LLM Sentiment Analysis on AWS Lambda

Cloud Computing — Project Presentation

Dario Loi, 1940849 Francesco Fazzari, 1935070

June 16, 2025

Sapienza, University of Rome

Introduction

The Goal

To deploy a sentiment analysis classifier on a serverless architecture and evaluate its real-time performance on CPU-only infrastructure. The model analyzes emotional tone in text, classifying it as happy, angry, sad, or other.

The Method

We leveraged AWS Lambda for its auto-scaling and simplicity. To overcome the challenges of large model sizes and dependencies, we used advanced techniques like model quantization and compilation to the ONNX standard.

The Evaluation

An extensive benchmark was performed using the Locust testing framework to simulate realistic user loads. We measured both user-oriented (e.g., response time) and system-oriented (e.g., concurrency) metrics.

System Architecture & Optimization

System Architecture

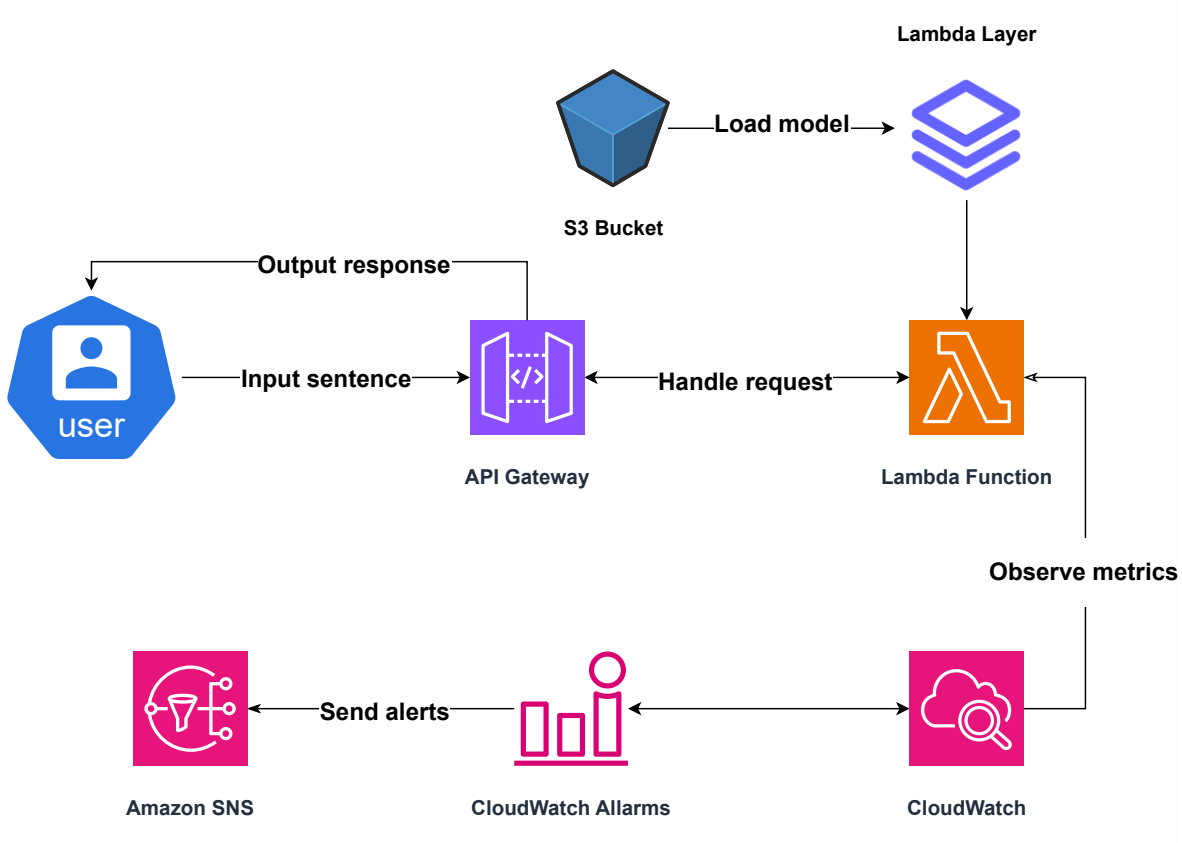

Figure 1: The user sends a sentence to an API Gateway, which triggers an AWS Lambda function. The function loads an optimized AI model from an S3 bucket to perform inference. Metrics are monitored by CloudWatch, which sends alerts via Amazon SNS if issues are detected.

AI Model Optimization

The Challenge

Deploying large AI models on serverless platforms like AWS Lambda presents two main hurdles:

- The PyTorch framework is too large, exceeding AWS Lambda's size limits.
- The model's parameters are in the dozens of megabytes, also violating size constraints.

