

Midterm Report: A Comparative Analysis of Cost-Performance for AI Model Serving on AWS

Rukai Xiao

124090729@link.cuhk.edu.cn

Peixin Yao

122090663@link.cuhk.edu.cn

Hezhe Wang

122040066@link.cuhk.edu.cn

November 16, 2025

1 Project Roadmap & Outline

This project seeks to address a critical information gap for developers: quantifying the cost-performance trade-offs for AI inference workloads across Amazon Web Services (AWS). Our core objective remains to answer the question: *Which cloud architecture provides the best performance per dollar?*

This midterm report outlines our established methodology and presents preliminary findings focused on demonstrating the feasibility of our approach, in line with the project's timeline and goals.

1.1 Standardized Workload

To ensure a fair and reproducible comparison, we utilize a pre-trained **MobileNetV2** model from the PyTorch library as our standard computational workload. The model and all its dependencies, including a heavy PyTorch library, are encapsulated within a **Docker image** to guarantee a consistent execution environment across all testbeds.

1.2 Test Environments Midterm Strategy

Our original proposal identified four representative AWS compute services for benchmarking:

- **GPU Instance:** g4dn.xlarge (High-performance baseline)
- **x86 CPU Instance:** t3.medium (General-purpose baseline)
- **ARM CPU Instance:** t4g.medium (Efficiency-optimized baseline)
- **Serverless Function:** AWS Lambda (Event-driven architecture)

For this Midterm Report, we have adopted a focused ‘**Triage-Sprint**’ strategy to prioritize feasibility and generate meaningful initial results. Our preliminary experiments and analysis will concentrate exclusively on the **AWS Lambda** environment. This approach allows us to validate our data collection pipeline and address core technical challenges before expanding the benchmark to other EC2 instances.

1.3 Remaining Roadmap

With the feasibility of our deployment and testing pipeline confirmed, our next steps are:

1. Complete a comprehensive set of benchmarks for the AWS Lambda environment, including a detailed analysis of cold vs. warm start performance.
2. Systematically extend the benchmarks to the remaining EC2 testbeds (g4dn.xlarge, t3.medium, t4g.medium).
3. Perform a detailed cost analysis based on the collected performance data to calculate the final ”Cost per 1 Million Inferences” metric for each service.
4. Synthesize all findings into the final report and prepare the demo video, as per the course requirements.

1.4 Data Collection & Key Metrics

Performance data is collected using a custom Python script (`performance_test.py`), which simulates concurrent user requests and measures key indicators. Our analysis focuses on two primary performance metrics:

Latency (ms): The time taken to process a single inference request.

Throughput (IPS): The number of inferences processed per second.

These technical figures will be normalized into our final, unified metric for comparison: **Cost per 1 Million Inferences**. This metric provides a direct translation of technical performance into financial efficiency. For instance-based services, it is calculated as:

$$\$/1M \text{ Inferences} = \left(\frac{\text{Hourly Cost}}{\text{IPS} \times 3600} \right) \times 1,000,000 \quad (1)$$

An equivalent value will be derived for AWS Lambda based on its per-invocation and duration-based pricing model to ensure metric consistency.

2 Preliminary Results & Analysis (AWS Lambda)

This section details our initial experiments conducted on the AWS Lambda platform, demonstrating the viability of our testing methodology.

2.1 Experimental Setup

- **Target Service:** The containerized MobileNetV2 model deployed as an AWS Lambda function.
 - Memory Allocated: 1024 MB
 - Timeout: 90 seconds
 - Architecture: x86_64
- **Trigger:** The Lambda function is invoked via a publicly accessible API Gateway endpoint.
- **Testing Client:** To mitigate client-side bottlenecks (see Section 3.1), all tests were executed from an over-provisioned AWS EC2 instance (c5.2xlarge). This ensures that measured latency reflects server-side performance, not client-side limitations.

2.2 Performance Measurements

We conducted a series of tests under varying concurrency levels to understand the performance characteristics of the Lambda deployment. The key findings are summarized below.

