprecedented demand, yet existing schedulers struggle to allocate resources fairly across diverse workloads. Current deployment strategies like First-Come-First-Served offer no client isolation against monopolization, while Requests Per Minute policy wastes resources during off-peak hours. The Virtual Token Counter attempts fair-sharing by tracking token consumption but inadequately captures LLM resource dynamics. The Transformer architecture's prefill-decode bifurcation creates conflicting resource patterns: memory-bound decode operations dominate latency, compute-bound prefill operations determine throughput, and GPU utilization follows independent batch-refresh patterns, making single-metric fairness unachievable.

We address these limitations with a dual-counter framework separating user and operator perspectives. The User Fairness Counter measures quality of service via weighted tokens and latency; the Resource Fairness Counter measures operational efficiency through throughput and GPU utilization. Since these metrics are only available post-execution, creating a scheduling paradox, we introduce a deterministic Mixture of Prediction Experts (MoPE) framework to predict user-perceived latency, output tokens, throughput, and GPU utilization. These predictions enable calculation of a unified Holistic Fairness score that balances both counters through tunable parameters for proactive fairness-aware scheduling. We implement this in Equinox, an open-source system with other optimizations like adaptive batching, and stall-free scheduling. Evaluations on production traces (ShareGPT, LMSYS) and synthetic workloads demonstrate Equinox achieves up to 1.3× higher throughput, 60% lower time-to-first-token latency, and 13% higher fairness versus VTC while maintaining 94% GPU utilization, proving fairness under bounded discrepancy across heterogeneous platforms.

1 Introduction

Large Language Models (LLMs) have reshaped artificial intelligence by combining modern neural networks with scalable computing and vast datasets. Their versatility spans tasks from content generation to software engineering, driving rapid adoption [4, 11, 32, 41, 42]. For example, ChatGPT gained 1 million users in five days—and 100 million in two months—while DeepSeek topped the U.S. iOS chart shortly after its January 2025 release [18, 21]. Yet, most prior research has primarily targeted improvements in model quality, inference speed [44, 49], and GPU utilization [14, 46], leaving fair

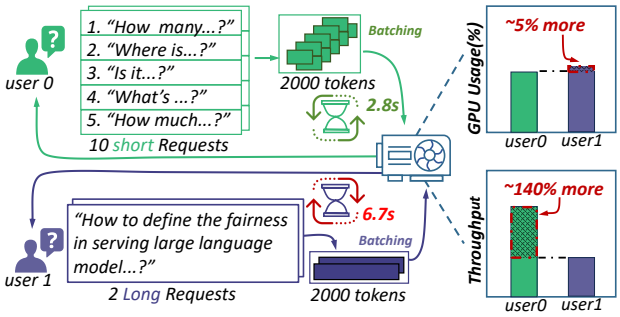


Figure 1. Token-count scheduling is unfair for independent, single-turn requests: with equal total tokens, many short vs. few long requests yield very different user latency, GPU utilization, and throughput. Batching mainly *shifts* the advantage—short identical requests benefit in-batch; without batching, queued short requests can lose. Setup: S-LoRA [3] on LMSYS Chat-1M [47], A100-80GB, matching VTC [39].

multi-tenant serving largely unexplored. This gap prompts a central question: *How can we allocate resources both efficiently and equitably as LLM demand and workload diversity grow?*

Most production deployments [12, 24, 28, 33, 36, 43] rely on First Come First Served (FCFS) scheduling, which processes requests strictly in arrival order, thereby offering no client isolation and allowing compute-intensive requests to monopolize resources and degrade co-tenants. To mitigate this, providers often impose static Requests Per Minute (RPM) quotas [26], a rate-limiting mechanism that caps the number of requests a client can send within a one-minute interval; however, these waste capacity during off-peak periods as the fixed allowance goes unused when demand is low.

The Virtual Token Counter (VTC) [39] addresses these scheduling limitations by adapting fair-share scheduling from network queuing and operating systems [9, 10, 13, 16, 22, 25]. In traditional domains, resource consumption patterns allow a single metric like CPU seconds or transmitted bytes to effectively represent overall usage, as these metrics naturally correlate with actual resource costs. VTC attempts to replicate this approach in LLM serving by adopting token count as its scheduling metric. The scheduler tracks cumulative input and output tokens per user and implements a work-conserving policy [20] that prioritizes users with lower token totals. This approach solves both prior failures: it prevents FCFS starvation by ensuring proportional service allocation while avoiding RPM's capacity waste through dynamic resource allocation instead of static quotas.

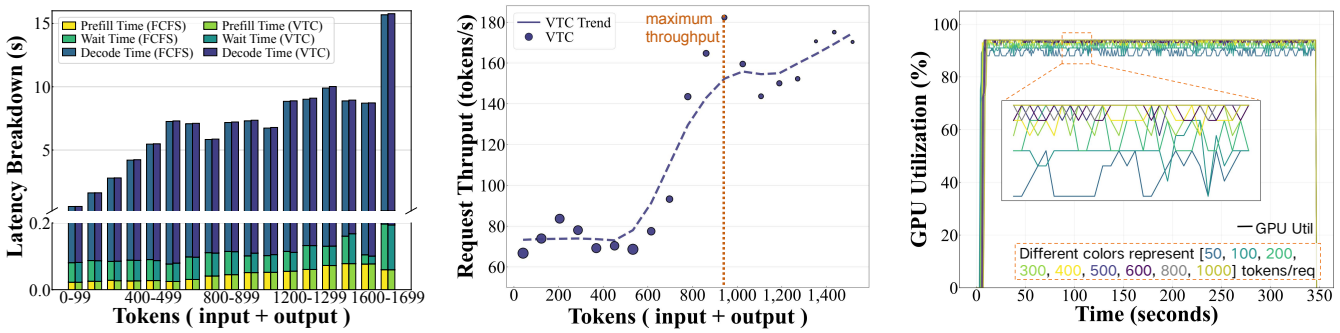
Key Insight. Although VTC improves upon FCFS and RPM, it inadequately adapts fair-share scheduling principles to LLM serving environments. Unlike traditional domains where one metric comprehensively captures resource usage, LLM serving exhibits metrics that actively conflict rather than correlate. The Transformer architecture's bifurcated nature drives this conflict because *prefill* processes prompts in parallel with superlinear compute-bound costs, while *decode* generates tokens sequentially with cache-dominated memory-bound costs [49]. Equal token counts thus yield vastly different resource consumption. As Figure 1 demonstrates, identical **token counts** produce divergent outcomes across **latency**, **throughput**, and **GPU utilization** due to opposing computational constraints. Latency grows monotonically as a memory-bound operation, with decode consuming over 90% of end-to-end time (Figure 2a). Throughput exhibits compute-bound non-monotonic patterns, initially increasing then declining after 1,000 tokens (Figure 2b). GPU utilization shows stepwise plateaus because shorter requests trigger frequent batch refreshes, introducing idle intervals during CPU-bound operations (Figure 2c). These observations show that token count cannot serve as a universal fairness metric and motivate a shift from single metric fairness to an explicitly multi objective formulation.

Key Idea. Motivated by this conflict, we propose a dual counter framework comprising the User Fairness Counter (UFC) and the Resource Fairness Counter (RFC) that acknowledges distinct stakeholder needs. The UFC ensures *equal* tenant Quality of Service (QoS) by combining weighted tokens with user perceived latency, with weighting justified because decode dominates both pricing and latency. The RFC ensures equitable operator allocation using GPU utilization and throughput, without weighting since throughput bottlenecks arise in prefill. We define holistic fairness (HF) as the joint consideration of UFC and RFC, reframing scheduling as the co-optimization of user experience and system efficiency rather than the division of a single resource. Implementing the dual-counter framework requires solving two key challenges: a paradox where computing UFC and RFC needs metrics that schedulers cannot predict, and the resulting tension between minimizing tenant latency and maximizing operator utilization.

Our Approach. We address both challenges through our open-source system Equinox, pairing a predictive framework with novel scheduling policy. Our Mixture of Prediction Experts (MoPE) provides low-overhead estimates for fairness components: user-perceived latency and output tokens for UFC; GPU utilization, and throughput for RFC. These predictions inform a fairness-aware scheduler jointly optimizing our composite metric, enabling the holistic balance that token-centric methods cannot provide.

Contributions. Our key contributions are:

- **Formalizing Holistic Fairness in LLM Serving.** We model LLM serving as a max-min fairness problem using UFC for per-client latency and weighted tokens, and RFC for GPU utilization and throughput, jointly optimizing metrics that transcend token-level fairness.



(a) FCFS/VTC shows **non-linear** latency variation across token ranges.

(b) VTC exhibits **non-linear** throughput patterns across token ranges.

(c) VTC demonstrates step-like GPU resource usage across token intervals.

Figure 2. Experiments on NVIDIA A100-80GB use real-world trace (lmsys-chat-1m) for (a,b) and synthetic workload for (c) with fixed total tokens (RPS × tokens/request) and 1:1 input:output ratio (prefill:decode). Real-world traces across vLLM and SGLang (Figure 16) confirm these patterns stem from architectural properties, not implementation choices.

- **Deterministic Prediction Framework for Scheduling.** We present the first deterministic MoPE framework that, unlike single proxy approaches [31] struggling with diverse output lengths, routes requests to specialized experts, reducing L1 prediction error for output tokens **from 80 to 33** versus prior methods.
- **Robust System Architecture for Fairness Enforcement.** Equinox, our open-source implementation combining MoPE with our fair scheduling algorithm, cuts worst-case service gaps by **42%** and average gaps by **86%** versus VTC, achieving fairness comparable to an `Oracle` predictor with only a 17% gap.