AI Model Optimization

The Solution: Quantization

- We converted our MobileBERT model to the **ONNX standard**, allowing us to use the much lighter `onnxruntime` library.
- During conversion, we performed **dynamic quantization**, reducing the model's parameters from 32-bit floats to 8-bit unsigned integers.
- **Result:** Model size was reduced by a factor of ≈ 3.68, from 81.5 MiB to just 22.1 MiB.

Performance Evaluation

Load Testing Scenarios

We used Locust to simulate realistic user behavior under three distinct scenarios to test horizontal scaling and robustness.

Table 1: Load Testing Configurations

Scenario	Users	Ramp-up	Duration	Request Delay (s)
Bursty	3	3 s	2 min	[5, 10]
Medium	50	25 s	2 min	[3, 5]
Heavy	100	50 s	1 min	[1, 3]

- **Bursty:** Models sparse, intermittent requests.
- **Medium & Heavy:** Test the infrastructure's ability to handle significant and high volumes of traffic.

Results

Results: Concurrency and Scaling

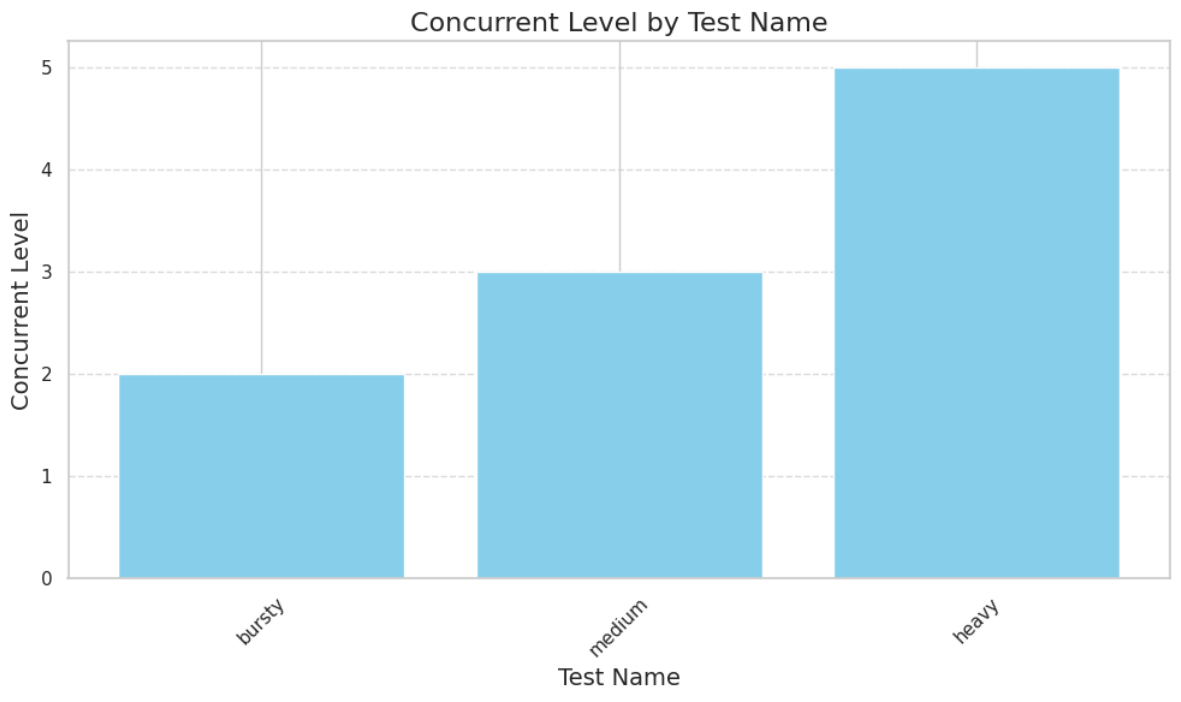

Figure 2: Maximum concurrent Lambda executions as reported by CloudWatch during each test.

Results: Concurrency and Scaling

Insight

The system is incredibly efficient. Even under a "Heavy" load of 100 concurrent users, the infrastructure scaled to a maximum of only **5 concurrent Lambda instances**. This demonstrates excellent horizontal scaling and cost-effectiveness, processing thousands of requests with 100% availability.

