International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

# Application of the Central Limit Theorem (CLT) for Performance Modeling in AI-Based Inference Systems

**Prakasharao Raghavapudi[1], Rashmi Gupta[2]**

[1]Independent Researcher, Dallas, TX, United States
Email: *rpraki385[at]gmail.com*

[2]Independent Researcher, Dallas, TX, United States
Email: *rashmig0829[at]gmail.com*

**Abstract:** ***Modern AI applications- including deep learning inference services, large-language-model (LLM) serving platforms, and real-time recommendation engines- operate at massive scale and experience high variability in request latency. Individual inference requests exhibit heavily skewed latency distributions due to network jitter, contention, memory stalls, and nondeterministic scheduling effects. However, performance engineers routinely analyze aggregated metrics such as mean latency, batch averages, and windowed system statistics, which display surprisingly stable and Gaussian-like behavior. This paper demonstrates that the Central Limit Theorem (CLT) provides the mathematical basis for this stability. We simulate AI inference latency using heavy-tailed distributions, compute aggregated batch means, and show that the distribution of batch means converges to a normal distribution even when individual latencies are non-normal. The findings confirm that CLT is foundational for AI performance monitoring, AIOps-based anomaly detection, confidence-bound estimation, and large-scale inference stability analysis.***

**Keywords:** Central Limit Theorem; AI Inference; Performance Engineering; Latency Modeling; AIOps; Statistical Methods; Normal Approximation; Batch Means; Large-Scale Systems.

## 1. Introduction

Artificial Intelligence (AI) systems deployed in production environments process millions of requests per second. These include transformer-based LLMs, deep neural network inference endpoints, and recommender systems. Performance Engineering for such systems focuses on modeling latency, understanding variability, detecting drift, and ensuring availability under unpredictable traffic conditions.

Despite the highly non-normal nature of individual request latencies, aggregated metrics appear stable and approximately Gaussian due to the Central Limit Theorem (CLT). This paper presents simulations demonstrating this effect.

## 2. Central Limit Theorem: Mathematical Background

The Central Limit Theorem states that given independent and identically distributed random variables with finite mean and variance, the normalized sum converges in distribution to a standard normal distribution as sample size increases. This holds regardless of the underlying distribution, making the CLT essential for analyzing aggregated metrics.

## 3. CLT in AI Performance Engineering

CLT influences multiple aspects of AI system monitoring and optimization:

- Latency modeling and smoothing of skewed inference times.
- Windowed metrics for monitoring and drift detection.
- Auto-scaling and forecasting.
- Stabilizing gradient noise during model training.

## 4. Simulation Methodology

Simulations were constructed using Python and C to reflect real AI inference scenarios. Heavy-tailed latency distributions (lognormal and exponential) were generated, and samples were grouped into batches of size 1, 4, 16, and 64. Batch means were computed and analyzed for convergence.

## 5. Results

Raw latency distributions were highly skewed. However, batch means quickly converged to a Gaussian-like distribution as batch size increased. For batch size 64, the distribution was nearly symmetric and bell-shaped, validating the CLT.

## 6. Discussion

The results demonstrate that CLT directly governs aggregated behaviors in AI inference systems. This justifies the use of Z-score thresholds, statistical confidence intervals, drift detection, and forecasting algorithms widely employed in AIOps.

## 7. Conclusion

CLT is fundamental to modern AI performance engineering. It guarantees stable aggregated metrics despite noisy and skewed individual latencies. Future work may extend simulations to GPU-level batching and distributed inference clusters.

## References

[1] Papoulis, A. Probability, Random Variables, and Stochastic Processes.
[2] Montgomery, D. Applied Statistics and Probability for Engineers.
[3] Dean, J., Barroso, L. The Tail at Scale.
[4] Bishop, C. Pattern Recognition and Machine Learning.
[5] IJSR Formatting Guidelines, 2023.

**Appendix A: C Simulation Code for CLT in AI Inference Latency**

The following C program simulates AI inference latency using an exponential distribution, groups requests into batches, computes batch-mean latencies, and writes the results to a CSV file. These batch means can be plotted to visually demonstrate the Central Limit Theorem (CLT).