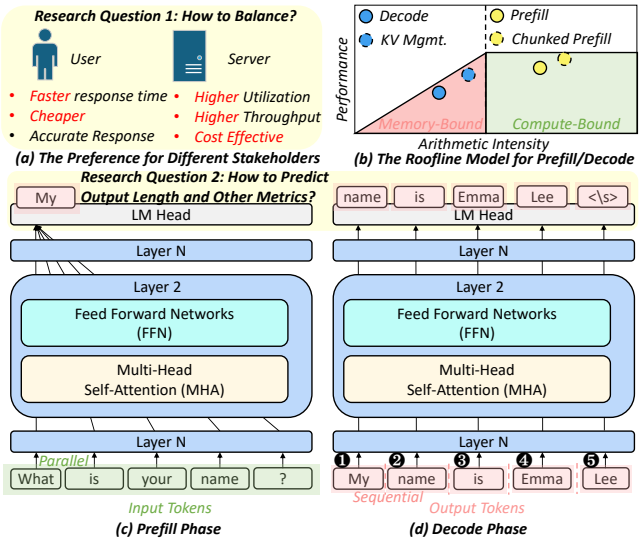


Figure 3. The core challenge of serving Large Language Models lies in balancing its two distinct inference phases: the parallel, compute-bound Prefill phase for processing prompts, and the sequential, memory-bound Decode phase for generating responses. Efficiently managing these conflicting workloads is essential to simultaneously achieve low latency for users and high throughput for service providers.

2 Background and Motivation

2.1 Large Language Models Serving

The LLM Serving Challenge. While the introduction established the dual-counter framework for balancing tenant experience (UFC) and operator efficiency (RFC), implementing this framework reveals fundamental challenges in LLM serving mechanics. Production deployments must handle concurrent multi-tenant requests while managing transformer inference's unique computational patterns, illustrated in Figure 3.

The prefill-decode bifurcation creates compounding nonlinearity beyond the basic distinction noted earlier [6, 7, 34]. As the KV cache expands during sequential token generation, both memory consumption and per-token latency increase disproportionately [29]. Although continuous batching improves GPU utilization by interleaving requests [46], the combination of workload heterogeneity and non-linear scaling properties demonstrates why token counts fundamentally fail as cost proxies.

From Token-Level to Holistic Fairness. Achieving holistic fairness requires co-optimizing four interconnected metrics: tenant-facing latency and token usage (UFC) alongside operator-facing GPU utilization and throughput (RFC). VTC's reliance on accumulated token counts cannot capture this multi-dimensional optimization space, manifesting in two critical failures. First, requests with identical token costs experience vastly different latencies, violating the principle

that similar payments yield similar service. Second, prioritizing purely by token accumulation ignores actual GPU cycles consumed, memory bandwidth utilized, and throughput achieved (Figure 5). This disconnection between the scheduling metric and actual objectives necessitates fundamental framework redesign.

Research Question 1: How can we design a scheduling framework that moves beyond token-counting to co-optimize the four metrics of holistic fairness: tenant-facing latency and token usage, and operator-facing GPU utilization and throughput?

2.2 LLM Token-Length Prediction

The Predictability Paradox in LLM Inference. Implementing Research Question 1's framework requires accurate predictions of all four fairness metrics before GPU pipeline entry. Paradoxically, while output generation remains inherently unpredictable due to dynamic token production, LLM inference's computational characteristics prove surprisingly amenable to prediction. The rapid, parallel prefill phase enables comprehensive input characteristic gathering before scheduling decisions, while the sequential decode phase exhibits predictable memory-bound behavior through systematic KV cache traversal.

Current approaches fail to exploit this opportunity, ranging from coarse proxy model categorizations [30, 31] to regression models and lightweight heuristics [5, 14, 17, 35, 37, 40, 45]. Figure 4a reveals that single proxy models trained on specific chatting datasets (Llama-7B, GPT-4, Vicuna-13B) and unified models trained across all data (All Models) using different loss function (L1/MSE) struggle to capture LLM complexity, resulting in high Mean Average Percentage Error (MAPE) for a large fraction of predictions. Figure 4b further demonstrates the challenges facing state-of-the-art proxy models by breaking down Mean Average Error (MAE) and MAPE by actual output tokens. While absolute error increases with length for both approaches, a single proxy model show significantly higher absolute error compared to Equinox, particularly struggling with long sequences where prediction errors compound dramatically. Training these models employs the same methodology [30] and we evaluate them on lmsys-chat-1m dataset [47].

Beyond Token Prediction: The Multi-Metric Challenge. The fundamental limitation extends beyond accuracy. Existing predictors estimate only token counts, providing merely one-quarter of holistic fairness's required metrics. Computing UFC demands predicting both user-perceived latency and tokens, as latency determines service quality. Computing RFC requires GPU utilization and throughput predictions that capture true resource consumption. Since token-to-metric relationships prove neither linear nor monotonic, schedulers equipped solely with token-length predictions remain blind to actual costs imposed on tenants and operators. While Research Question 1 defines the necessary scheduling framework, implementation requires solving a more fundamental prediction challenge—schedulers cannot optimize unmeasurable metrics, and current predictors provide insufficient information.

Research Question 2: How can we design a prediction framework that accurately estimates not only token length but also the latency, GPU utilization, and throughput required to enable a holistically fair scheduler?

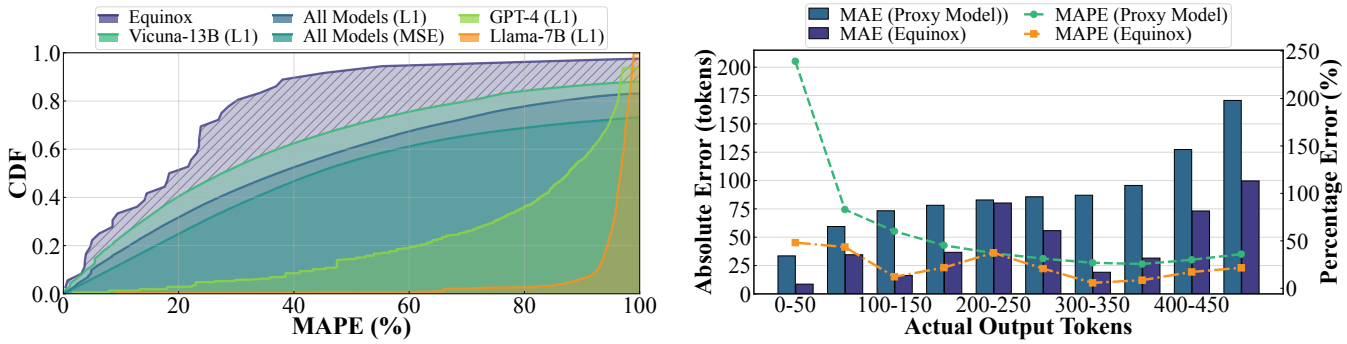
3 Defining Holistic Fairness

As we argued, the interdependent nature of batched GPU execution necessitates a fairness model that moves beyond the traditional "fair share" paradigm to consider the distinct objectives of both principals in the service agreement. We therefore formalize this concept of holistic fairness by evaluating it through two complementary lenses: the **user** (representing the tenant or service consumer) and the **system** (representing the service operator).

User fairness guarantees equitable access by equalizing response latency and token allocation, thereby consistently satisfying user expectations. System fairness, in turn, emphasizes efficient resource allocation to maximize throughput and maintain full GPU utilization, which directly benefits the operator by lowering operational costs. To formalize and co-optimize these competing goals, we introduce two distinct fairness counters, which represent four metrics include token counts, latency, GPU utilization and throughput.

3.1 User-Fairness Counter

The User-Fairness Counter (UFC) quantifies service quality from the user's perspective by combining two essential metrics: token consumption and perceived latency. When the scheduler selects a request for GPU execution, it updates the corresponding client's UFC to reflect their accumulated resource usage. The token component follows established industry pricing models [2, 8, 15, 27] and VTC conventions, weighting predicted output tokens ($\text{Tokens}_{\text{req}}^{\text{out}}$) four times more heavily than input tokens ($\text{Tokens}_{\text{req}}^{\text{in}}$). This **4x** multiplier reflects the computational asymmetry between prefill and decode phases, where generating output tokens consumes substantially more resources than processing input prompts. The latency component captures the complete user experience through two measurements. First, $\text{WaitTime}_{\text{req}}$ tracks the elapsed time since request submission to the server queue. Second, $\text{PredictTime}_{\text{req}}$ estimates the expected GPU inference duration once execution begins. Together, these metrics represent the total service time users experience from submission to completion.



(a) CDF of prediction error (MAPE) comparing single proxy models, unified models, and Equinox.

(b) MAE and MAPE breakdown by actual output tokens for proxy models versus Equinox.

Figure 4. Error analysis of LLM prediction approaches demonstrates (a) limitations of current proxy models in capturing complex behaviors and (b) challenges in managing diverse token ranges, both addressed by Equinox through dynamic routing and expert specialization (detailed in section 6).