Results: Horizontal Scaling Under Load

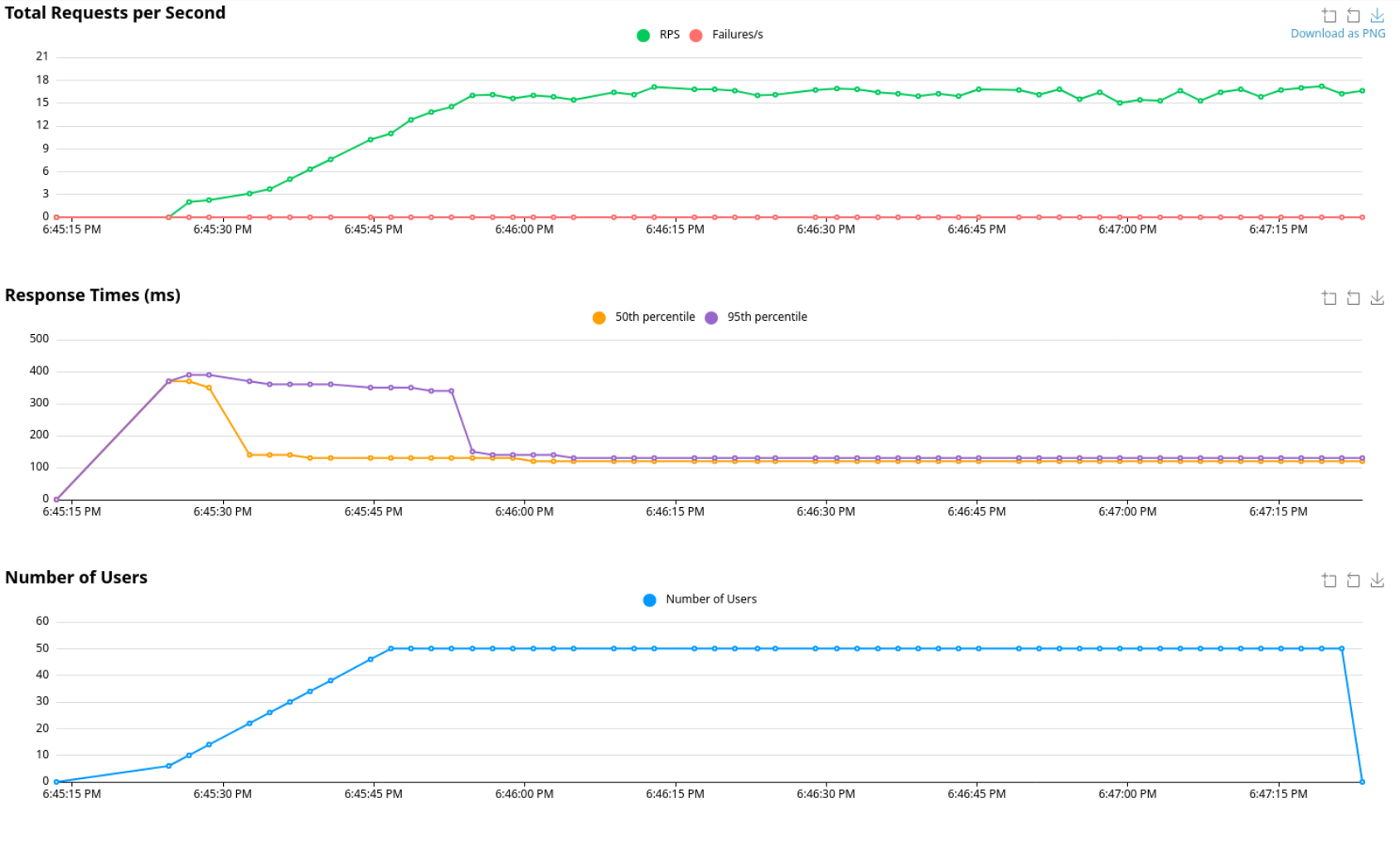

Figure 3: Graphs for the Medium Load Test.

Results: Horizontal Scaling Under Load

Insight

The top graph shows a linear correlation between active users and requests per second, indicating successful scaling. The high 99th-percentile response time during the ramp-up period (middle graph) suggests that new users were assigned to new Lambda instances, forcing a "cold start" and proving that our infrastructure was **actively and automatically scaling out** to meet demand.

