Sensitivity Analysis of MoE Kernels under All-to-All Routing

Haoyu Yan, Yaxi Zeng, Shengjing Zhang, Amber Deng

December 2025

1 Introduction

Mixture-of-Experts (MoE) models have emerged as a powerful approach for scaling deep neural networks by replacing standard feed-forward layers with a set of parallel expert networks [1, 2]. A lightweight gating function selects a sparse subset of experts for each input, enabling *sparse conditional computation*—activating only a few experts at a time—which significantly reduces computation while maintaining high model capacity. This architecture allows different experts to specialize in diverse input patterns, improving both generalization and scalability.

Despite these advantages, MoE inference introduces significant system-level challenges. In particular, uneven token-to-expert assignment can lead to expert load imbalance, resulting in inefficient execution and increased latency. Prior work has shown that routing imbalance and expert parallelism play a critical role in determining MoE performance, motivating careful analysis of routing behavior and load distribution [3].

Recently, the vLLM inference engine introduced an All-to-All (All2All) routing mechanism that enables efficient token dispatch across devices using collective communication [4]. While All2All routing improves scalability for MoE inference, its performance sensitivity to workload characteristics and expert load imbalance remains insufficiently understood.

In this paper, we present a sensitivity analysis of MoE inference under All2All routing. We focus on three key dimensions:

- **Batch and sequence length sensitivity**: How inference-time parameters such as max-num-seqs and max-num-batched-tokens affect latency and throughput.
- **Expert-Parallel Load Balancing (EPLB)**: Whether enabling EPLB provides measurable performance improvements under fixed routing and workload settings.
- **Load imbalance sensitivity**: How both *expected* (input-induced) and *observed* (runtime-measured) expert load imbalance correlate with performance metrics, particularly time-to-first-token (TTFT).

In this work, we present a systematic sensitivity analysis of MoE inference performance under All-to-All routing. We first characterize the impact of inference-time configuration parameters, including `max-num-seqs` and `max-num-batched-tokens`, on latency and throughput by comparing a baseline MoE execution with the default All2All routing implementation in vLLM.

We then study routing- and mechanism-level effects by evaluating Expert-Parallel Load Balancing (EPLB) under otherwise identical configurations, isolating whether explicit post-routing rebalancing provides measurable performance benefits beyond standard All2All routing.

Finally, we isolate the effect of expert load imbalance itself by constructing datasets with controlled workload characteristics. By varying input-induced imbalance scores while keeping all system parameters fixed, we analyze how both expected and observed imbalance correlate with inference performance. Together, these experiments clarify how system parameters, routing mechanisms, and workload-induced imbalance interact to shape MoE inference behavior, providing practical guidance for deploying MoE models with predictable latency and throughput.

2 Background

Mixture-of-Experts (MoE) architectures have become an effective paradigm for scaling large language models (LLMs) through conditional computation. Instead of activating all model parameters for every input, MoE models route each token to a small subset of specialized expert networks, significantly reducing computation while maintaining high model capacity.

A standard MoE layer consists of a shared input representation, a gating (routing) mechanism, and a set of E expert subnetworks. For each input token, the router selects $k \ll E$ experts—most commonly via top-k gating—and forwards the token only to those experts. This sparse execution model enables efficient training and inference at scale, particularly when combined with appropriate parallelization strategies.

In distributed deployments, *All-to-All* routing is widely adopted to dispatch tokens to experts residing on different devices. By allowing tokens to be dynamically exchanged across GPUs, All-to-All routing improves overall expert utilization and enables expert parallelism. However, this flexibility also makes system performance highly sensitive to expert load imbalance, as skewed token-to-expert assignments can create stragglers and synchronization bottlenecks.

Recent studies have shown that MoE performance is strongly affected by expert load imbalance. For example, MegaBlocks [5] and FlexMoE [6] demonstrate that uneven expert activation can lead to warp divergence and hotspot experts, significantly degrading throughput. Tutel [3] further highlights how expert count and routing granularity influence batched GEMM efficiency and overall system behavior. Similarly, FastMoE [7] and DeepSpeed-MoE [8] report substantial variability in latency and throughput as a function of batch size, token distribution, and routing variance.

To mitigate routing-induced imbalance, several systems introduce explicit *expert-level load balancing* mechanisms, such as Expert-Parallel Load Balancing (EPLB), which attempt to redistribute expert workloads to achieve more uniform utilization across devices. While EPLB can improve average load balance, it may also introduce additional communication, synchronization, or scheduling overhead, making its net performance impact highly dependent on workload characteristics and system design.

As a result, it remains unclear whether enabling EPLB consistently improves inference performance under identical model and hardware configurations, or whether its overhead can outweigh its benefits in certain regimes. Moreover, the relationship between *input-induced imbalance*—arising from token repetition, domain concentration, or low entropy—and *runtime-observed expert imbalance* has not been systematically examined in practical inference settings.

Motivated by these observations, we conduct a systematic investigation into the sensitivity of MoE inference performance under All-to-All routing. We first compare a baseline MoE execution with vLLM's default All-to-All routing to characterize how inference-time configuration parameters—specifically batch size and sequence length—affect key performance metrics, including time-to-first-token (TTFT), time per output token (TPOT), and output throughput.

We then analyze the role of expert load imbalance by controlling workload characteristics while holding all system parameters fixed, allowing us to study how imbalance correlates with inference performance. Finally, we evaluate Expert-Parallel Load Balancing (EPLB) under otherwise identical configurations to isolate whether explicit post-routing rebalancing provides measurable performance benefits beyond standard All-to-All routing.

3 Description of Tools

In this work, we focus on understanding the sensitivity of MoE inference performance under controlled workload and routing conditions. To ensure that observed performance trends are attributable to routing behavior and workload characteristics—rather than model- or engine-specific artifacts—we carefully select a representative MoE model, a configurable inference engine, and a set of supporting tools that together enable reproducible execution, system-level inspection, and consistent result analysis.

