INDENG 174 FINAL PRESENTATION

Simulation and Optimization of Batching Strategies and Scheduling Policies in Large Language Model Inference Systems

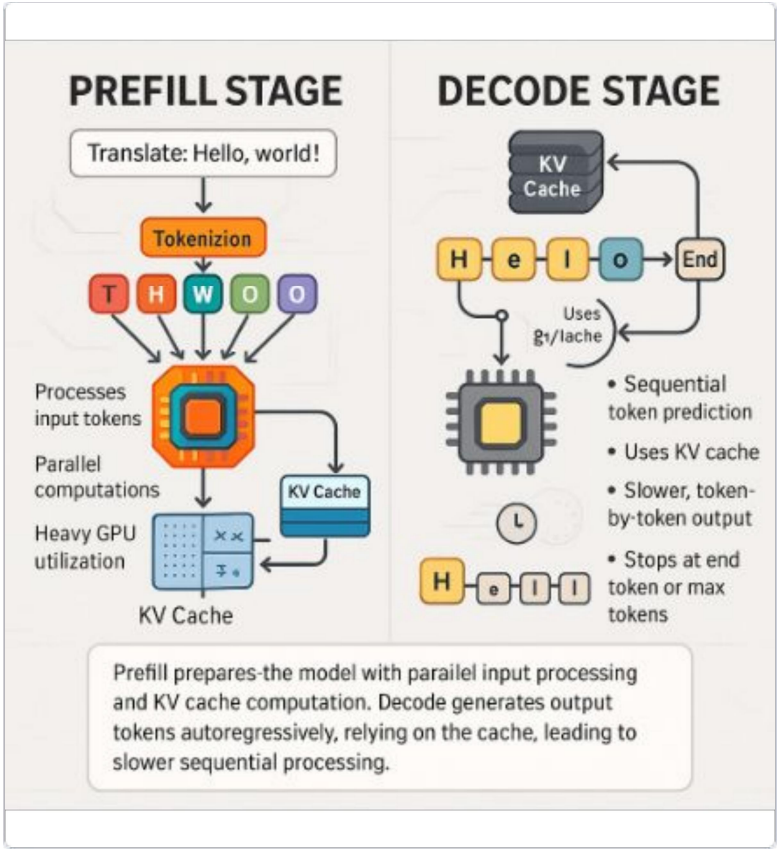
Jiedong Zhang, Qingyang Xu, Runyuan He

December 18, 2025

Abstract

Large Language Model (LLM) inference represents a unique class of stochastic service systems characterized by highly variable service times, distinct two-phase execution dynamics comprising compute-bound prefill and memory-bound decoding, and stringent latency requirements.

This project develops a comprehensive discrete-event simulation (DES) framework to rigorously analyze the performance trade-offs between system throughput, tail latency, and fairness in LLM serving. Utilizing the SimPy library, we modeled an inference engine that replicates the behavioral physics of state-of-the-art systems such as vLLM and Orca.



Motivation

The paradigm shift brought about by Generative AI has transitioned the industry focus from model training to efficient inference serving. Unlike traditional web services with uniform requests, LLM inference imposes unprecedented challenges due to heterogeneity and statefulness.

The "Memory Wall" phenomenon implies that the decoding phase is bound by memory bandwidth rather than compute capability. Understanding these trade-offs requires a simulation environment that can granularly model the interaction between the request queue, the scheduler, and GPU characteristics.

Problem Definition

The core optimization problem is to minimize response latency (p99) while maximizing system throughput and ensuring fairness.

Batch Size Adaptation

How does the selection of batch size B influence the non-linear trade-off between throughput saturation and tail latency degradation?

Fairness & Scheduling

In the presence of heterogeneous request lengths, how do policies like SJF and Priority Scheduling compare against FCFS in terms of fairness?

System Stability

Can adaptive batching strategies mitigate performance collapse during traffic burst periods better than static over-provisioning?

Theoretical Framework

Two-Phase Execution Dynamics

The execution of a Transformer model is divided into two distinct phases with opposing hardware characteristics: the **compute-bound prefill phase** and the **memory-bound decode phase**.

Prefill Latency (Compute Bound):

$$T_{\text{prefill}} = \alpha \cdot \sum_{i \in \text{batch}} l_{\text{prompt},i} + \beta$$

Decode Latency (Memory Bound):

$$T_{\text{decode\_total}} = \gamma \cdot \max_{i \in \text{batch}} l_{\text{output},i}$$

Parameters Calibrated for GPU:

- **α = 0.00015 s/token:** Compute time per prompt token.
- **β = 0.008 s:** Constant system overhead.
- **γ = 0.010 s/step:** Memory access time per decode step.