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/*
 * Simulate CLT for AI inference latency in C.
 *
 * - Per-request latency ~ base_latency + Exponential(lambda)
 *   (skewed, heavy-tailed, similar to real-world async/network delays)
 * - We group requests into batches and compute average latency per batch.
 * - Output (batch_size, batch_index, batch_mean_latency_ms) to CSV.
 */

#define NUM_REQUESTS 200000
#define NUM_BATCH_SIZES 4

/* Generate uniform random number in (0,1) */
static double rand_uniform() {
    return (rand() + 1.0) / ((double)RAND_MAX + 2.0);  // avoid log(0)
}

/* Generate exponential random variable with rate lambda */
static double rand_exponential(double lambda) {
    double u = rand_uniform();
    return -log(u) / lambda;
}

int main(void) {
    int i, b;
    int batch_sizes[NUM_BATCH_SIZES] = {1, 4, 16, 64};
    double *latencies = NULL;
    double base_latency_ms = 50.0;
    double lambda = 1.0 / 20.0;   // mean ~ 20ms for exponential noise

    /* Seed RNG */
    srand((unsigned int)time(NULL));

    /* Allocate memory for simulated latencies */
    latencies = (double *)malloc(NUM_REQUESTS * sizeof(double));
    if (!latencies) {
        fprintf(stderr, "Failed to allocate memory\n");
        return 1;
    }

    /* Step 1: simulate per-request latency */
    for (i = 0; i < NUM_REQUESTS; ++i) {
        double noise = rand_exponential(lambda);
        latencies[i] = base_latency_ms + noise;  // ms
    }

    /* Print some basic stats for raw latencies */
    double sum = 0.0, sum_sq = 0.0, min = latencies[0], max = latencies[0];
```

```
for (i = 0; i < NUM_REQUESTS; ++i) {
   double x = latencies[i];
   sum += x;
   sum_sq += x * x;
   if (x < min) min = x;
   if (x > max) max = x;
}
double mean = sum / NUM_REQUESTS;
double var = (sum_sq / NUM_REQUESTS) - (mean * mean);
double std = (var > 0) ? sqrt(var) : 0.0;

printf("Raw latency stats (ms):\n");
printf("  mean = %.3f\n", mean);
printf("  std  = %.3f\n", std);
printf("  min  = %.3f\n", min);
printf("  max  = %.3f\n\n", max);

/* Step 2: open CSV output */
FILE *fp = fopen("clt_ai_inference_batch_means.csv", "w");
if (!fp) {
   fprintf(stderr, "Failed to open output CSV file\n");
   free(latencies);
   return 1;
}

fprintf(fp, "batch_size,batch_index,batch_mean_latency_ms\n");

/* Step 3: compute batch means for each batch size */
for (b = 0; b < NUM_BATCH_SIZES; ++b) {
   int batch_size = batch_sizes[b];
   int num_batches = NUM_REQUESTS / batch_size;

   printf("Batch size = %d, num_batches = %d\n", batch_size, num_batches);

   double global_sum = 0.0;
   double global_sum_sq = 0.0;

   for (i = 0; i < num_batches; ++i) {
      double batch_sum = 0.0;
      int j;
      for (j = 0; j < batch_size; ++j) {
         int idx = i * batch_size + j;
         batch_sum += latencies[idx];
      }
      double batch_mean = batch_sum / batch_size;

      /* write to CSV */
      fprintf(fp, "%d,%d,%.6f\n", batch_size, i, batch_mean);

      /* accumulate for batch-mean stats */
      global_sum += batch_mean;
      global_sum_sq += batch_mean * batch_mean;
   }

   double mean_b = global_sum / num_batches;
   double var_b = (global_sum_sq / num_batches) - (mean_b * mean_b);
   double std_b = (var_b > 0) ? sqrt(var_b) : 0.0;

   printf("  batch-mean stats: mean = %.3f ms, std = %.3f ms\n\n", mean_b, std_b);
}

fclose(fp);
free(latencies);
```

```
    printf("Wrote batch mean latencies to clt_ai_inference_batch_means.csv\n");
    printf("You can plot histograms grouped by batch_size to see CLT in action.\n");

    return 0;
}
```