3.1 Model: Mixtral-8x7B

All experiments are conducted using `mistralai/Mixtral-8x7B-v0.1`, a large-scale sparse Mixture-of-Experts (MoE) language model. Mixtral contains 46.7 billion total parameters, while activating only 12.9 billion parameters per token through sparse expert routing. Each MoE layer consists of 8 experts, with top-k routing selecting $k = 2$ experts per token.

Mixtral is well suited for our study for several reasons. First, its sparse routing mechanism naturally exposes expert load imbalance effects, making it an appropriate platform for analyzing how imbalance manifests during inference. Second, the model operates at a scale where batching efficiency, communication overhead, and expert dispatch costs are non-trivial, allowing system-level performance effects to become observable. Finally, Mixtral supports long context lengths (up to 32k tokens), enabling us to explore a wide range of batch size and sequence length configurations under realistic inference settings.

Importantly, we keep the model architecture fixed across all experiments. This design allows us to isolate the impact of workload characteristics, routing strategies, and load balancing mechanisms without confounding changes in model structure.

3.2 Inference Engine: vLLM

We use vLLM as the inference engine for all experiments. vLLM provides an efficient and configurable serving framework for large language models, with native support for MoE models and All-to-All (All2All) routing across devices. Its design exposes fine-grained control over inference-time parameters such as batch size, maximum number of batched tokens, and routing configuration, which are central to our sensitivity analysis.

A key advantage of vLLM is its ability to support both baseline MoE execution and optimized All2All routing within the same engine. This enables controlled comparisons between routing strategies while keeping scheduling, memory management, and tokenization behavior consistent. As a result, differences in latency and throughput can be more confidently attributed to routing behavior and workload effects rather than engine-level variation.

3.3 System Profiling Tool: NVIDIA Nsight Systems

To gain visibility into the system-level behavior of MoE inference, we use NVIDIA Nsight Systems (nsys) as our primary profiling tool. Nsight Systems enables detailed tracing of GPU execution, including kernel launches, CUDA stream scheduling, synchronization points, and communication events.

We use nsys to profile representative inference runs under different routing configurations, including baseline MoE execution and All2All routing. These traces allow us to qualitatively inspect execution patterns and validate whether observed performance trends align with changes in computation, communication, or synchronization behavior. Profiling is used as a diagnostic and validation tool rather than as a source of direct quantitative metrics, and all profiling runs are conducted under identical hardware and software configurations.

3.4 Experiment Execution and Data Logging

All experiments are executed using scripted pipelines to ensure reproducibility and consistent configuration across runs. Inference parameters such as batch size, maximum number of batched tokens, routing mode, and load balancing settings are explicitly specified and varied according to the experimental design.

For each inference run, we record detailed runtime statistics, including latency metrics (e.g., time-to-first-token and time per output token), throughput metrics, and routing-related statistics when enabled. These measurements are logged in structured JSON format, enabling systematic post-processing and comparison across configurations. This design minimizes manual intervention and reduces variability introduced by ad hoc measurements.

3.5 Result Aggregation and Visualization

We use custom Python-based analysis scripts to aggregate experimental results and generate all figures presented in this paper. Collected logs are parsed, filtered, and aligned across configurations to ensure consistent comparison between routing strategies, workload settings, and imbalance levels.

Visualization is performed using a unified data processing pipeline built on standard Python libraries, with figures generated using `matplotlib`. This pipeline enables clear and consistent presentation of relationships between workload characteristics, routing configuration, and performance metrics. All plots are produced from the same aggregated data sources, ensuring reproducibility and consistency across experiments.

Summary

Together, these tools provide a controlled and expressive experimental platform for studying MoE inference behavior. By combining a realistic MoE model, a configurable inference engine, system-level profiling, and structured experiment execution and analysis, we are able to isolate and analyze the effects of routing strategy, workload-induced imbalance, and explicit load balancing mechanisms on latency and throughput under All-to-All routing.

4 Metrics

We evaluate MoE inference behavior using a combination of performance metrics and expert load imbalance metrics. Performance metrics characterize latency and throughput under different routing and workload configurations, while imbalance metrics quantify the degree of skew in expert utilization.

4.1 Performance Metrics

Time To First Token (TTFT)

Time To First Token (TTFT) measures the latency from submitting an inference request to the generation of the first output token. TTFT captures prompt processing overhead, routing decisions, and initial execution costs, and serves as a primary indicator of inference responsiveness.

Time Per Output Token (TPOT)

Time Per Output Token (TPOT) measures the average time required to generate each output token after the first token has been produced. TPOT reflects steady-state generation efficiency and is commonly used to assess sustained inference performance.

Output Throughput

Output throughput measures the end-to-end rate of token generation, reported as the number of output tokens generated per second. This metric summarizes overall system efficiency and captures the combined effects of computation, communication, and scheduling.

4.2 Expert Load Imbalance Metrics

To analyze the impact of expert load skew on inference performance, we quantify expert load imbalance using two complementary measures: an observed (runtime) imbalance metric and an expected (input-induced) imbalance metric.

Observed Expert Load Imbalance

Observed expert load imbalance measures how unevenly tokens are distributed across experts during inference execution. Let x_i denote the number of tokens assigned to expert i in a given MoE layer. We quantify imbalance using the coefficient of variation (CV):

$$\mathrm{CV} = \frac{\sigma(\{x_i\})}{\mu(\{x_i\})},$$

where σ and μ denote the standard deviation and mean of expert token loads, respectively. Higher CV values indicate greater skew in expert utilization.

Expected (Input-Induced) Expert Load Imbalance

Expected expert load imbalance estimates the degree of imbalance implied by the input workload prior to inference execution. This metric is derived from token-level statistics of the input prompts, such as lexical repetition, vocabulary diversity, and distributional concentration.

The expected imbalance score is normalized to the range $[0, 1]$, where higher values indicate greater concentration and lower effective diversity in the input token distribution. This measure allows us to characterize workload-induced imbalance independently of routing strategy or system configuration.

