

# Performance Testing

## Overview

Performance testing is the process of evaluating how a system behaves under expected workloads. For the City Analysis & Citizen Services AI project, performance testing ensures that the AI model responds quickly, handles multiple user requests efficiently, and provides accurate results without delays.

Since this project is running on Google Colab with T4 GPU, the hardware resources are limited compared to production-grade servers. Hence, it is important to validate how well the system utilizes GPU/CPU, manages memory, and responds under different scenarios.

## Objectives of Performance Testing

1. Measure the response time of the AI model for both city analysis and citizen interaction queries.
2. Check the system throughput – number of queries handled per second/minute.
3. Validate the scalability of the system for multiple parallel users.
4. Monitor resource utilization (GPU memory, CPU usage, RAM usage).
5. Identify bottlenecks and ensure the system does not crash on large inputs.

## Performance Testing Strategy

### Test Environment

Platform: Google Colab

GPU Runtime: T4 GPU (16 GB VRAM)

Frameworks Used: PyTorch, Transformers, Gradio

Deployment Mode: Gradio web interface with share=True

## **Types of Performance Testing Conducted**

### **1. Load Testing**

Measure how the system handles multiple sequential requests (e.g., 50 queries in a row).

### **2. Stress Testing**

Push the system with large inputs (long city descriptions or long queries) to see if it slows down or fails.

### **3. Scalability Testing**

Simulate multiple users by sending concurrent requests through Gradio or Python scripts.

### **4. Latency Testing**

Measure the time taken from query submission to AI-generated response.

### **5. Resource Utilization Monitoring**

Monitor GPU memory, CPU load, and RAM usage during testing.

## **Tools Used**

Python Time Library – to calculate response time.

Torch CUDA Monitoring – for GPU utilization.

Gradio Logs – to check concurrent request handling.

Manual Observation – for latency in interactive usage.

## Test Cases and Results

### Test Case 1: Response Time Measurement

Input: City name = Mumbai

Expected Result: Response generated within 5 seconds.

Observed Result: Average response time = 3.8 seconds (GPU), 8.5 seconds (CPU).

Status:  Pass

### Test Case 2: Multiple Sequential Requests

Input: 50 queries (25 city analysis + 25 citizen services).

Expected Result: All queries processed without crash.

Observed Result:

GPU handled all 50 queries with stable performance.

Slight increase in response time after 40+ requests (due to GPU memory pressure).

Status:  Pass

### Test Case 3: Stress Testing with Long Input

Input: Very detailed query (1000+ characters).

Expected Result: System should generate meaningful output within 10 seconds.

Observed Result: Response generated in 9.2 seconds on GPU.


Status:  Pass (but performance slowed slightly).

#### **Test Case 4: Concurrent Users Simulation**

Method: Two users sending queries at the same time.

Expected Result: Both queries processed without error.

Observed Result: Responses were slightly delayed (average 6.1 seconds).

Status:  Acceptable but needs optimization for heavy multi-user load.

#### **Test Case 5: Resource Utilization**

Observation:

GPU usage peaked at ~80%.

Memory usage stayed within 12 GB (out of 16 GB available).

No system crashes observed.

Status:  Stable

### **Conclusion & Recommendations**

#### **Key Findings**

The AI system performs well under normal workloads (single or few users).

Response time is significantly faster on GPU (3–5 seconds) compared to CPU (8–12 seconds).

The system can handle long queries and multiple sequential requests efficiently.

Concurrent requests slightly increase response latency, but the system remains stable.

Resource utilization is within safe limits on Google Colab T4 GPU.

#### **Bottlenecks Identified**

Multi-user simultaneous queries cause noticeable delays.

Very large prompts (beyond 1000 tokens) may slow down response generation.

Colab GPU limits (session timeout, restricted memory) may affect prolonged testing.

### **Recommendations for Improvement**

1. Optimize Model Parameters – reduce max token length where possible to improve speed.
2. Enable Batch Processing – handle multiple queries in batches to reduce overhead.
3. Deploy on Dedicated Servers – for real-world use, deploy on cloud GPUs (AWS, Azure, GCP) instead of Colab.
4. Introduce Caching – store frequently requested city analyses to avoid repeated computation.
5. Scale with APIs – connect to REST APIs for real-time city data to improve accuracy and reduce computation.