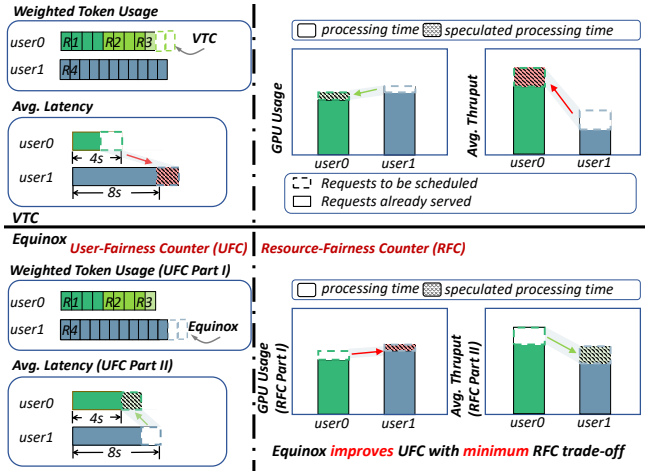


Figure 5. Holistic fairness in LLM scheduling. VTC picks *user0* for having fewer tokens yet fairness spans users and resources. *Equinox* serves the request with the smallest Holistic Fairness score computed from two counters, a User Fairness Counter that blends token usage and latency and a Resource Fairness Counter that reflects GPU utilization and throughput, achieving max min fairness; weights α and β tune priority with α greater than β to favor user experience.

For each user f, the UFC updates when selecting a new request (req) according to the following calculation:

$$UFC \leftarrow UFC + \omega_f \frac{\text{Tokens}^{\text{in}}_{req} + 4 \cdot \text{Tokens}^{\text{out}}_{req}}{1 + \delta \cdot (\text{WaitTime}_{req} + \text{PredictTime}_{req})}$$

, where ω_f is the priority weight of user f, and δ configures the latency compensation factor, which is tested and set as 0.1 in our system. This factor prioritizes backlogged users by scaling their UFC with accumulated latency.

3.2 Resource-Fairness Counter

The Resource-Fairness Counter (RFC) maintains server-side fairness by dynamically updating after processing each request batch. RFC integrates 2 per-batch critical performance metrics: system throughput in tokens per second, GPU utilization percentage. For every client, the update operation of RFC per request follows the implementation exactly:

$$RFC \leftarrow RFC + \omega_f \cdot \text{TPS} \cdot \text{Util}_{GPU},$$

where TPS is the estimated request throughput (tokens per second in GPU), Util_{GPU} is the estimated GPU utilization percentage, scaled from 0 to 1.

3.3 Putting it Together

With the UFC and RFC established, we can now define holistic fairness. The core challenge lies in co-optimizing these two, often competing, sets of objectives. A scheduler that exclusively minimizes UFC might select latency-sensitive requests at the cost of poor batching, harming system throughput. Conversely, a scheduler that strictly maximizes RFC could prioritize high-throughput batches, potentially starving users with less computationally convenient requests.

Our approach is to unify these counters into a single, composite score that guides scheduling decisions. We formally define the Holistic Fairness (HF) for a user f as a weighted linear combination of their normalized UFC and RFC values:

$$\text{HF}_f = \alpha \cdot \text{UFC}_f + \beta \cdot \text{RFC}_f,$$

where α and β are weighting parameters that represent the system's strategic priorities, with $\alpha + \beta = 1$.

The scheduler's objective is to maintain max-min fairness by always prioritizing requests from the user with the lowest current HF. This ensures that over time, all users receive a balanced QoS according to our holistic definition. Crucially, the weights α and β allow operators to tune the fairness

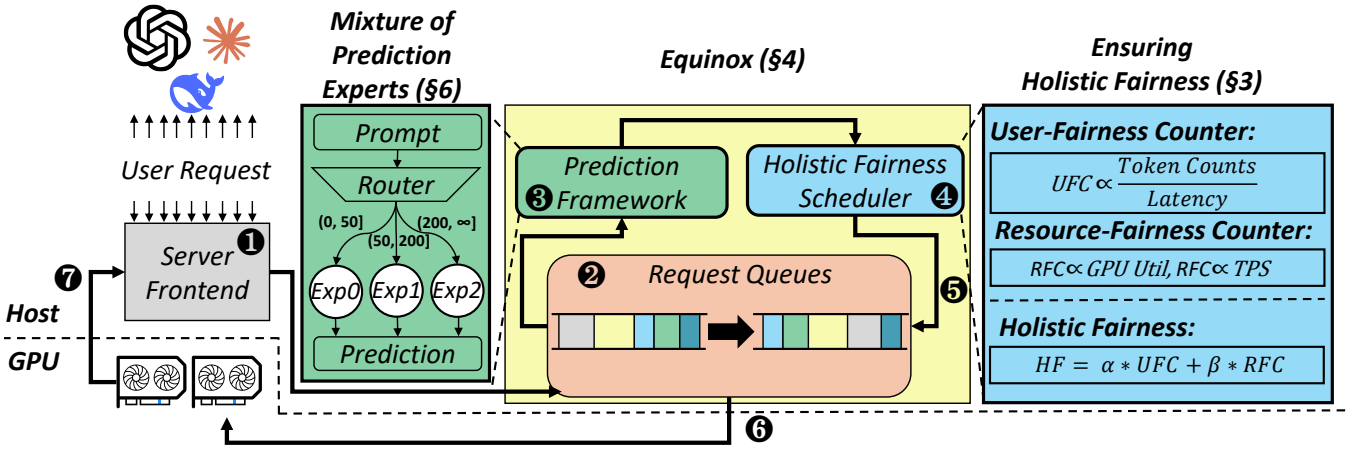


Figure 6. System Design Overview

policy. In our framework, we assert that the end-user experience is the ultimate goal of the service. Therefore, we assign a higher weight to the UFC, setting $\alpha > \beta$, reflecting a strong but not absolute preference for user-centric metrics over system-centric ones. The sensitivity and impact of α and β are evaluated in subsection 7.6. As an illustration, Figure 5 applies Eq. (3.3) and our policy choice $\alpha > \beta$: a traditional token-aware scheduler (VTC) would select *user0* solely because it has accumulated fewer tokens. However, Equinox recognizes that *user0* already enjoys lower latency. Because our policy weights the latency-sensitive UFC more heavily, the composite HF score correctly identifies *user1* as more underserved, demonstrating a more nuanced approach to fairness.

4 System Design

We introduce Equinox, a multi-stage system designed to handle the demanding task of serving Large Language Model (LLM) inference requests by enforcing holistic fairness. Holistic fairness represents our core architectural principle, balancing user-centric metrics (token allocation and latency) with system-centric objectives (GPU utilization and throughput) to achieve equitable service delivery. The architecture comprises four components that collectively implement this holistic fairness paradigm: (i) The Server Frontend ingests requests, performs authentication and semantic validation, and applies rate limiting. (ii) Request Queues buffer validated tasks, decouple ingestion from execution and support dynamic reordering based on holistic fairness scores. (iii) A Mixture of Prediction Experts predicts output lengths by routing queries to specialized proxy models, supplying accurate scheduling estimates essential for holistic fairness calculations. (iv) The Holistic Fairness Scheduler reprioritizes requests by computing and balancing UFC and RFC scores through the holistic fairness equation, ensuring equitable resource allocation while maximizing system efficiency.

Workflow Overview. Figure 6 illustrates the end to end pipeline implementing holistic fairness. Clients submit requests to the **Server Frontend** (❶); invalid inputs are dropped, whereas valid ones enter the distributed **Request Queues** (❷). The **Prediction Framework** estimates response token counts and maps historical performance metrics for each queued request (❸), providing critical input for holistic fairness computation. The **Holistic Fairness Scheduler** employs these predictions and mappings to calculate accurate UFC and RFC scores. The scheduler then applies the holistic fairness equation to rank clients and batches requests for GPU execution, selecting those from clients with the lowest holistic fairness scores (❹). After reprioritization based on holistic fairness (❺), selected requests execute on GPU accelerators (❻), where the system continuously monitors key performance indicators like latency, compute utilization, and throughput, leveraging insights derived from offline profiling on representative workloads such as the lmsys-chat-1m dataset. These metrics feed back into the holistic fairness calculation, creating a continuous optimization loop. Upon completion, generated tokens stream back through the Server Frontend for response assembly and delivery (❼), completing the holistic fairness driven service cycle.

5 Algorithm Analysis

The core logic of Equinox's fair scheduling algorithm is presented in algorithm 1. The algorithm maintains a per-client holistic score HF_c. It initializes the predictor P and a priority queue Q keyed by HF_c. The loop runs continuously to be consistent with continuous batching. For each incoming request req from client c, the scheduler invokes P.predict(req) to estimate $\mathrm{Tokens}_{\mathrm{out}}$ and uses P.map($\cdot$) to attach predicted (Latency, $\mathrm{GPU}_{\mathrm{util}}$, TPS) to the request. The request's arrival time is recorded, and the request is enqueued keyed by the current HF_c.