Metric Usage

In our analysis, TTFT, TPOT, and output throughput are used to evaluate inference performance under different routing strategies and system parameters. Observed and expected imbalance metrics are used to study how workload characteristics translate into runtime expert load skew, and how such imbalance correlates with latency and throughput.

5 Implementation Details

5.1 Experiment 1: Impact of Inference-Time Configuration Parameters

Modern MoE inference systems expose a set of server-side configuration parameters that directly determine batching behavior, scheduling decisions, and execution scale. In this experiment, we study how such inference-time parameters affect latency and throughput under different routing strategies. The goal is to characterize system-level performance trends before introducing workload-induced expert imbalance or explicit load balancing mechanisms.

Importantly, these parameters are controlled entirely at the *server side*. The client is used only to generate sufficient workload to saturate the server and does not influence batching or scheduling behavior. This design ensures that observed performance differences can be attributed to inference configuration and routing strategy rather than request arrival patterns.

Configuration Parameters. We focus on two inference-time parameters provided by vLLM:

- **Maximum number of sequences** (`--max-num-seqs`): the upper bound on the number of concurrent requests admitted by the scheduler.
- **Maximum number of batched tokens** (`--max-num-batched-tokens`): the maximum total number of tokens processed in a single forward pass.

We fix the maximum model length (`--max-model-len`) to 4096 across all experiments, as it controls attention memory layout and KV-cache capacity and is not a variable of interest in this study.

Experimental Setup. We compare two routing configurations under identical model, hardware, and workload conditions:

- **Baseline MoE execution**, which uses standard MoE routing without All-to-All communication.
- **All2All routing (vLLM default)**, which distributes tokens to experts across devices using collective communication.

For each routing configuration, we perform a grid sweep over inference-time parameters, varying:

- `max-num-seqs` $\in \{4, 8, 16, 32\}$
- `max-num-batched-tokens` $\in \{2048, 4096, 8192, 16384\}$

All other settings are held constant.

The inference server is launched with the specified configuration, while a separate client process issues a fixed number of requests using the same dataset. The client does not control batching behavior; its role is solely to ensure that the server operates near its configured capacity.

Metrics. For each configuration, we measure time-to-first-token (TTFT), time per output token (TPOT), request throughput, and output throughput. Each configuration is evaluated across multiple runs to reduce variance.

Purpose of the Experiment. This experiment isolates the effect of inference-time configuration parameters on MoE performance. By fixing workload generation and varying only server-side execution limits, we establish a baseline understanding of how batch capacity and concurrency interact with routing strategy to shape latency and throughput. These results provide context for later experiments that focus on workload-induced expert imbalance and explicit load balancing.

5.2 Experiment 2: Effect of Expert-Parallel Load Balancing (EPLB)

This experiment examines whether enabling Expert-Parallel Load Balancing (EPLB) provides additional performance benefits beyond standard All-to-All (All2All) routing during MoE inference. EPLB is a post-routing mechanism that attempts to reduce peak expert load by redistributing token assignments across experts, while preserving the original top-k routing decisions.

To isolate the effect of EPLB, we evaluate two configurations under an identical and representative inference setting:

- **All2All without EPLB**: vLLM default All2All routing with EPLB disabled.
- **All2All with EPLB**: the same routing configuration with EPLB enabled.

All other factors are held constant, including the model architecture (Mixtral-8x7B), batch size (`max-num-seqs = 32`), maximum number of batched tokens (`max-num-batched-tokens = 8192`), sequence length, and hardware configuration. This controlled setup allows us to attribute any observed performance differences specifically to the presence of EPLB.

We evaluate both configurations using GPU execution traces collected via NVIDIA Nsight Systems (`nsys`), extracting kernel-level timing statistics for the `fused_moe_kernel` and NCCL communication primitives. Key metrics include total kernel time, average and maximum duration, standard deviation, coefficient of variation (CV), and max/min ratio for load balance assessment, as well as NCCL and AllReduce overhead to quantify communication costs. By comparing these trace-derived metrics under identical routing and workload conditions, we assess whether reducing expert load skew through EPLB translates into measurable improvements in MoE kernel efficiency or is offset by increased communication overhead.

This experiment is designed as a focused ablation rather than a full parameter sweep. Its goal is to determine whether explicit post-routing load balancing offers additional benefits once All2All routing is already in place, under a fixed and practically relevant operating point.

5.3 Experiment 3: Load Imbalance Sensitivity

This experiment isolates how expert load imbalance affects MoE inference performance under All-to-All (All2All) routing. Unlike Experiment 1, where we vary inference-time configuration parameters (e.g., `max-num-seqs` and `max-num-batched-tokens`), here we keep all system-level settings fixed and vary only the *workload.* Concretely, we systematically construct prompt datasets with different lexical and semantic characteristics to induce different levels of expert load imbalance, and we measure how imbalance correlates with latency and throughput.

5.3.1 Workload Construction (Synthetic Datasets)

To control expert load imbalance independently of execution scale, we generate a collection of synthetic inference datasets in JSONL format. Each dataset contains a fixed number of prompts (N prompts), while batch size, maximum number of batched tokens, model length, and routing configuration are held constant during inference. This design ensures that observed performance differences are primarily attributable to imbalance rather than changes in batch or sequence configuration.

We organize datasets into several families, each targeting a distinct source of imbalance:

- **Repetition family.** These datasets control lexical redundancy by varying the degree of token- and phrase-level repetition, ranging from extreme cases with nearly identical prompts to more natural inputs with minimal repetition.
- **Semantic family.** These datasets control domain concentration by sampling prompts from one or multiple semantic domains (e.g., programming, science, cooking). We construct workloads spanning a spectrum from single-domain inputs to multi-domain mixtures and uniformly diverse prompts, enabling systematic variation of semantic diversity while keeping prompt length comparable.
- **Structure family.** These datasets control syntactic regularity by using identical or templated prompt structures with varying content. Despite surface-level token variation, repeated structural patterns can still concentrate routing behavior and induce imbalance.