Results: Vertical Scaling

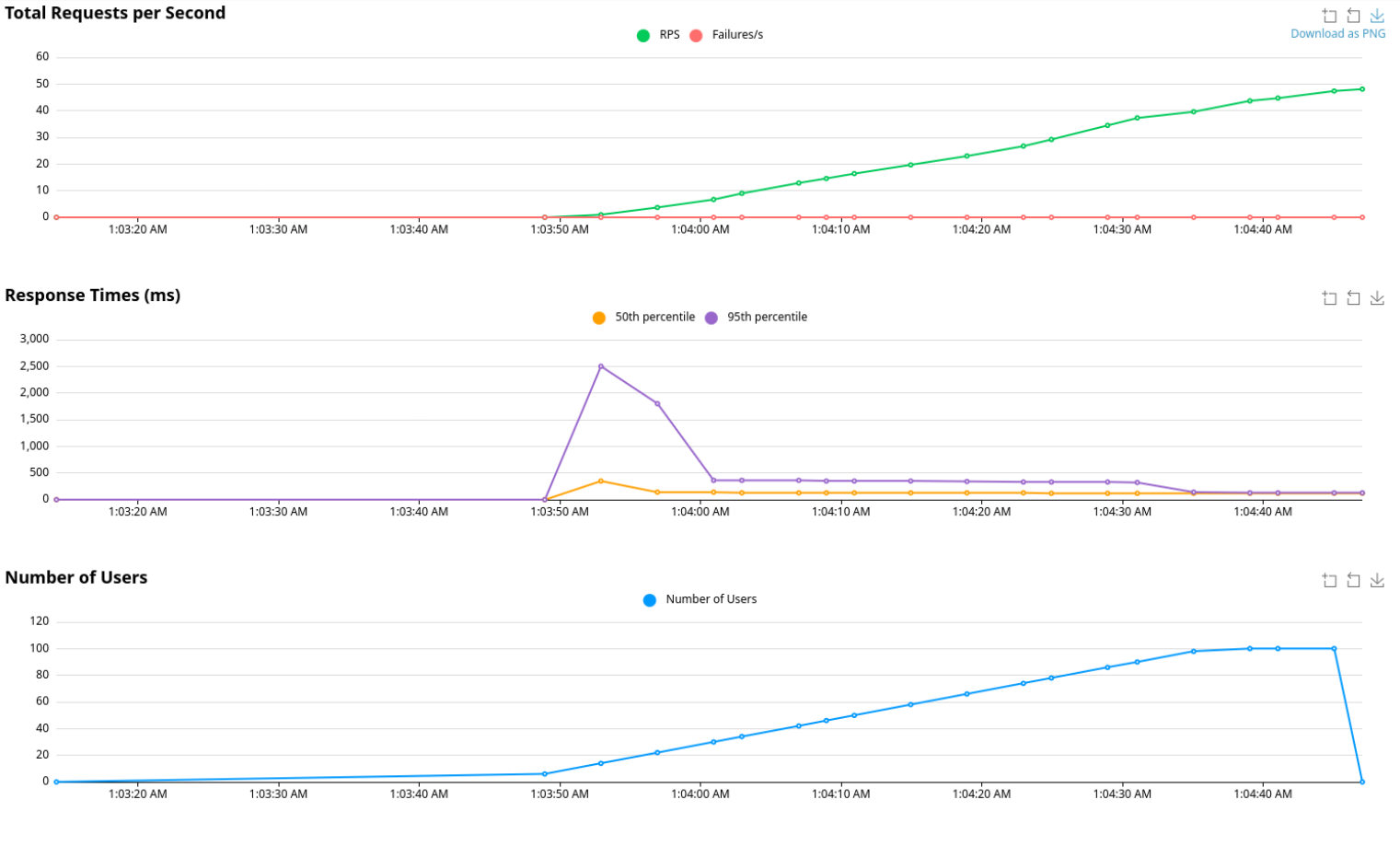

Figure 4: Graphs for the Heavy Load Test with Lambda RAM doubled to 1024 MiB.

Results: Vertical Scaling

Insight

Doubling the provisioned RAM (and associated CPU power) showed **no significant improvement** in response times. This suggests that individual requests were already being processed near-optimally and that the primary bottleneck is not compute resources, but rather network overhead, which we estimated to be around 100ms.

Aggregated Performance

Key Takeaway

Across all three demanding test scenarios, the service handled thousands of requests with **zero failures or errors**. The median response time remained stable at around 120-130ms, demonstrating a robust and reliable system. The high "Max" times are attributed to initial "cold starts".

Conclusion

Conclusion

Achievements

- We successfully demonstrated that it's possible to deploy a lightweight, efficient, and performant LLM for a real-world task on a serverless, CPU-only cloud architecture.
- Our proposed architecture proved to be simple, highly scalable, and fault-free under realistic traffic loads.

Conclusion

Key Findings

- The system scales effortlessly to meet user demand, both horizontally (adding instances) and vertically (utilizing resources).
- The primary performance bottleneck was identified as network overhead, not computational power.

Future Work

Network latency could be further reduced by deploying the serverless infrastructure closer to end-users via edge locations, an easy-to-implement next step.

The End

Thank you for your attention!

Questions?