The scheduler creates an empty batch B and, while Q is non-empty, repeatedly selects the client c^* with the minimum

Algorithm 1: Holistic Fair Scheduling Algorithm

Input: α, β, δ, batch size L_b, GPU memory M
Output: B

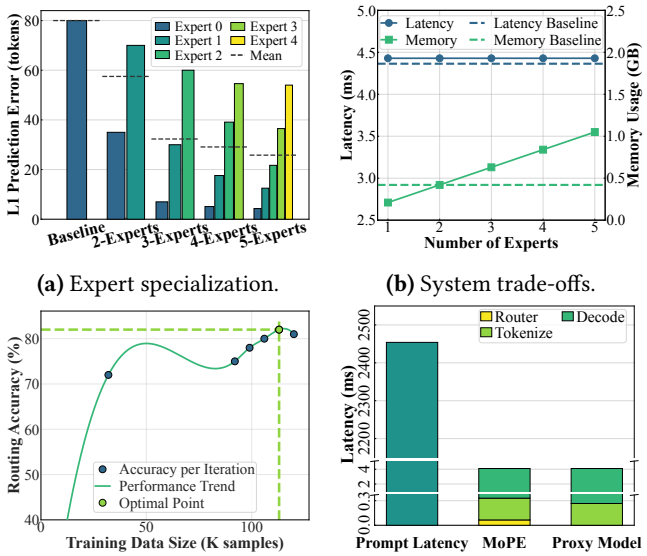
Initialize Predictor P, and PriorityQueue Q;
while *continuously* **do**
  foreach *incoming req from client c* **do**
    $\text{req.Tokens}_{\text{out}} \leftarrow P.\text{predict}(req)$;
    $(\text{req.Latency}, \text{req.GPU}_{\text{util}}, \text{req.TPS}) \leftarrow P.\text{map}(\text{Tokens}_{\text{out}})$;
    record req.arrival;
    Q.enqueue(req, HF_c);
  end
  $B \leftarrow \emptyset$;
  while Q *not empty* **do**
    $c^* \leftarrow \arg\min_{c \in Q} \text{HF}_c$;
    $req \leftarrow Q$.dequeue(c^*);
    if canSchedule(req, B, M, L_b) **then**
      $B \leftarrow B \cup \{req\}$;
      updateCounter(req, c^*);
    end
  end
  Execute B on GPU;
  foreach *completed request from client c* **do**
    Update HF_c and $P.map()$ with actual metrics;
  end
end

HF_{c^*}, dequeues its head request, and check feasibility via canSchedule(req, B, M, L_b). canSchedule(req, B, M, L_b) checks whether the new request satisfies the current batch-size and GPU-memory resource constraints. If feasible, the request is loaded into B and the scheduler calls updateCounter(req, c^*) to update HF_{c^*} using the request's predicted ($\text{Tokens}_{\text{out}}$, Latency, GPU_{util}, TPS) together with the observed wait time (now$-req$.arrival). Because selection always picks the smallest HF, clients with slower-growing UFC tend to remain with lower HF and are preferentially admitted next, while the RFC nudges the batch toward efficient GPU utilization.

After building B, the batch executes on the GPU. Upon completion, the scheduler refreshes HF_c using the actual metrics and updates $P.\text{map}(\cdot)$ accordingly. This closes the feedback loop and continuously calibrates predictions to the observed hardware behavior.

6 Prediction Framework for Scheduling

To address the limitations of relying on a single, generic proxy model for token count prediction, we introduce *MoPE*, a framework comprising multiple specialized predictors (*experts*) coordinated by a routing layer. MoPE dynamically assigns incoming requests to the most suitable expert, effectively handling diverse prompt types and output token lengths characteristic of modern LLM workloads. Unlike



(a) Expert specialization. **(b)** System trade-offs.

(c) Router learning patterns. **(d)** Latency breakdown.

Figure 7. MoPE insights and data analysis.

conventional mixture-of-experts architectures focused on generation, MoPE specifically targets accurate token count prediction within a lightweight structure suitable for production LLM serving.

The offline training pipeline (Figure 8, left) first trains the router. Using the true output lengths from the training corpus, the router learns to classify prompts based on input length thresholds and automatically identified keywords indicative of output length classes. This process employs feature embedding and similarity lookups, balancing different signals via a mixing weight, and iteratively selects data to maximize classification accuracy.

The expert training phase then partitions the corpus according to the router's learned classifications (e.g., short, medium, long regimes). For each partition, a specialized regression BERT-base model is fine-tuned, typically by adapting a pretrained encoder with a regression head [31]. Training focuses on prediction accuracy (e.g., minimizing L1 error) via standard techniques like stratified splits and early stopping.

As detailed in Figure 7, empirical evaluations show that a configuration of three experts strikes the optimal balance. Figure 7a Specifically, the L1 prediction error is 80 for one expert, 33 for three experts, and 25 for five experts. We reduce the model weights from FP32 to BF16 precision to decrease memory consumption for multiple experts, therefore the resource usage (memory, latency) increases substantially beyond three experts (Figure 7b). For router training, We present the router accuracy performance across training data sizes up to 120k from LMSYS dataset [47]. Accuracy gradually increases before 50k samples, reaching a local optimum. It then achieves peak accuracy of 80% at approximately 110k

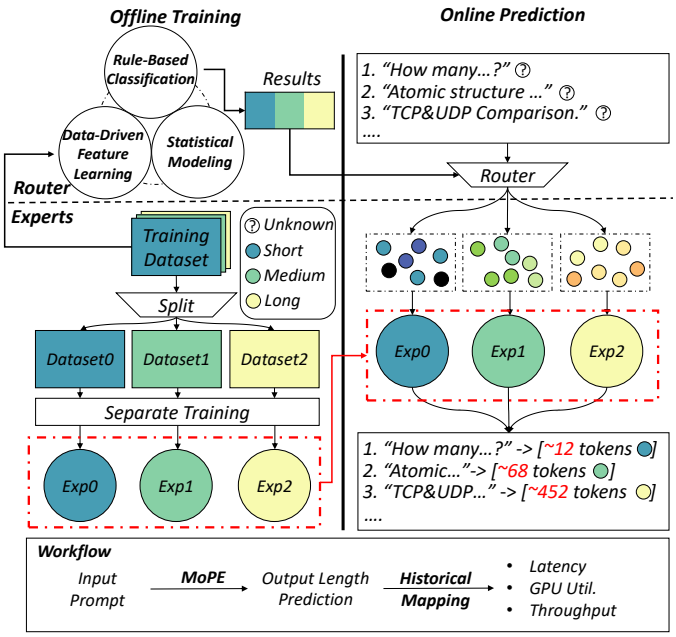


Figure 8. Overview and workflow of MoPE architecture

samples in Figure 7c. We report the end-to-end latency breakdown of MoPE and prompt inference in Figure 7d. MoPE only introduces router overhead (0.02 ms) compared to the original proxy model, with total MoPE inference time being 4.5 ms. This represents less than 1% of the average prompt latency (2,400 ms), demonstrating that the overall latency added by MoPE remains negligible compared to total prompt processing time. We also compare the end-to-end prediction errors between MoPE and the stae-of-the-art proxy model, as detailed in Figure 4. These experiments collectively support the three-expert configuration as maximizing predictive accuracy relative to system cost.

During online prediction, the router rapidly classifies an incoming prompt and directs it to the relevant expert (`short`, `medium`, or `long`), incorporating the target LLM identity during preprocessing. The selected expert performs a forward pass to generate the token length prediction. Finally, we model token-latency-throughput-GPU utilization relationships using historical offline data in Figure 2. For inputs processed by MoPE, we use this historical mapping to predict all metrics required for computing Holistic Fairness (HF). The offline modeling method we use has been validated for latency prediction [19], we extend it to estimate GPU utilization and throughput.

7 Evaluation

We implemented Equinox atop existing systems (S-LoRA [3], vLLM [23], SGLang [48]) using ~1000 lines of Python. Its lightweight layer coordinates optimizations like continuous batching, chunked prefill [1], PagedAttention [23], and KV-cache reuse, and supports adapter-level fairness (e.g., per-LoRA).

7.1 Evaluation Methodology

Workload and Testbed. We use synthetic workloads and real traces from LMSYS Chatbot Arena [47] and ShareGPT [38]. During runs, MoPE predicts token lengths and other resource demands, and the scheduler interleaves prefill/decode to balance UFC and RFC. For synthetic traces, experiments run on an A100 GPU (80GB) with Llama-2-7b, mirroring VTC's setup. For real-world traces, experiments run on an 8-A100 GPU (40GB) cluster with Llama-2-70b. The CPU is Intel(R) Xeon(R) Gold 5218@2.30GHz with 256 GB DRAM. For vLLM and SGLang, we use Tensor Parallelism (TP) = 8.

Baselines. For scheduling algorithm, we compare Equinox against FCFS and VTC. For prediction, we choose single proxy model [31] as the baseline. We further conduct an ablation study removing MoPE to isolates its contribution to fairness.

MoPE Configuration. Following Section 6, we use 3 experts and a fariness metirc mapping trained on LMSYS Chatbot Arena dataset. Boundaries partition the training data into $33th$, $66th$ and $99th$ points based on output length (<53, 53-210, >210 tokens). Generalizability is tested on the unseen ShareGPT dataset (Section 7.3).

Metrics. Key metrics include:

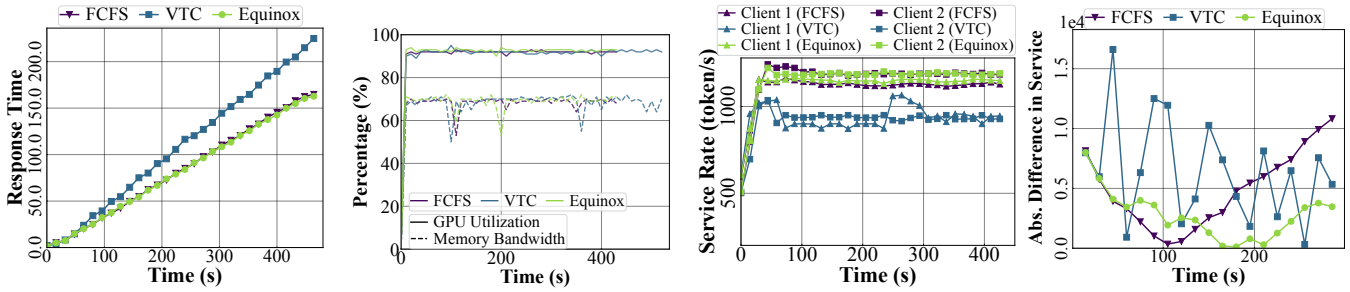
- **Service Rate:** per-client weighted tokens processed per second.
- **Service Difference:** Absolute difference in accumulated weighted service between clients.
- **latency:** We use response time (time to first token) and end-to-end latency to measure latency in different cases.
- **Jain's Fairness Index** is a widely-used metric for evaluating the fairness of resource allocation in networked systems. In our context, we compute Jain's Fairness Index by letting x_i denote the **HF** of client i, which is defined in subsection 3.3. The index is mathematically defined as:

$$J(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \tag{1}$$

where x_i represents the allocation for the i^{th} client, and n is the total number of clients. The value of J ranges from $\frac{1}{n}$ (minimum fairness, when one client monopolizes all resources) to 1 (maximum fairness, when resources are equally distributed).