Across all families, prompt sampling is randomized with a fixed seed to ensure reproducibility. In addition to discrete dataset families, we generate workloads that continuously vary domain concentration and include randomized combinations to avoid clustering around a narrow regime. We verify that the resulting dataset suite spans a broad range of induced imbalance scores, providing coverage from low-imbalance to high-imbalance workloads.

5.3.2 Imbalance Quantification

We quantify imbalance using two complementary measures: an *expected* (input-induced) imbalance score computed *before* running inference, and an *observed* (runtime-measured) imbalance score computed *during* inference from actual token-to-expert assignments.

Expected (Input-Driven) Imbalance Score. To estimate expert load imbalance *prior* to running MoE inference, we implement an input-driven scoring model, `ImbalancePredictor`, which produces a scalar score $s \in [0, 1]$ for a given dataset. A larger score indicates that the input workload is more likely to induce skewed expert utilization. Importantly, this score is computed using only prompt text statistics and does not require access to internal routing signals.

Tokenization and feature extraction. Given a set of prompts, we tokenize all prompts with the Mixtral tokenizer and aggregate all token IDs into a single token stream $\mathcal{T}$. Let $\mathcal{V}$ denote the set of unique tokens in $\mathcal{T}$, and let p_i be the empirical frequency of token i. We extract the following distributional features:

- **Token repetition rate:**

$$r = 1 - \frac{|\mathcal{V}|}{|\mathcal{T}|},$$

  capturing lexical redundancy (higher r implies more repetition).

- **Vocabulary diversity:** $\frac{|\mathcal{V}|}{|\mathcal{T}|}$.
- **Top-k token concentration (with $k = 10$):** the fraction of tokens covered by the 10 most frequent tokens.
- **Unigram entropy:**
$$H = -\sum_i p_i \log_2 p_i,$$
and its **normalized entropy** $\hat{H} = \frac{H}{\log_2(\max(|\mathcal{V}|,2))}$.
- **Gini coefficient:** measuring inequality of the token frequency distribution.
- **Bigram concentration:** the fraction of bigrams covered by the 10 most frequent bigrams.
- **Semantic similarity (optional):** mean pairwise cosine similarity between TF-IDF vectors of prompts; raw similarity in $[-1, 1]$ is linearly rescaled to $[0, 1]$.

Score aggregation. Each feature is scaled to $[0, 1]$ and aggregated via a weighted sum:

$$s = \frac{1}{Z}\sum_{j=1}^{n} w_j f_j, \qquad Z = \sum_{j=1}^{n} w_j,$$

where f_j is the j-th feature value and w_j is its weight. We use the following weights (renormalized over the features that are enabled):

- $w_{\text{repetition}} = 0.25$
- $w_{\text{top10}} = 0.20$
- $w_{\text{inv-vocab}} = 0.15$
- $w_{\text{inv-entropy}} = 0.10$
- $w_{\text{gini}} = 0.10$
- $w_{\text{bigram}} = 0.10$
- $w_{\text{semantic}} = 0.10$ (if enabled)

We then clamp the final score to $[0, 1]$. This expected imbalance score serves as an input-side proxy for routing skew: workloads with higher repetition, higher concentration, lower entropy, and higher inequality are assigned higher predicted imbalance.

Observed (Runtime-Measured) Imbalance. To measure imbalance experienced by the model during inference, we instrument the MoE routing logic in vLLM to log expert-level token assignment statistics for Mixtral. During inference, we record the number of tokens routed to each expert at every MoE layer and routing step.

Let x_i denote the total number of tokens assigned to expert i within a given MoE layer over an inference run (aggregated across all routing steps). We measure expert load imbalance using the coefficient of variation (CV):

$$\text{CV} = \frac{\sigma(\{x_i\})}{\mu(\{x_i\})},$$

where $\sigma(\cdot)$ and $\mu(\cdot)$ are the standard deviation and mean of expert token loads, respectively. CV is scale-normalized, making it suitable for comparing imbalance across runs with the same execution configuration. For each dataset, we compute CV per MoE layer and then aggregate across layers (e.g., by averaging) to obtain a single observed imbalance score per run. This score represents the ground-truth runtime imbalance under a specific routing strategy and workload.

5.3.3 Execution Protocol and Metrics Collection

For each synthetic dataset, we run inference under two routing configurations:

- **Baseline MoE execution**, and
- **vLLM default All2All routing**.

All system-level parameters (batch size, maximum number of batched tokens, model length, decoding settings, and hardware configuration) are held constant across datasets and across routing configurations. This controlled protocol allows a direct comparison of how the *same workload* behaves under different routing implementations.

For every run, we collect standard inference performance metrics including time-to-first-token (TTFT), time per output token (TPOT), and output throughput. We then correlate each performance metric with both (i) the expected imbalance score and (ii) the observed CV-based imbalance score.

5.3.4 Analysis

We analyze imbalance sensitivity by plotting performance metrics as a function of imbalance, and by comparing trends under baseline versus All2All routing. In addition, we compare expected versus observed imbalance to evaluate how input-induced concentration translates into actual runtime expert skew. When generating scatter plots, we apply the same IQR-based outlier filtering rule (1.5×IQR) used throughout this work to remove extreme points on both the imbalance axis and the performance axis for cleaner visualization.

6 Results and Evaluation

In this section, we present the experimental results of our sensitivity analysis and evaluate how different system configurations, workload characteristics, and routing strategies affect MoE inference performance. We report latency and throughput metrics under controlled settings and analyze performance trends across experiments to highlight the impact of inference-time parameters, expert load imbalance, and explicit load balancing mechanisms.

6.1 Experiment 1: Impact of Inference-Time Configuration Parameters

This experiment studies how inference-time configuration parameters influence MoE inference performance under different routing implementations. We focus on two key parameters exposed by vLLM: `max-num-seqs`, which controls the number of concurrent sequences processed (batch-level concurrency), and `max-num-batched-tokens`, which sets an upper bound on the total number of tokens processed per batch. Together, these parameters determine the effective batching behavior during inference.

We compare a baseline MoE execution mode against vLLM's default fused All2All routing configuration, and evaluate performance using three metrics: **time-to-first-token (TTFT)**, **time per output token (TPOT)**, and **output throughput**.