The decode phase is strictly bound by memory bandwidth (Memory Wall), where increasing batch size linearly improves throughput only up to a saturation point.

Simulation Design

SimPy was selected for its Pythonic generator-based architecture, allowing for flexible modeling of complex logic like preemption and custom scheduling policies.

1. Request Generator

Simulates user queries via a non-homogeneous Poisson process. Prompt lengths follow a LogNormal distribution (μ=3.5, σ=0.8).

2. Scheduler

Acts as the "brain" managing the waiting queue. Implements policies: FCFS, SJF, Predicted-SJF, and Priority with aging.

3. Inference Server

Orchestrates request lifecycles, forming batches of size B and utilizing the Batch Processor to simulate time.

Metric Instrumentation

We track latency metrics including queue time (T_{wait}) and processing time (T_{service}), aggregating these to Total Latency across p50, p95, and p99 percentiles.

Fairness is quantified using **Jain's Fairness Index**, where values near 1 indicate perfect fairness and values near 1/n indicate worst case.

$$J = \frac{(\sum x_i)^2}{n \cdot \sum x_i^2}$$

Throughput is measured as completed requests per second and tokens per second.

Experimental Design

We designed a rigorous experimental campaign consisting of five scenarios. Each data point is the aggregate of 5–10 independent replications with 95% Confidence Intervals.

Exp 1: Batch Size Sensitivity

Determining optimal B for throughput vs. latency trade-off. Range: 1 to 128.

Exp 2: Scheduling Policies

Comparing FCFS, SJF, and Priority under bimodal workload (Short vs Long requests).

Exp 3: Load Stress Testing

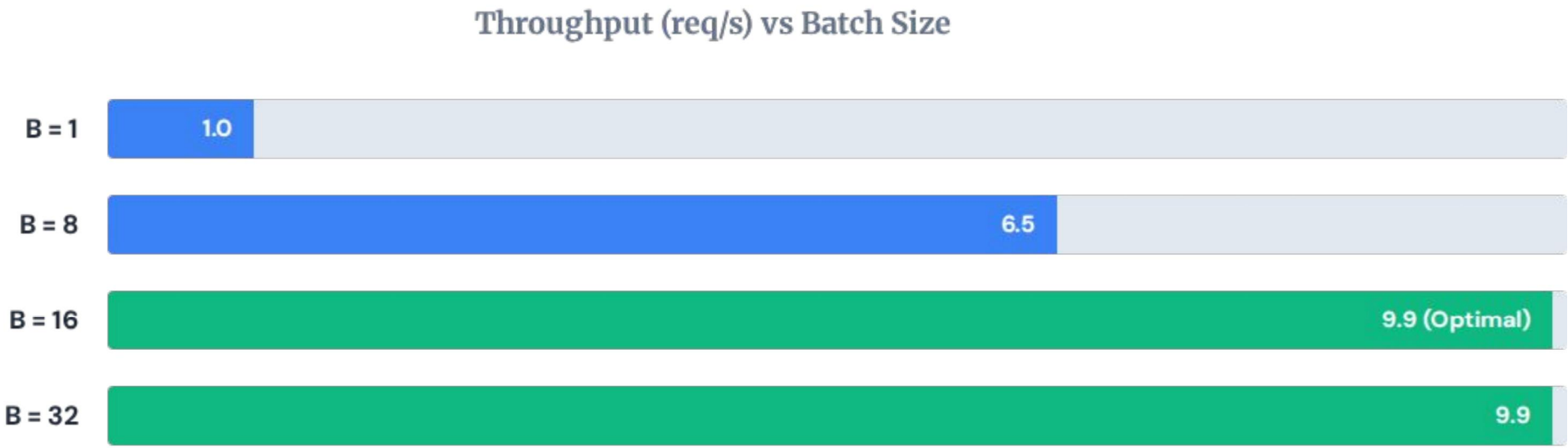
Identifying system saturation point by ramping arrival rate from 1 to 50 req/s.

Exp 5: Adaptive Batching

Evaluating system stability and latency under non-stationary traffic spikes.

Results: Batch Size Optimization

At low batch sizes ($B \leq 8$), the system is severely throughput-bound. The optimal operating point is between B=16 and B=32, where the system maintains low latency while achieving maximum throughput.