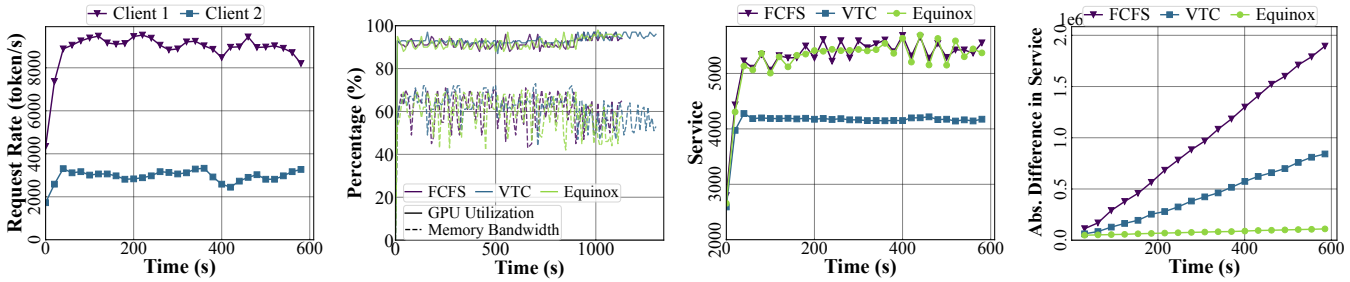
7.2 Evaluation with Synthetic Workloads

We design experiments using synthetic workloads to rigorously evaluate the fairness properties of Equinox under controlled conditions. These scenarios allow for clear visualization and analysis of the scheduler's behavior. This section focuses on two key scenarios: a balanced load with heterogeneous client demands and a stochastic arrival pattern mimicking real-world unpredictability. Further analyses



(a) Total Response Time **(b)** GPU Util. & Bandwidth **(c)** Per-client Service Rate **(d)** Service Difference

Figure 9. Balanced load scenario. Equinox maintains fairness despite different request rates and lengths.



(a) Client Request Rate **(b)** GPU Util. & Bandwidth **(c)** Total Service Rate **(d)** Service Difference

Figure 10. Poisson arrivals with heterogeneous requests. Equinox maintains fairness under stochastic load.

under overload, underload, and dynamic load conditions are detailed in Appendix A.

7.2.1 Balanced Load Scenario. We configure a scenario where the system operates under a balanced load but serves clients with distinct demands. Client 1 sends requests at a constant rate of 2 req/s for input lengths of 100 and output lengths of 400, while Client 2 sends requests at a lower rate of 1 req/s but for longer output lengths of 900. This setup assesses fairness when request rates and per-request service needs differ under a manageable load.

Overall performance. Equinox successfully maintains fairness between clients with differing request characteristics while achieving high efficiency, as illustrated in Figure 9. Appendix Figure 17a show that, Equinox's fairness algorithm prioritizes throughput and delivers a higher total service rate than FCFS or VTC (1.3x). Because the balanced workload alternates between light and heavy contention, Equinox is able to capture both advantages: its service distance and aggregate service rate are simultaneously better than those of FCFS and VTC.

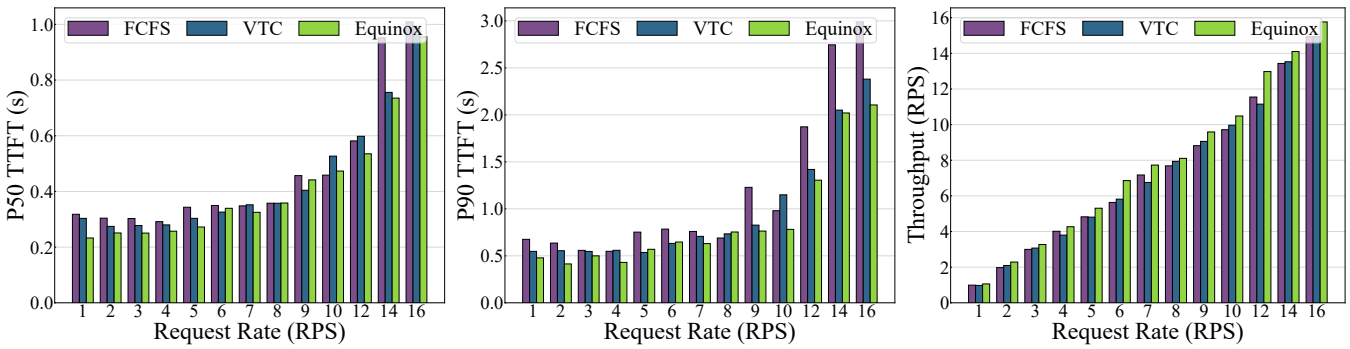
Breakdown. The fairness achieved by Equinox is evident in the per-client service rate (Figure 9c), showing comparable service allocation over time and modestly higher absolute throughput than FCFS or VTC (1.3x). Crucially, the accumulated service difference remains tightly bounded (Figure 9d), demonstrating consistent fairness enforcement. Mirroring the underload study, Equinox also maintains up to 60 % lower response times than VTC (Figure 9a) while sustaining high GPU utilization (Figure 9b), confirming that fairness need not come at the cost of efficiency.

7.2.2 Stochastic Arrivals Scenario. To simulate more realistic, unpredictable traffic, we employ a Poisson process for request arrivals. Client 1 issues requests with an average rate of 16 req/s, primarily requiring prefill computation (input lengths of 512 and output lengths of 32). Client 2 issues requests at a lower average rate of 3 req/s, focusing on decoding (input lengths of 32 and output lengths of 512). Detailed request pattern is shown in Figure 10a. This configuration tests Equinox's robustness under stochastic arrivals and mixed computational demands (prefill-heavy vs. decoding-heavy).

As depicted in Figure 10, Equinox demonstrates holistic fairness even when subjected to stochastic arrivals and heterogeneous request types. It effectively keeps the total service rate virtually identical to FCFS in Figure 10c, confirming that its fairness logic does not sacrifice throughput even when arrivals are highly variable. Simultaneously, it significantly reduces the accumulated service difference compared to both baseline methods (Figure 10d). VTC's token-level metric fails here because it does not have the predictor mechanism thus undervaluing the long-decode requests of Client 2 while Equinox's MoPE design corrects this bias.

7.3 Evaluation with Real-world Workloads

This section focuses on Equinox's performance within the SGLang and vLLM platform and presents a direct comparison



(a) P50 TTFT **(b)** P90 TTFT **(c)** Throughput

Figure 11. Performance comparison using ShareGPT trace in SGLang.

of fairness across multiple setups and metrics. Analyses using traces within S-LoRA are detailed in Appendix B.

7.3.1 SGLang with ShareGPT Dataset. We configure an experiment using the SGLang serving system [48] processing requests derived from the ShareGPT dataset. The workload comprises 256 simulated clients, with the aggregate request arrival rate dynamically varying between 1 and 16 requests per second (RPS) and the total number of prompts as 1280, which is a standard benchmark integrated into SGLang Benchmark[1]. This setup rigorously tests Equinox's performance, scalability, and responsiveness under fluctuating demand within a state-of-the-art serving platform designed for complex interaction patterns.

Equinox demonstrates significant performance advantages when integrated into the SGLang environment and subjected to the ShareGPT dataset, as illustrated in Figure 11. Compared to FCFS and VTC, Equinox achieves improvements in P50/P90 time to first token (TTFT) (up to 30%) and mild system throughput (up to 25%) when RPS is high, highlighting its efficiency in realistic settings. Although designed for fairness, Equinox monitors UFC/RFC and reorder and prioritize requests from clients are vulnerable to starvation. This mechanism, akin to priority scheduling based on aging or urgency, partially mitigates head-of-line (HOL) blocking, thereby optimizing TTFT and system throughput.

7.3.2 vLLM with ShareGPT Dataset. We also evaluate performance using vLLM, again leveraging the ShareGPT workload. In this specific experimental setup, the number of concurrent clients varies from 1 to 8, and the per-client request rate is kept constant at 3.5 requests per second in a poisson distribution. Each client sends 1000 requests in total.

The Figure 12 illustrates both the performance and fairness gains of Equinox across multiple metrics. Equinox achieves higher and more stable Jain's fairness index compared to FCFS and VTC (up to 33% in Figure 12a) and slightly lower average TTFT and end-to-end latency (~5% in Figure 12b and Figure 12d), ensuring better quality of service for each client. It maintains a slightly higher per-client service rate (Figure 12c), reflecting efficient token-level scheduling. Together, these improvements validate Equinox's robustness and fairness under different setups of clients in vLLM.