TTFT Sensitivity. Figure 1 shows the effect of `max-num-seqs` (left) and `max-num-batched-tokens` (right) on TTFT under both routing configurations.

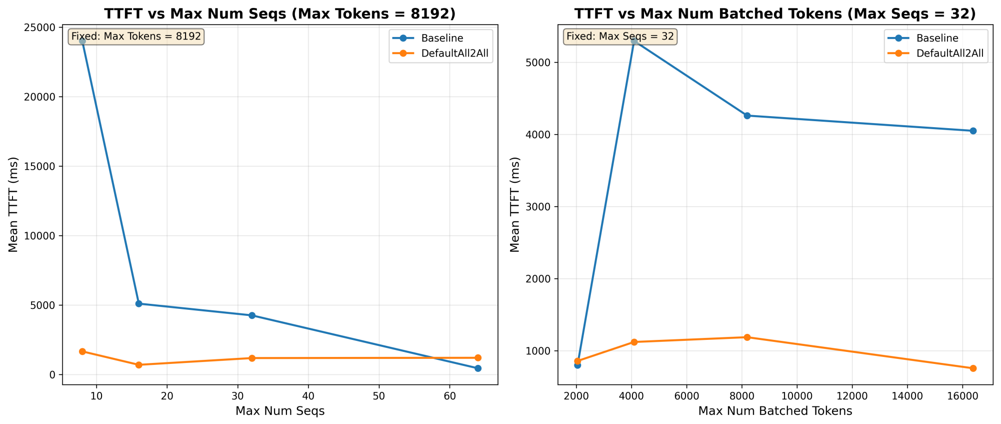


Figure 1: TTFT under baseline and All2All

As shown in Figure 1, the baseline configuration exhibits strong sensitivity to both inference-time parameters. Increasing `max-num-seqs` substantially reduces TTFT, indicating that higher sequence-level concurrency helps amortize fixed startup costs such as request scheduling, kernel launch overhead, and initial expert dispatch. With more concurrent sequences, these overheads are shared across requests, resulting in faster time-to-first-token.

In contrast, increasing `max-num-batched-tokens` leads to higher TTFT under the baseline configuration. This trend reflects the cost of processing larger token batches before any output can be produced: as the total token budget grows, the model must perform more prefill computation and expert routing work prior to emitting the first token, directly increasing startup latency. Together, these results highlight a trade-off in the baseline kernel between concurrency-driven amortization and token-volume-induced prefill overhead.

All2All routing demonstrates consistently lower and more stable TTFT across both dimensions. TTFT remains relatively insensitive to increases in either `max-num-seqs` or `max-num-batched-tokens`, suggesting that fused routing and parallel expert dispatch effectively reduce prefill overhead and mitigate startup latency growth under larger batch sizes and token budgets.

TPOT Sensitivity. Figure 2 evaluates how the same parameters affect TPOT, which reflects steady-state decoding efficiency.

Figure 2 illustrates the sensitivity of TPOT to inference-time configuration parameters. Under the baseline configuration, TPOT exhibits significant variability as `max-num-seqs` increases. In particular, we observe a sharp spike in TPOT at intermediate batch sizes, indicating unstable steady-state decoding performance. This non-monotonic behavior suggests that the baseline kernel is highly sensitive to batching boundaries and expert dispatch patterns.

In contrast, All2All routing achieves consistently lower and more stable TPOT across the entire parameter range. Varying either `max-num-seqs` or `max-num-batched-tokens` results in only minor changes in TPOT, demonstrating that fused All2All routing effectively amortizes per-token routing and communication overhead during decoding.

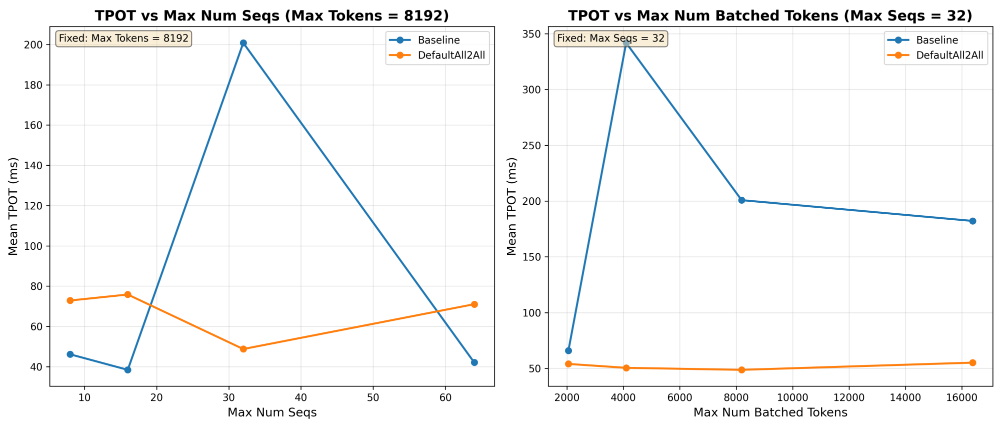


Figure 2: TPOT under baseline and All2All

Notably, increasing the token budget substantially degrades TPOT under the baseline configuration, while All2All routing remains largely unaffected. This highlights the improved scalability of All2All routing in steady-state inference and its ability to sustain efficient token generation under larger batching regimes.

Throughput Scaling. Figure 3 reports output throughput under the same parameter sweep.