Note: At B=1, average latency exceeds 870 seconds due to queue explosion.

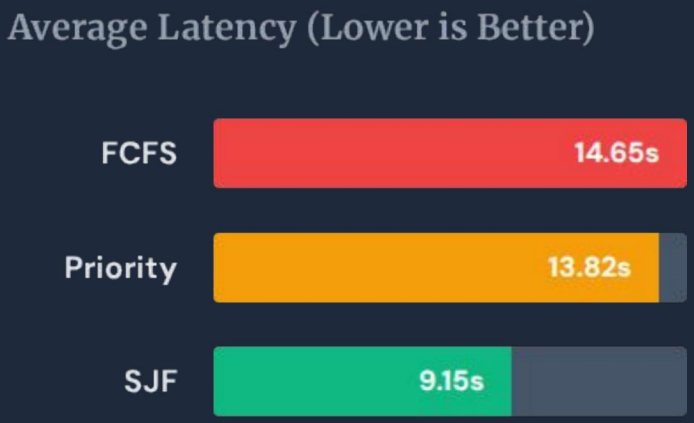
Results: Scheduling Policies

Efficiency vs. Fairness

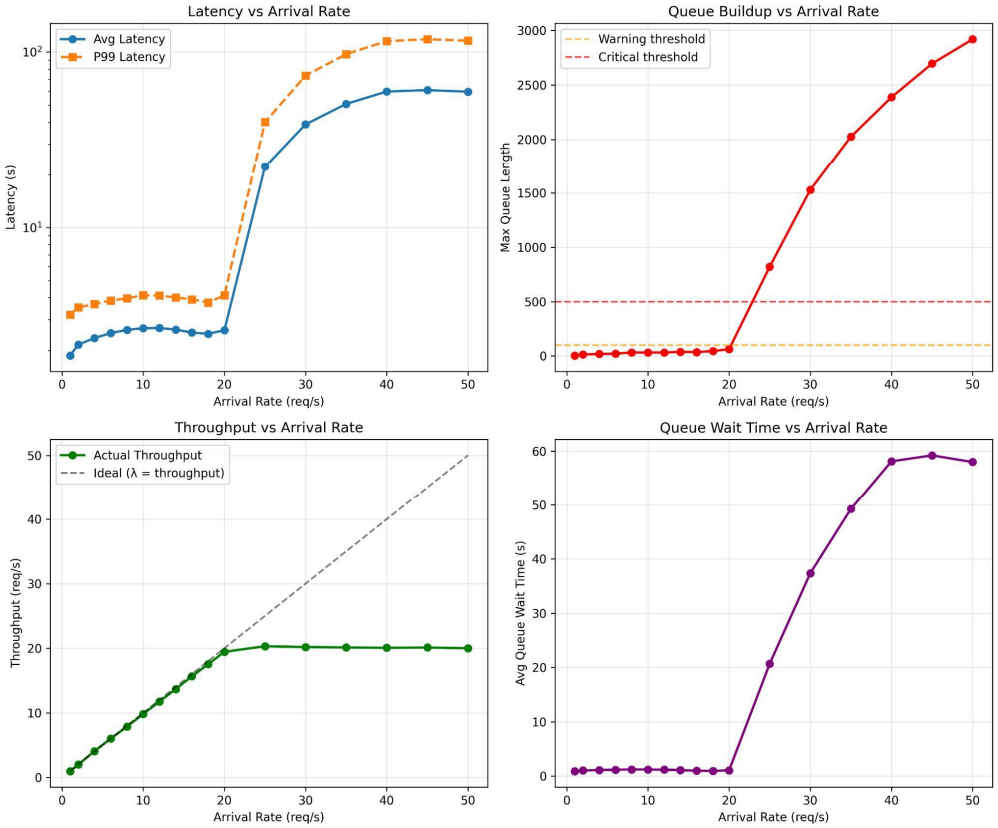
Shortest Job First (SJF) reduces average latency by 37% compared to FCFS. However, this comes at the cost of significant unfairness.

- SJF P99 latency explodes to 190 seconds (7.5x worse than FCFS).
- Jain's Fairness Index drops from 0.87 to 0.09.
- 7% of requests experience starvation under SJF.

Priority (Aging) offers a balanced compromise, maintaining high fairness (0.84) while improving latency.