[1] https://github.com/sgl-project/sglang/tree/main/benchmark/hicache

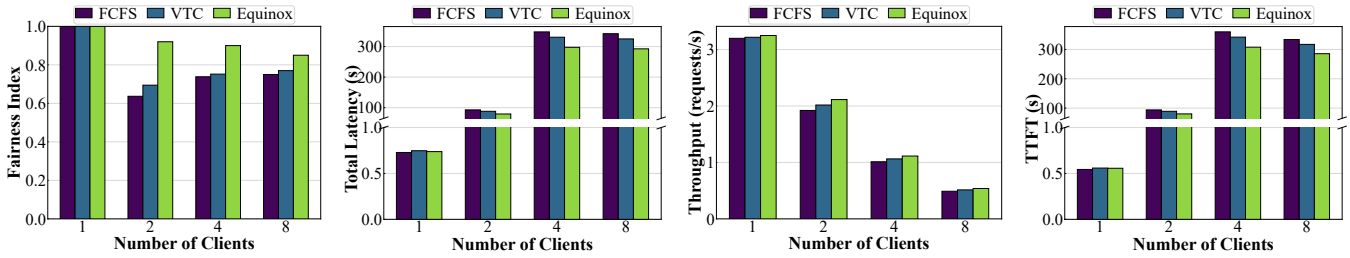
Table 1. Ablation of fairness (Max/Avg/Var of Service Difference) across schedulers and predictors under synthetic load.

Scheduler Variant	Max Diff	Avg Diff	Diff Var
FCFS	1864.42	1400.40	44868.58
VTC	1505.13	1106.31	45151.63
VTC + Single	3344.00	2992.92	44588.27
VTC + MoPE	**1390.00**	**1003.82**	**35400.60**
VTC + Oracle	**1375.80**	**999.12**	**33859.38**
Equinox + Single	1385.23	1005.47	34210.75
Equinox + MoPE	**865.62**	**150.64**	**28578.68**
Equinox + Oracle	**715.12**	**99.80**	**22156.91**

7.3.3 Cross-Serving System Fairness Comparison. To provide a consolidated assessment of fairness across different LLM serving system, we use Jain's Fairness Index to compare the fairness achieved by Equinox against FCFS and VTC. This comparison spans multiple popular serving systems: S-LoRA, vLLM, and SGLang, the detailed workload and setup are detailed in subsection 7.1. As depicted in Figure 13, Equinox consistently delivers 13% superior fairness compared to both FCFS and VTC. Counterintuitively, VTC's fairness index on HF is no better than FCFS. This demonstrates that scheduling policies relying on multipled metrics improves and ensures equitable service opportunities for clients.

7.4 Ablation Study

We isolate the impact of MoPE within Equinox by comparing its fairness performance against baseline schedulers (FCFS, VTC) under a challenging synthetic workload (details in subsubsection 7.2.2). We use three predictors: MoPE, a single proxy model (Single) [30], and an Oracle (perfect prediction) as an ideal benchmark. Fairness is measured by the



(a) Jain's Fairness Index **(b)** Per-client Response Time **(c)** Per-client Recv. Service Rate **(d)** Per-client TTFT

Figure 12. Performance comparison using ShareGPT datasets in vLLM.

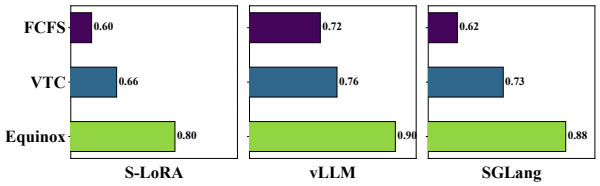
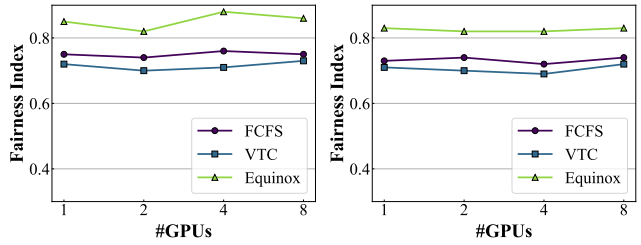


Figure 13. Fairness comparison across S-LoRA, vLLM and SGLang.



(a) vLLM Fairness Scalability **(b)** SGLang Fairness Scalability

Figure 14. Experiments to test Jain's Fairness on vLLM and SGLang scaling GPU counts from 1 to 8

maximum, average, and variance of the accumulated absolute service difference between clients over the experiment (lower is better), presented in Table 1.

Analysis. Table 1 shows FCFS provides poor fairness. Baseline VTC, lacking predictive capabilities, also performs poorly as it cannot account for varying request costs. Incorporating token length prediction significantly boosts the fairness of VTC. VTC + MoPE approaches the Oracle performance, demonstrating the necessity of accurate prediction for token-level fairness. However, Equinox's core scheduling logic yields far greater fairness improvements. Equinox variants consistently outperform their VTC counterparts using the same predictor. Notably, Equinox + MoPE dramatically reduces service differences compared to VTC + MoPE, confirming the advantage of Equinox's holistic fairness approach over VTC's token-centric method. Equinox + single proxy model offers little benefit over VTC with the same predictor, highlighting that advanced scheduling requires good prediction. The close performance of Equinox + MoPE to Equinox + Oracle further validates MoPE's effectiveness.

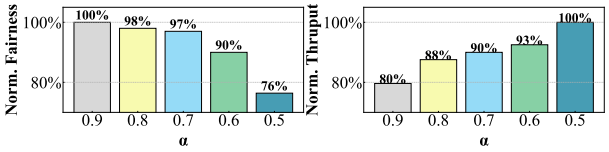


Figure 15. Impact of α/β weight ratios on client fairness and system throughput

7.5 Scalability Analysis

We repeated experiments on vLLM and SGLang, scaling GPU counts from 1 to 8 with proportional tensor parallelism (TP) adjustments. Across all configurations, Equinox consistently outperformed VTC and FCFS in fairness. This demonstrates that Equinox's core innovations—UFC/RFC, MoPE, and scheduling algorithm—exhibit robust setup-agnostic behavior. While extending Equinox to multi-node deployment would require additional engineering efforts (e.g., distributed UFC/RFC updates), we maintain that its high-level design principles remain fundamentally sound.

7.6 Hyperparameter Analysis

We repeated experiments on SGLang at RPS=16, adjusting α from 0.5 to 0.9 while setting β to 1 - α. For each configuration, we recorded Jain's Fairness Index for client P90 TTFT and system throughput (requests per second), normalizing values to their maximums. As shown in Figure 15, reducing α decreases latency fairness but increases throughput: At $\alpha = 0.9$, peak fairness occurs but throughput decreases by 20% compared to $\alpha = 0.5$; conversely at $\alpha = 0.5$, maximum throughput is achieved but fairness drops by 23%. We choose the optimal balance as $\alpha = 0.7$ and $\beta = 0.3$, which maintains 97% peak fairness (only 3% below maximum) while achieving 90% of maximum throughput. This configuration was selected to prioritize user-perceived latency while sustaining high system performance.

8 Related Work

LLM Serving Systems. Recent systems have optimized latency and throughput, but they do not fundamentally resolve the resource allocation challenges posed by the prefill-decode bifurcation. For instance, *Sarathi-Serve* introduces

chunked-prefills to admit new requests without stalling ongoing decodes, while *DistServe* disaggregates prefill and decode phases onto separate GPU pools to eliminate interference. Other innovations include dynamic early exits in *Apparate* and neuron caching in *PowerInfer*. Critically, while these methods mitigate interference, they don't alter the underlying per-iteration computation, leaving the pipeline bound by either prefill or decode. Consequently, they optimize around the architectural split without addressing its fairness implications, typically defaulting to simplistic FCFS or token-based schedulers.

Fair Scheduling in LLM. Token-based fairness mechanisms, used in systems like VTC, aim to prevent resource monopolization by prioritizing users with the fewest accumulated tokens. However, this approach is flawed because the prefill-decode bifurcation means identical token counts correspond to vastly different resource consumption; decode disproportionately affects latency, while prefill drives throughput bottlenecks. This mismatch makes token-counting an inaccurate proxy for the actual costs imposed on users or the system. Our dual-counter framework overcomes this limitation by explicitly separating user concerns (the UFC, combining weighted tokens and latency) from operator concerns (the RFC, tracking GPU utilization and throughput). This transforms fairness from a single-metric problem into a multi-objective optimization that reflects the true computational dynamics of LLM serving.

Length Prediction in LLM. Existing prediction models often focus on a single metric, such as using proxy models to estimate output token length for shortest-job-first scheduling. While techniques like range prediction or learning-to-rank offer some improvement, they are insufficient for holistic fairness, which requires estimating four distinct metrics: user latency and token counts, alongside operator utilization and throughput. Our Mixture of Prediction Experts (MoPE) reconceptualizes this task. Instead of predicting one value, MoPE routes prompts to specialized experts that estimate all four fairness components with minimal overhead, enabling the comprehensive scheduling decisions that single-metric predictors cannot support.

9 Conclusion

Equinox redefines fairness in multi-tenant LLM serving by recognizing that the prefill-decode bifurcation makes single-metric fairness fundamentally unachievable. Its dual-counter framework separates user concerns through the User Fairness Counter (combining weighted tokens with perceived latency) from operator concerns through the Resource Fairness Counter (tracking GPU utilization and throughput), transforming fairness from resource allocation to multi-objective optimization. The deterministic MoPE enables this framework by predicting required metrics before execution, resolving the scheduling paradox where fairness computation requires post-execution information. Experiments show that Equinox achieves up to 1.3× higher throughput, 60% lower time-to-first-token latency, and 13% higher fairness compared to VTC while maintaining 94% GPU utilization on production traces. These results demonstrate that holistic fairness, which balances both user experience and operator efficiency, is achievable in heterogeneous LLM serving systems when scheduling decisions account for the computational dynamics of transformer .