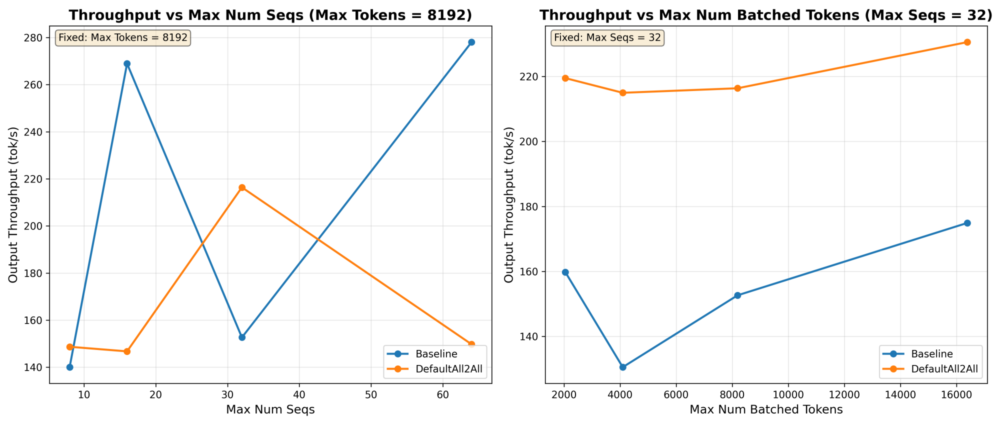


Figure 3: Output throughput under baseline and All2All

As shown in Figure 3, under the baseline configuration, throughput exhibits highly non-monotonic behavior with respect to both configuration parameters. Increasing `max-num-seqs` does not consistently improve throughput; instead, we observe sharp fluctuations and even performance degradation at intermediate batch sizes. A similar pattern emerges when increasing `max-num-batched-tokens`, where a larger token budget fails to translate into proportional throughput gains. These trends closely mirror the TPOT instability observed in Figure 2, indicating that the baseline kernel struggles to efficiently convert increased parallelism and token volume into sustained output throughput.

In contrast, All2All routing achieves smoother and more predictable throughput scaling along both dimensions. As `max-num-seqs` increases, throughput under All2All remains stable and avoids the sharp drops observed in the baseline case. Likewise, increasing `max-num-batched-tokens` leads to a steady rise in output throughput, demonstrating that larger token budgets are effectively amortized under All2All routing. Overall, these results show that All2All routing more reliably converts both sequence-level concurrency and token-level parallelism into sustained output throughput, highlighting its superior scalability under larger and more demanding batching configurations.

Summary. Overall, Experiment 1 shows that inference-time configuration parameters have a substantial impact on MoE performance, particularly under baseline execution. Fused All2All routing significantly reduces sensitivity to batching configuration, providing lower and more predictable TTFT, improved TPOT, and higher throughput across a wide range of inference settings.

6.2 Experiment 2: Expert-Parallel Load Balancing (EPLB)

To evaluate the effectiveness of Expert Parallel Load Balancing (EPLB) in mitigating load imbalance during MoE inference, we conducted a controlled experiment on NERSC's Perlmutter supercomputer. We deployed Mixtral-8x7B using vLLM's Expert Parallelism (EP=4) with the default All2All MoE kernel, running two separate inference sessions: one with EPLB enabled and one with EPLB disabled. Both configurations used identical serving parameters and processed the same input dataset to ensure a fair comparison. We collected GPU execution traces using NVIDIA Nsight Systems (`nsys`) and extracted kernel-level timing statistics from the trace files, focusing on the `fused_moe_kernel` and NCCL communication primitives to quantify the trade-off between load balance improvements and communication overhead.

Table 1: Performance comparison of Expert Parallel Load Balancing (EPLB) on Mixtral-8x7B with vLLM EP=4 on Perlmutter.

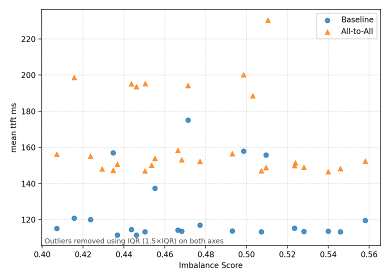
Metric	**EPLB OFF**	**EPLB ON**	**Change**	**Verdict**
MoE Kernel Performance				
fused_moe Total Time (ms)	102,664	108,419	+5.6%	Worse
fused_moe Avg Duration (μs)	383.68	398.68	+3.9%	Worse
fused_moe Max Duration (μs)	239,809	225,736	−5.9%	Better
fused_moe StdDev (μs)	2,472	2,446	−1.1%	Better
fused_moe CV (σ/μ)	6.44	6.14	−4.8%	Better
fused_moe Max/Min Ratio	92,519×	86,027×	−7.0%	Better
Communication Overhead				
NCCL Total Time (ms)	85,965	92,534	+7.6%	Worse
AllReduce Total Time (ms)	1,272	1,928	+51.5%	Worse

Table 1 presents the kernel-level performance comparison between EPLB-enabled and EPLB-disabled configurations. EPLB successfully improves load balance metrics: the coefficient of variation decreased by 4.8%, maximum kernel duration decreased by 5.9%, and the max/min ratio improved by 7.0%. However, these gains come at the cost of increased communication overhead, with total NCCL time increasing by 7.6% and AllReduce time increasing by 51.5%. The net effect is a 5.6% increase in total MoE kernel time, indicating that for this workload configuration, the communication overhead introduced by EPLB exceeds the computational savings from improved load balance. Notably, the extreme max/min ratios observed in both configurations (>86,000×) are primarily attributable to the inherent variance between prefill and decode phases rather than expert-level imbalance, suggesting that EPLB may provide greater benefit for workloads exhibiting more pronounced natural expert load skew.

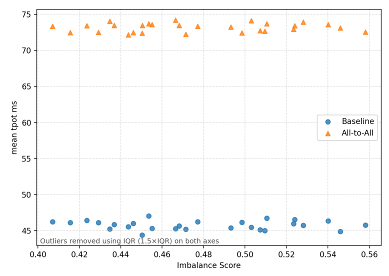
6.3 Experiment 3: Load Imbalance Sensitivity

This experiment examines how expert load imbalance influences MoE inference performance under fixed system settings. We consider two notions of imbalance: an *induced* imbalance score derived from input-token statistics, and an *observed* (real) imbalance rate measured from runtime token-to-expert dispatch. We evaluate how each form of imbalance relates to TTFT, TPOT, and output throughput under both the baseline and All2All configurations.