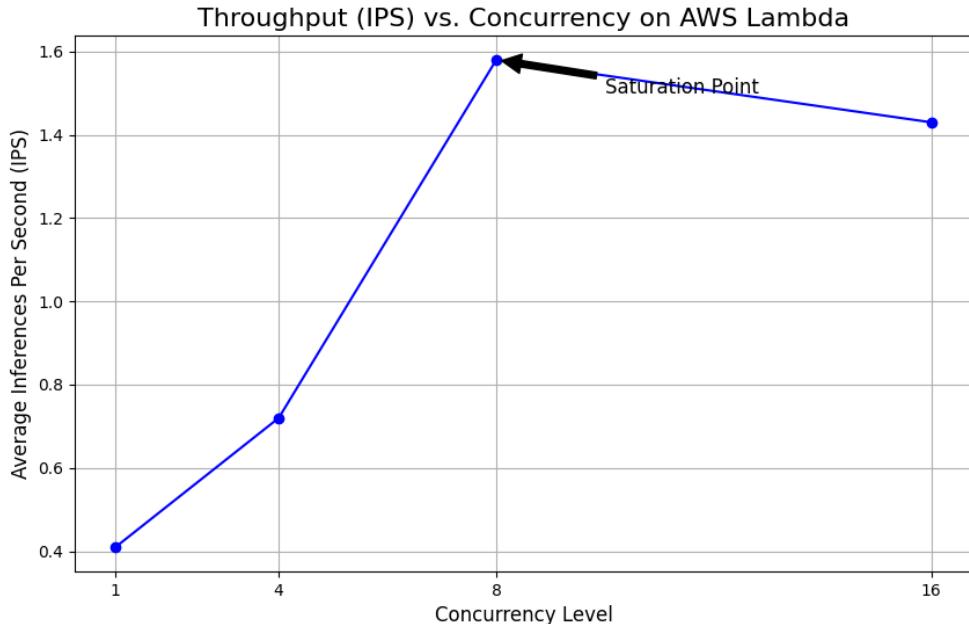
Figure 1: Example raw output from our performance testing script for the C=16 test run.

```
PS C:\Temp_project> python performance_test.py --url https://8keg68m54a.execute-api.us-east-1.amazonaws.com/default/csc4160-inference-target --image-dir test_images/ --concurrency 16 --duration 60
Found 1 images for testing.
Starting test for https://8keg68m54a.execute-api.us-east-1.amazonaws.com/default/csc4160-inference-target with concurrency: 16...
Raw data saved to: results\8keg68m54a_execute-api_us-east-1_amazonaws_com_default_csc4160-inference-target_c16_duration60s_20251115_014243.csv
=====
Performance Test Summary
=====
Target URL:      https://8keg68m54a.execute-api.us-east-1.amazonaws.com/default/csc4160-inference-target
Concurrency:    16
Actual Duration: 92.33 seconds
-----
Total Requests: 132
Successful:     132 (100.00%)
Failed:          0
-----
Average IPS:    1.43 (Inferences Per Second)
-----
Min Latency:    440.23 ms
Avg Latency:    698.62 ms
Max Latency:    2494.47 ms
P95 Latency:    2180.56 ms
P99 Latency:    2272.30 ms
=====
PS C:\Temp_project>
```

Table 1: Preliminary Performance of MobileNetV2 on AWS Lambda

Concurrency Level	Avg. Latency (ms)	Throughput (IPS)	Test Mode
1	2456.07	0.41	10 Requests
4	1380.44	0.72	8 Requests
8	630.82	1.58	60 Seconds
16	698.62	1.43	60 Seconds

Figure 2: Throughput vs. Concurrency on AWS Lambda



2.3 Preliminary Cost-Performance Analysis

Using the throughput data from our C=8 test run, which represents the saturation point of our current Lambda configuration, we can perform a preliminary calculation of the "Cost per 1 Million Inferences".

The pricing for AWS Lambda (us-east-1, x86) is approximately \$0.20 per 1M requests and \$0.0000166667 for every GB-second of duration. With our 1024MB (1GB) memory configuration:

- **Sustained Throughput (IPS):** 1.58 (from C=8 test)
- **Average Latency:** 630.82 ms (or 0.631 seconds)

The cost is composed of a per-request cost and a duration cost:

- **Duration Cost per Inference:** $1\text{GB} \times 0.631\text{s} \times \$0.0000166667/\text{GB-s} \approx \0.0000105
- **Request Cost per Inference:** $\$0.20/1,000,000 = \0.0000002

Total Cost per 1 Million Inferences:

$$(\$0.0000105 + \$0.0000002) \times 1,000,000 \approx \$10.70 \quad (2)$$

This preliminary calculation demonstrates our ability to derive the final comparison metric and provides a baseline cost figure for the serverless architecture.

3 Technical Challenges

3.1 Identified Challenges

During this initial phase, we identified several primary technical challenges:

1. Client-Side Bottleneck:

The initial test script (`performance_test.py`) utilized a CPU-bound `multiprocessing` library. High concurrency settings would saturate the client machine's CPU, producing polluted latency data. Our containment protocol—running the script on a powerful EC2 instance (c5.2xlarge)—successfully mitigated this issue. A later revision of the script using `threading` further optimized client-side performance.

2. AWS Lambda Cold Start & Timeouts:

The large container image size (>3GB initially) and the need to download model weights on first invocation led to severe cold start latencies exceeding 30 seconds. This surpassed the API Gateway's 29-second timeout limit, causing initial tests to fail completely. This was resolved by:

- Separating the large PyTorch libraries onto an EFS volume.
- Pre-caching the model weights directly onto the EFS.
- Utilizing Provisioned Concurrency to keep at least one instance warm.

3. Container Environment Constraints:

The read-only nature of the Lambda file system (except for `/tmp`) initially conflicted with PyTorch's default behavior of writing to a home directory cache. This was resolved by explicitly setting the `TORCH_HOME` environment variable to a writable path on the mounted EFS.