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 117–134. https://www.usenix.org/conference/osdi24/presentation/agrawal

[2] Anthropic. 2025. Pricing. https://www.anthropic.com/pricing

[3] Sara Babakniya, Ahmed Roushdy Elkordy, Yahya H. Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy, and Salman Avestimehr. 2023. SLoRA: Federated Parameter Efficient Fine-Tuning of Language Models. arXiv:2308.06522 [cs.LG] https://arxiv.org/abs/2308.06522

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[5] Ke Cheng, Wen Hu, Zhi Wang, Peng Du, Jianguo Li, and Sheng Zhang. 2024. Enabling Efficient Batch Serving for LMaaS via Generation Length Prediction. arXiv:2406.04785 [cs.DC] https://arxiv.org/abs/2406.04785

[6] Minsik Cho, Mohammad Rastegari, and Devang Naik. 2024. KV-Runahead: Scalable Causal LLM Inference by Parallel Key-Value Cache Generation. arXiv:2405.05329 [cs.DC] https://arxiv.org/abs/2405.05329

[7] Cade Daniel, Chen Shen, Eric Liang, and Richard Liaw. 2023. *How Continuous Batching Enables 23x Throughput in LLM Inference While Reducing p50 Latency*. Anyscale. https://www.anyscale.com/blog/continuous-batching-llm-inference Accessed April 13, 2025.

[8] DeepSeek. 2025. Models & Pricing. https://platform.deepseek.com/api-docs/en/models-and-pricing/ Pricing includes standard/discount tiers and cache hit/miss rates for input tokens..

[9] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and simulation of a fair queueing algorithm. In *Symposium Proceedings on Communications Architectures & Protocols* (Austin, Texas, USA) *(SIGCOMM '89)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/75246.75248

[10] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and simulation of a fair queueing algorithm. In *Symposium Proceedings on Communications Architectures & Protocols* (Austin, Texas, USA) *(SIGCOMM '89)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/75246.75248

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[12] Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova Dassarma, Dawn Drain, Nelson Elhage, Sheer El Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Scott Johnston, Andy Jones, Nicholas Joseph, Jackson Kernian, Shauna Kravec, Ben Mann, Neel Nanda, Kamal Ndousse, Catherine Olsson, Daniela Amodei, Tom Brown, Jared Kaplan, Sam McCandlish, Christopher Olah, Dario Amodei, and Jack Clark. 2022. Predictability and Surprise in Large Generative Models. In *2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*. ACM, 1747–1764. https://doi.org/10.1145/3531146.3533229

[13] S. Jamaloddin Golestani. 1994. A self-clocked fair queueing scheme for broadband applications. *Proceedings of INFOCOM '94 Conference on Computer Communications* (1994), 636–646 vol.2. https://api.semanticscholar.org/CorpusID:1030307

[14] Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. 2025. Past-Future Scheduler for LLM Serving under SLA Guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 798–813. https://doi.org/10.1145/3676641.3716011

[15] Google. 2025. Google AI Pricing. https://ai.google.dev/pricing Covers Gemini models via Google AI Studio..

[16] Pawan Goyal, Harrick M. Vin, and Haichen Chen. 1996. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Palo Alto, California, USA) *(SIGCOMM '96)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/248156.248171

[17] Ozgur Guldogan, Jackson Kunde, Kangwook Lee, and Ramtin Pedarsani. 2024. Multi-Bin Batching for Increasing LLM Inference Throughput. arXiv:2412.04504 [cs.CL] https://arxiv.org/abs/2412.04504

[18] Krystal Hu. 2023. ChatGPT sets record for fastest-growing user base - analyst note. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/ Accessed April 13, 2025.

[19] Saki Imai, Rina Nakazawa, Marcelo Amaral, Sunyanan Choochotkaew, and Tatsuhiro Chiba. 2024. Predicting LLM Inference Latency: A Roofline-Driven ML Method. https://mlforsystems.org/assets/papers/neurips2024/paper28.pdf. Presented at NeurIPS 2024.

[20] Wei Jin, Jeffrey S. Chase, and Jasleen Kaur. 2004. Interposed proportional sharing for a storage service utility. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA) *(SIGMETRICS '04/Performance '04)*. Association for Computing Machinery, New York, NY, USA, 37–48. https://doi.org/10.1145/1005686.1005694

[21] Dara Kerr. 2025. DeepSeek hit with 'large-scale' cyber-attack after AI chatbot tops app stores. https://www.theguardian.com/technology/2025/jan/27/deepseek-cyberattack-ai. https://www.theguardian.com/technology/2025/jan/27/deepseek-cyberattack-ai Accessed April 13, 2025.

[22] Jacek Kobus and Rafał Szklarski. 2009. *Completely Fair Scheduler and its Tuning*. Technical Report. Nicolaus Copernicus University. http://fizyka.umk.pl/~jkob/prace-mag/cfs-tuning.pdf Based on R. Szklarski's M.S. thesis, Toruń.

[23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. https://doi.org/10.1145/3600006.3613165

[24] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. 2024. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 929–945. https://www.usenix.org/conference/osdi24/presentation/lin-chaofan

[25] John Nagle. 1987. On Packet Switches with Infinite Storage. *IEEE Transactions on Communications* 35, 4 (1987), 435–438. https://doi.org/10.1109/TCOM.1987.1096810

[26] OpenAI. 2024. *Rate Limits Guide*. OpenAI. https://platform.openai.com/docs/guides/rate-limits?context=tier-free Accessed April 13,

2025.

[27] OpenAI. 2025. Pricing. https://openai.com/pricing

[28] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2025. An Empirical Study of the Non-Determinism of ChatGPT in Code Generation. *ACM Transactions on Software Engineering and Methodology* 34, 2 (Jan. 2025), 1–28. https://doi.org/10.1145/3697010

[29] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG] https://arxiv.org/abs/2211.05102

[30] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. 2024. Power-aware Deep Learning Model Serving with μ-Serve. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 75–93. https://www.usenix.org/conference/atc24/presentation/qiu

[31] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. 2024. Efficient Interactive LLM Serving with Proxy Model-based Sequence Length Prediction. arXiv:2404.08509 [cs.DC] https://arxiv.org/abs/2404.08509

[32] Alec Radford and Karthik Narasimhan. 2018. Improving Language Understanding by Generative Pre-Training. https://api.semanticscholar.org/CorpusID:49313245

[33] Amon Rapp, Chiara Di Lodovico, and Luigi Di Caro. 2025. How do people react to ChatGPT's unpredictable behavior? Anthropomorphism, uncanniness, and fear of AI: A qualitative study on individuals' perceptions and understandings of LLMs' nonsensical hallucinations. *International Journal of Human-Computer Studies* 198 (2025), 103471. https://doi.org/10.1016/j.ijhcs.2025.103471

[34] Pol G. Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll. Berral. 2025. Mind the Memory Gap: Unveiling GPU Bottlenecks in Large-Batch LLM Inference. arXiv:2503.08311 [cs.DC] https://arxiv.org/abs/2503.08311

[35] James Requeima, John F Bronskill, Dami Choi, Richard E. Turner, and David Duvenaud. 2024. LLM Processes: Numerical Predictive Distributions Conditioned on Natural Language. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=HShs7q1Njh

[36] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are Emergent Abilities of Large Language Models a Mirage? arXiv:2304.15004 [cs.AI] https://arxiv.org/abs/2304.15004

[37] Karthick Panner Selvam. 2025. Why Large Language Models Fail at Precision Regression. https://karthick.ai/blog/2025/LLM-Regression/

[38] ShareGPT. 2023. ShareGPT: Share Your Wildest ChatGPT Conversations. https://sharegpt.com Deprecated. Please use OpenAI's built-in sharing instead. Thanks to everyone who used ShareGPT to share over 438,000 conversations..

[39] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in Serving Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 965–988. https://www.usenix.org/conference/osdi24/presentation/sheng

[40] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1348–1362. https://doi.org/10.1109/HPCA61900.2025.00102

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215 [cs.CL] https://arxiv.org/abs/1409.3215

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[43] Peiran Wang, Linjie Tong, Jiaxiang Liu, and Zuozhu Liu. 2025. Fair-MoE: Fairness-Oriented Mixture of Experts in Vision-Language Models. arXiv:2502.06094 [cs.CV] https://arxiv.org/abs/2502.06094

[44] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 640–654. https://doi.org/10.1145/3694715.3695948

[45] Charlene Yang, Yunsong Wang, Steven Farrell, Thorsten Kurth, and Samuel Williams. 2020. Hierarchical Roofline Performance Analysis for Deep Learning Applications. arXiv:2009.05257 [cs.DC] https://arxiv.org/abs/2009.05257

[46] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[47] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. 2024. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. arXiv:2309.11998 [cs.CL] https://arxiv.org/abs/2309.11998

[48] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 62557–62583. https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf

[49] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670 [cs.DC] https://arxiv.org/abs/2401.09670

A Supplementary Synthetic Workload Evaluations

This appendix provides supplementary results from synthetic workload experiments, illustrating Equinox's behavior under specific conditions complementary to those presented in subsection 7.2.

Constant Overload Scenario. We configure a scenario with constant, extreme overload to test fairness under high contention. Two clients send requests deterministically: Client 1 at a high rate of 20 req/s for short requests (input lengths of 20 and output lengths of 180), and Client 2 at a lower rate of 2 req/s but for very long requests (input lengths of 200 and output lengths of 1800). Both client demands exceed system capacity.