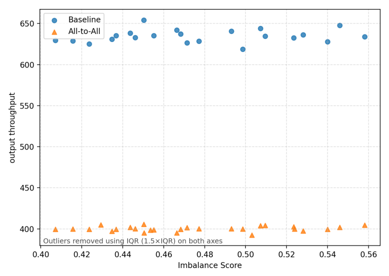
Outlier handling. For clarity in visualization, we remove extreme outliers using an IQR-based rule (1.5×IQR) applied to both the imbalance axis and the performance metric axis. This filtering suppresses rare transient effects (e.g., warm-up or scheduling noise) without affecting overall trends, and is applied consistently across all metrics and configurations.



(a) TTFT vs. Imbalance Score



(b) TPOT vs. Imbalance Score



(c) Throughput vs. Imbalance Score

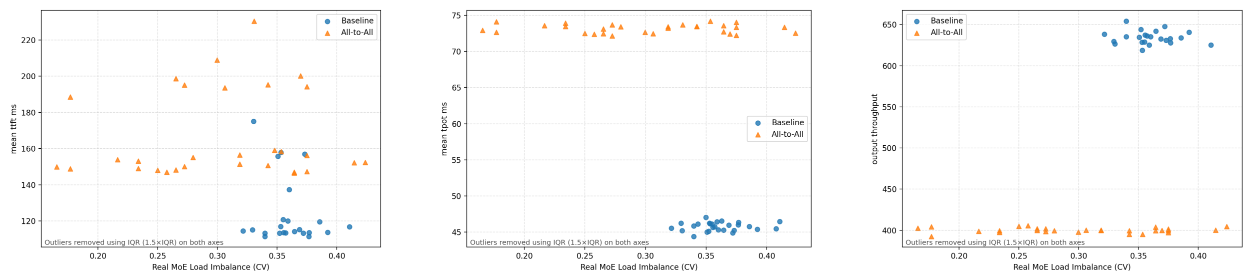
Figure 4: Performance vs. Induced Expert Load Imbalance (Imabalance Scores). Imbalance Scores are derived from token-input statistics.

Induced Imbalance Score (Input-Driven). Figure 4 reports performance as a function of the induced expert load imbalance score. Overall, the imbalance scores are relatively evenly distributed across the x-axis and span a limited range from 0.40 to 0.56. This behavior is expected, as the induced score is computed from coarse-grained token statistics aggregated over each dataset. With a fixed number of prompts and comparable prompt lengths, different datasets can exhibit similar aggregate token distributions, resulting in limited variation in the induced score.

As shown in Figure 4(a), TTFT exhibits no strong monotonic relationship with the induced imbalance score for either configuration. The baseline results remain clustered with moderate variance, while All2All consistently incurs higher TTFT across the entire induced-score range. This suggests that startup latency is dominated by routing and scheduling overheads rather than by input-side token concentration alone.

Figure 4(b) shows similarly weak sensitivity of TPOT to induced imbalance scores. The baseline achieves substantially lower TPOT, whereas All2All maintains a higher and relatively flat TPOT across all induced scores. This indicates that steady-state decoding cost under All2All is largely governed by routing and communication overheads that do not vary significantly with moderate changes in input statistics.

In Figure 4(c), output throughput shows little dependence on the induced imbalance score. The baseline consistently achieves higher throughput, while All2All remains lower and relatively stable. The absence of a clear throughput degradation trend with increasing imbalance score suggests that the input-driven proxy alone is not sufficient to explain performance variation under fixed system settings.



(a) TTFT vs. Imbalance Rate (b) TPOT vs. Imbalance Rate (c) Throughput vs. Imbalance Rate

Figure 5: Performance vs. Actual Expert Load Imbalance Rate (Imbalance Rate) measured during execution.

Observed Imbalance Rate (Runtime-Measured). Figure 5 presents the same analysis using the *observed* expert load imbalance measured at runtime. Unlike the induced imbalance score, this imbalance rate directly reflects realized token-to-expert dispatch behavior during execution. Notably, the baseline and All2All configurations occupy distinct regions along the x-axis: All2All consistently produces lower observed imbalance rates, while the baseline exhibits substantially higher imbalance. This separation indicates that routing strategy itself significantly influences the realized expert load distribution.

As shown in Figure 5(a), TTFT does not exhibit a strong monotonic relationship with the observed imbalance rate within either configuration. However, a clear vertical separation is visible between execution modes: despite achieving lower imbalance, All2All consistently incurs higher TTFT than the baseline. This suggests that startup latency is dominated by routing and communication overhead rather than by expert load skew alone.

Figure 5(b) shows that TPOT remains largely insensitive to variations in observed imbalance. The baseline achieves lower TPOT across its imbalance range, while All2All maintains a higher and relatively flat TPOT even at low imbalance levels. This indicates that reducing expert load imbalance does not directly translate into improved steady-state decoding efficiency under All2All routing.

In Figure 5(c), output throughput exhibits minimal dependence on the observed imbalance rate for both configurations. The baseline consistently achieves higher throughput despite higher imbalance, whereas All2All remains throughput-limited even when expert loads are well balanced. This further suggests that execution-mode overheads, rather than imbalance itself, dominate throughput behavior in this regime.

Induced vs. observed imbalance. Comparing Figures 4 and 5 highlights an important distinction. The induced imbalance score serves as an input-side proxy based on token statistics and exhibits limited dynamic range and predictive power. In contrast, the observed imbalance rate reveals that the routing and execution mode itself substantially affects realized expert load distribution. Although All2All reduces observed imbalance, it does not translate into improved TTFT, TPOT, or throughput in our experiments. This suggests that, under fixed system settings, the dominant performance bottlenecks lie in routing and communication overheads rather than expert load imbalance alone.

7 Discussion

This project highlights both the opportunities and challenges involved in analyzing Mixture-of-Experts (MoE) inference performance under All-to-All routing. While our experimental framework enabled controlled sensitivity studies across configuration parameters, routing strategies, and workload-induced imbalance, several practical difficulties and limitations emerged throughout the process.

