

Short Report: Comparison of REST, tRPC, and gRPC for Image Classification

1. Introduction

This lab exam focuses on developing and evaluating an AI-based image classification system using three different communication approaches: REST, tRPC, and gRPC. The goal is to analyze their performance and architectural characteristics in a distributed and microservices-based environment.

2. System Implementation Overview

- **REST API:** Implemented using Node.js and Express. Images are uploaded using HTTP POST requests and responses are returned in JSON format.
- **tRPC API:** Implemented using tRPC with Express, providing type-safe client-server communication over HTTP using JSON.
- **gRPC API:** Implemented using Protocol Buffers with two microservices:
- **Microservice A:** API/Gateway Service
- **Microservice B:** AI Model Service gRPC is used for service-to-service communication with binary serialization.

A dummy AI classifier was used to return labels and confidence values, which is sufficient for comparing distributed system performance.

3. Experimental Setup

- Environment: Local execution on VS Code (Node.js v22.15.0)
- Tests Performed:
 - REST image upload and classification
 - tRPC procedure execution
 - gRPC unary RPC with service-to-service communication
- Metrics Observed:
 - Response time (from terminal logs)
 - Serialization and payload characteristics

4. Observed Results

Technology	Observed Response Time	Payload Type	Type Safety	Remarks
REST	~11 ms (server), ~162 ms (end-to-end)	JSON (Text)	No	Simple and widely supported
tRPC	Comparable to REST	JSON (Text)	Yes	Compile-time client-server type safety

Technology	Observed Response Time	Payload Type	Type Safety	Remarks
gRPC	~196 ms (internal call)	Protobuf (Binary)	Yes	Efficient binary serialization
gRPC (Microservice)	~196 ms (A → B communication)	Protobuf (Binary)	Yes	Scalable and loosely coupled

All values were taken directly from terminal outputs during the lab exam execution.

5. Analysis

REST demonstrated low server processing time but higher overall latency due to HTTP and JSON overhead. It is easy to implement and debug but less efficient for high-performance systems.

tRPC provided similar performance to REST because it also uses JSON; however, it significantly improved developer productivity by ensuring end-to-end type safety between client and server.

gRPC used Protocol Buffers for binary serialization, resulting in smaller payload sizes and better serialization efficiency. Although internal service-to-service communication added some latency, it enables clean microservice separation and better scalability.

6. Conclusion

Based on the observed results, gRPC is the most suitable choice for high-performance and scalable microservice architectures due to its binary serialization and efficient communication. REST is best suited for simple and interoperable APIs, while tRPC is ideal for TypeScript-based systems that require strong type safety without sacrificing simplicity.

This lab successfully demonstrates the practical differences between REST, tRPC, and gRPC in distributed and parallel computing systems.