Overall performance. Figure 17 demonstrates that under constant overload, Equinox achieves dual advantages: it maintains the same fairness level as VTC while delivering enhanced service rates. Both schedulers successfully enforce fairness between clients with vastly different request rates and service demands, in contrast to FCFS which fails to provide equitable allocation in this high-contention regime. Notably, Equinox accomplishes this fairness guarantee while surpassing VTC in total throughput efficiency.

Breakdown. As evidenced by Figure 17d, Equinox matches VTC's fairness characteristics through their equivalently small and bounded cumulative service differences. The service rate parity between clients (Figure 17c) further confirms that Equinox maintains VTC-caliber fairness while achieving service rate improvements - its per-client allocations remain balanced yet attain higher absolute values compared to VTC. Crucially, as shown in Figure 17a, Equinox elevates the total service rate beyond VTC's baseline while sustaining comparable GPU utilization (Figure 17b), demonstrating its unique ability to optimize throughput without compromising fairness.

Dynamic Load Increase Scenario. This experiment evaluates Equinox's adaptability to dynamic changes in workload intensity. Client 1 maintains a constant request rate of 1 req/s for input lengths of 100 and output lengths of 400. Client 2 initially sends requests at the same input lengths and output lengths but increases its rate fourfold to 4 req/s midway through the experiment, significantly elevating the overall system load.

Overall performance. Equinox demonstrates effective adaptation to the increasing load while upholding fairness principles, as shown in Figure 18. As Client 2 becomes more demanding, Equinox adjusts resource allocation dynamically without allowing this client to monopolize service.

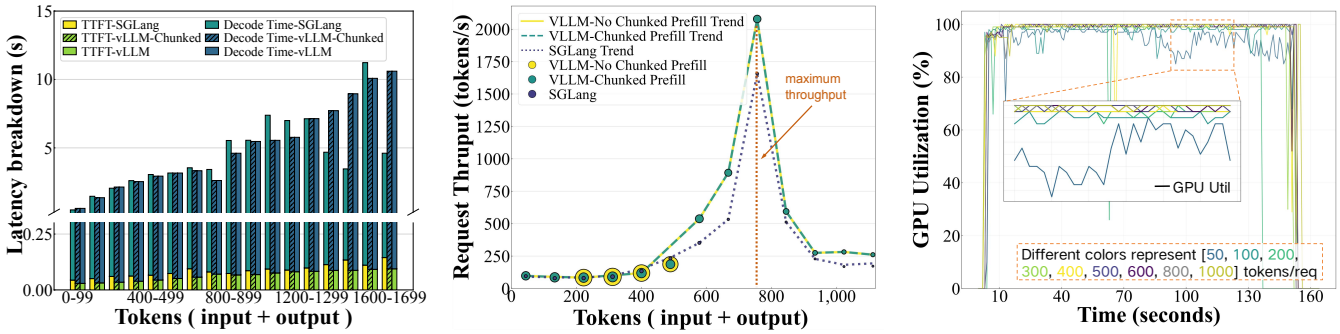
Breakdown. The sharp increase in Client 2's request rate is visible in Figure 18a. In response, Equinox dynamically recalibrates the per-client service rates (Figure 18c), ensuring Client 1 continues to receive its fair share despite Client 2's increased demand. This adaptation is accompanied by an expected rise in overall response times (Figure 18d) as system load intensifies. Correspondingly, GPU utilization increases (Figure 18b), reflecting the heightened demand.

B Supplementary Real-world Workload Evaluations

This appendix contains supplementary results from real-world workload evaluations conducted using S-LoRA and vLLM systems, complementing the main results presented in subsection 7.3.

S-LoRA with LMSYS Dataset. We conducted experiments using S-LoRA [3] under a workload constructed from the trace logs of LMSYS Chatbot Arena. This scenario involved 27 distinct clients submitting requests based on patterns observed in these real-world interactions.

Workload Dynamics and System Response. Figure 19 illustrates key characteristics of this dynamic workload and the system's response within S-LoRA. The per-client request rates (Figure 19a) exhibit considerable variation over time, leading to fluctuations in the total instantaneous request rate offered to the system (Figure 19b). Consequently, the observed per-client response times (following VTC, we select 13th, 14th and 26th, 27th clients based on the number of requests they send from minimum to maximum) (Figure 19c) also vary, reflecting the interplay between the dynamic request arrivals, their resource requirements, and the scheduling decisions made by the system. While this setup is used for the cross-system fairness comparison (Figure 13), these specifically show the workload dynamics.

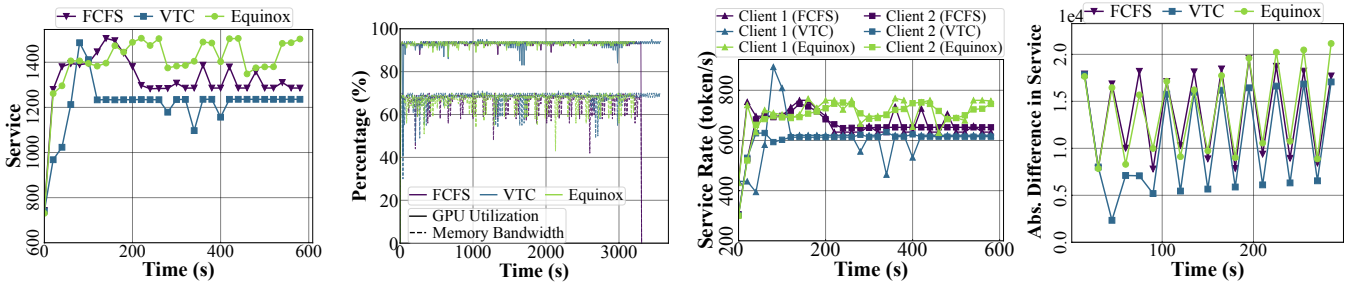


(a) vLLM and SGLang exhibit similar latency patterns.

(b) vLLM and SGLang show non-linear throughput gains across token ranges.

(c) vLLM and SGLang display step-like GPU resource usage patterns.

Figure 16. Results show metric independence from token count under varied scheduling and serving systems. Analysis reveals **non-linear**, inconsistent latency, throughput, and GPU utilization, contradicting token-level fairness assumptions. Crucially, even with optimizations like chunked prefill, per-token metrics unreliably predict performance across adjacent token ranges. Experiments utilized an NVIDIA A100 GPU with ShareGPT (a-b), and controlled synthetic workloads (c).



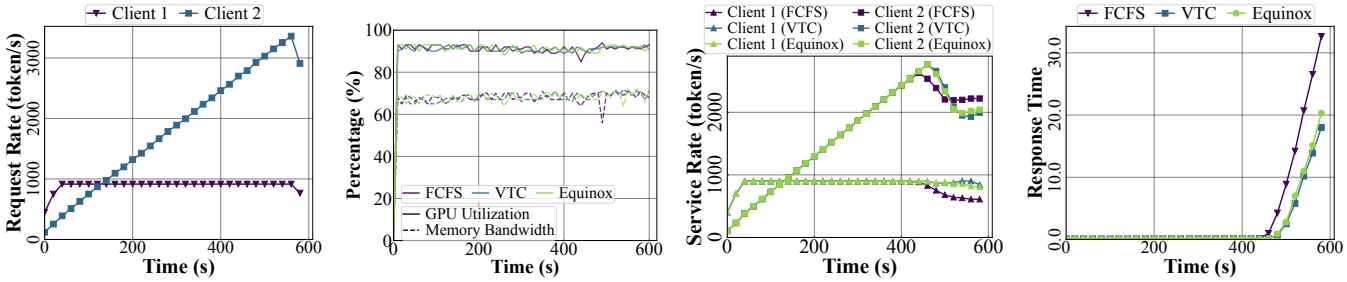
(a) Total Service Rate

(b) GPU Util. & Bandwidth

(c) Per-client Service Rate

(d) Service Difference

Figure 17. Constant overload scenario.



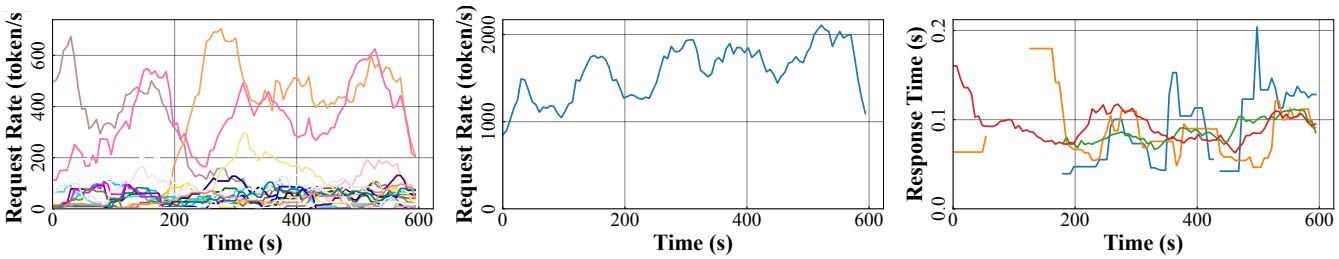
(a) Client Request Rate

(b) GPU Util. & Bandwidth

(c) Per-client Service Rate

(d) Total Response Time

Figure 18. Increasing load scenario.



(a) Per-client Request Rate **(b)** Total Request Rate **(c)** Selected-client Response Time

Figure 19. Real workload dynamics from LMSYS trace evaluated in S-LoRA, involving 27 clients. Shows variability in client request rates and resulting system response times.