Engineering and system challenges. Setting up reproducible MoE experiments proved to be substantially more complex than standard dense-model inference. Integrating vLLM with different MoE kernel backends, enabling expert-level routing logs, and maintaining consistent configurations across runs required significant engineering effort. In particular, compatibility issues between vLLM and PPLX-style fused MoE kernels repeatedly blocked experimentation due to failures in JIT compilation (e.g., during the `torch-c-dlpack-ext` build step). These issues appeared to stem from race conditions caused by concurrent compilation attempts across workers. Despite attempts to precompile extensions and simplify the build process, the problem persisted until switching to a different vLLM version. This experience underscores that kernel-level MoE research is often constrained as much by tooling maturity as by experimental design.

Limits of imbalance controllability. A central challenge in this study was controlling expert load imbalance across a wide dynamic range. Despite constructing synthetic workloads designed to maximize lexical repetition, semantic concentration, or structural similarity, both induced imbalance scores and observed runtime imbalance rates remained within a relatively narrow range. Extremely high or low imbalance regimes were difficult to realize in practice. This suggests that modern learned routers, such as Mixtral's gating mechanism, implicitly regularize expert utilization and prevent pathological skew under typical inference conditions. As a result, imbalance effects may be inherently bounded for realistic workloads, limiting their observable performance impact.

Induced versus observed imbalance. Another important observation is the weak and noisy relationship between induced (input-driven) imbalance scores and observed runtime imbalance. While the induced score captures useful statistical properties of input workloads—such as repetition, entropy, and token concentration—it does not fully account for batching effects, routing interactions, or execution order. In contrast, the runtime imbalance metric derived from expert token counts provides a more faithful measure of actual execution skew, but requires intrusive instrumentation and logging. This gap highlights the difficulty of predicting MoE performance behavior using input-level proxies alone, and suggests that direct runtime measurement remains essential for accurate analysis.

Dominance of inference-time configuration. Across all experiments, inference-time configuration parameters—particularly batch size and total token budget—exerted a stronger and more consistent influence on performance than workload-induced imbalance. Larger batch sizes significantly improved throughput and stabilized All-to-All execution, while smaller batches led to sharp performance fluctuations, especially under baseline kernels. This indicates that, under modern fused All-to-All routing implementations, system-level execution characteristics may dominate over routing-level imbalance effects, at least within the imbalance range achievable by realistic workloads.

Implications for MoE evaluation. Taken together, these findings suggest that evaluating MoE inference performance requires careful separation of system effects from workload effects. While expert load imbalance remains an important theoretical concern, its practical impact may be masked or secondary under optimized routing and batching regimes. Future work would benefit from deeper runtime instrumentation, larger-scale multi-node experiments, and more expressive imbalance metrics capable of capturing temporal and layer-wise routing dynamics. Additionally, more robust and standardized tooling support would greatly lower the barrier to systematic MoE kernel evaluation.

Overall, this project demonstrates that MoE performance analysis is as much a systems engineering problem as it is a modeling problem. The insights gained here provide a useful foundation for future studies aimed at disentangling routing behavior, kernel design, and execution efficiency in large-scale MoE inference.

8 Conclusion

In this work, we conducted a systematic sensitivity analysis of MoE inference performance under All-to-All (All2All) routing using the Mixtral-8x7B model and the vLLM inference engine. Across three controlled experiments, we studied (1) sensitivity to inference-time configuration parameters, (2) the impact of Expert-Parallel Load Balancing (EPLB), and (3) how expert load imbalance correlates with latency and throughput under fixed system settings.

First, we find that inference-time configuration has a first-order impact on performance. In particular, `max-num-seqs` (sequence-level concurrency) and `max-num-batched-tokens` (the per-iteration total token budget cap) jointly determine batching behavior and scheduling efficiency. Under All2All routing, appropriate settings lead to smoother scaling and more stable throughput, while small or poorly tuned configurations amplify variability, especially under the baseline MoE execution path.

Second, we evaluate EPLB under otherwise identical configurations to isolate the effect of explicit post-routing rebalancing. Our results suggest that EPLB provides limited and regime-dependent benefits in our setup, indicating that its net impact depends on how its additional balancing overhead interacts with the underlying routing and batching behavior.

Finally, we isolate the role of expert load imbalance by constructing synthetic workloads with controlled lexical and semantic characteristics while keeping system parameters fixed. Input-induced imbalance scores are useful for generating diverse workloads, but they only weakly reflect performance trends. In contrast, imbalance measured directly from runtime token-to-expert dispatch (via CV over expert token loads) better captures realized routing skew and aligns more consistently with variations in TTFT, TPOT, and output throughput across runs. This runtime view also indicates that All2All routing tends to reduce both the magnitude and variance of observed imbalance relative to the baseline configuration.

Taken together, our findings suggest that routing implementation and execution configuration dominate MoE inference behavior, while workload characteristics primarily influence performance indirectly through realized expert imbalance. More broadly, this study underscores the value of direct instrumentation, careful control of inference-time settings, and system-aware evaluation when deploying MoE models for predictable latency and throughput.

References

[1] Lepikhin, D., et al. *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding*. ICLR 2021.

[2] Fedus, W., Zoph, B., & Shazeer, N. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. ICLR 2022.

[3] Zhang, S., Ye, J., Wang, D., et al. *Tutel: Adaptive Mixture-of-Experts at Scale*. MICRO 2022.

[4] Kwon, W., et al. *vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention*. SOSP 2023.

[5] Jeong, M., Kumar, A., Zinkevich, M., & Smola, A. *MegaBlocks: Efficient Sparse Training with Mixture-of-Experts*. ICML 2023.

[6] Liu, M., Ma, D., Lin, H., et al. *FlexMoE: Scaling MoE Training via Dynamic Slicing*. Meta AI, 2024.

[7] He, Y., Zhang, X., Ren, Y., et al. *FastMoE: A Fast Training System for Large-Scale Sparse Models*. arXiv:2106.04546.

[8] Rajbhandari, S., Ruwase, O., Rasley, J., & He, Y. *DeepSpeed-MoE*. arXiv:2104.07857.