**Performance testing:**

**Performance Testing for Educational AI Assistant**

## **1. Overview**

This document describes the performance testing strategy for the **Educational AI Assistant** built with Gradio, Transformers, and the ibm-granite/granite-3.2-2b-instruct model. Performance testing ensures the app runs efficiently under expected workloads, identifies bottlenecks, and validates scalability for end users.

## 2. **Objectives**

* Measure **response time** for inference requests.
* Evaluate **throughput** (requests per second).
* Monitor **GPU/CPU utilization** and **memory consumption**.
* Test behavior under **concurrent user load**.
* Assess **model scaling** when running locally (CPU vs GPU).

## **3. Test Environment**

* **Hardware**
  + CPU-only system (baseline).
  + GPU-enabled system (e.g., NVIDIA T4, A100, or consumer GPU like RTX 4090).
* **Software**
  + Python ≥ 3.9
  + transformers, torch, gradio
  + Load testing tools: locust, ab (Apache Benchmark), or wrk

## **4.** **Key Metrics**

1. **Response Time (Latency)**
   * Average, p95, and p99 latency for single requests.
   * Breakdown between CPU and GPU inference.
2. **Throughput (RPS – Requests Per Second)**
   * Maximum sustained RPS before system degrades.
3. **Resource Utilization**
   * GPU memory usage (MB).
   * CPU utilization (%).
   * RAM consumption (MB).
4. **Scalability**
   * Performance under increasing concurrent users.
   * Effect of batching (if applied).

## **5**. **Testing Scenarios**

### a. **Single Request Benchmark**

* Send single prompts (e.g., "Explain gravity").
* Measure time from request → response.

### b. **Batch Testing**

* Run multiple sequential prompts.
* Record average latency and system stability.

### c. **Concurrent User Load**

* Simulate 10, 50, 100+ users with tools like locust.
* Measure system degradation, response drops, or timeouts.

### d. **Stress Testing**

* Push beyond expected load until system failure.
* Identify maximum throughput capacity.

### e. **Long Prompt Testing**

* Evaluate performance for large inputs (~512 tokens).
* Check truncation handling and memory usage.

### f. **Quiz Generator Stress**

* Generate long outputs (~1000 tokens).
* Validate memory and speed consistency.

## **6. Tools & Scripts**

### **a. Basic Timer for Latency**

import time

prompt = "Explain the concept of gravity with examples."

start = time.time()

response = concept\_explanation("gravity")

end = time.time()

print("Response Time:", end - start, "seconds")

### **b. Locust Load Test:**

from locust import HttpUser, task, between

class AIUser(HttpUser):

wait\_time = between(1, 5)

@task

def test\_concept(self):

self.client.post("/", json={"data": ["gravity"]})

Run with:

locust -f load\_test.py --host http://127.0.0.1:7860

### **c.** **System Monitoring**

* GPU usage:
* nvidia-smi -l 1
* CPU/memory:
* top -o %CPU
* htop

## **7**. **Acceptance Criteria**

* **Single request latency (GPU):** < 3s for 512 tokens.
* **Single request latency (CPU):** < 10s for 512 tokens.
* **Concurrent load (10 users):** No timeouts; p95 < 5s.
* **Memory usage:** Model fits within available VRAM without OOM.

## **8.** **Reporting**

* Document response time metrics (avg, p95, p99).
* Log throughput (RPS).
* Capture resource usage screenshots (GPU/CPU).
* Identify bottlenecks (e.g., model size, max\_length, sampling strategy).

## **9.** **Optimization Recommendations**

* Enable **torch.compile()** (PyTorch 2.x) for faster inference.
* Reduce **max\_length** for shorter outputs.
* Use **FP16 or INT8 quantization** on GPU for efficiency.
* Apply **model sharding / offloading** if GPU memory is limited.
* Use **request batching** if serving multiple users.