Results: Load Stress & Saturation



System Breaking Point

The system maintains stability up to approximately **20 requests per second**. Beyond this point, queue length grows linearly and latency degrades exponentially.

Recommendation

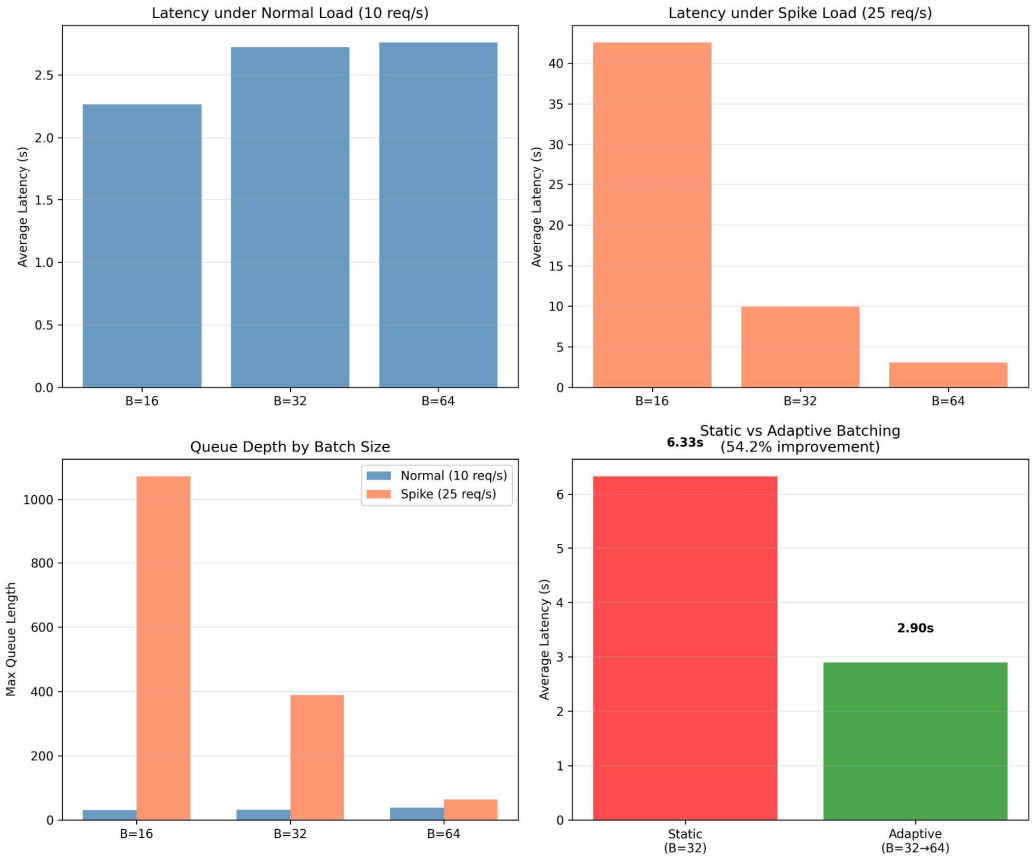
The recommended maximum operating rate is **20 req/s** (80% of saturation point) to maintain SLO guarantees with a safety margin.

Adaptive Batching

Under traffic spikes (25 req/s), static small batch sizes fail catastrophically. The **Adaptive Strategy** (switching from B=32 to B=64 during spikes) proves superior.

Avg Latency Under Spike Load

54% Reduction in Aggregate Latency



Implications for Enterprise Operations

Minimum Batch Size

Ensure batch size is large enough to sustain throughput. At 10 req/s, $B \geq 16$ is required; smaller batches cause catastrophic queue buildup.

Adaptive Batching

Implement queue-depth-based batch size scaling. Use B=32 for low latency and scale to B=64 during spikes to prevent queue explosion.

Tiered Scheduling

If providing tiered services, use priority aging mechanisms rather than pure SJF to prevent the 7% starvation rate seen in static policies.

Capacity Planning

Operate at 80% of the saturation point (~20 req/s at B=32) to ensure system stability and meet Service Level Objectives (SLOs).

Conclusion

This project successfully developed a discrete-event simulation framework to analyze the operational dynamics of LLM inference.

Key Takeaways:

- Batch sizes of 16–32 offer the optimal operating point.
- SJF improves latency by 37% but requires priority aging to mitigate unfairness.
- Adaptive batching reduces latency by 54% under traffic spikes.

Code available at: https://github.com/momoway/llm-infer