Scalable AI Infrastructure Project

Group 1: Michael Amadi and Christian Ollen

Project Summary: *Distributed Workflow for Astronomy Data Processing Using AWS*

Building a distributed pipeline on AWS to analyze large-scale astronomy-based multimodal datasets

Leverages serverless functions for inter-process communication and parallel inference

Goal: evaluate scalability, cost, and performance of the serverless architecture

Project is divided into four components:

- Orchestration
- Communication
- Local inference
- Distributed inference

Data

Sourced from Google Drive, the dataset structure was stored in serialized PyTorch .pt files containing tensor-based data for anomaly detection inference benchmarks

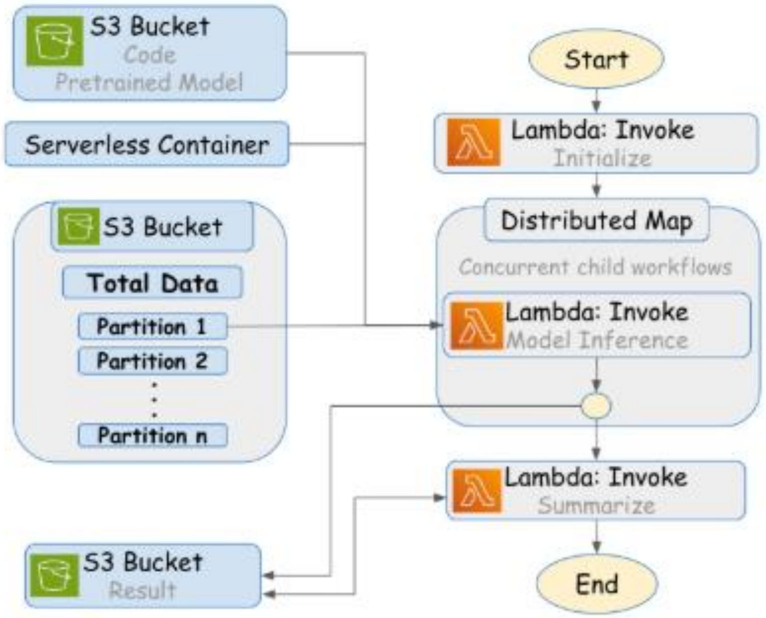
Split into smaller partitions (e.g., 100MB chunks) using a custom split_data.py script to enable parallel execution on AWS Lambda or SageMaker

Preprocessing steps included data cleaning, chunking, and uploading to S3 for structured and scalable processing in a serverless environment

Experimental Plan

- **Step Functions:** Measure cost, runtime, and memory usage by varying the "world size" (parallelism level).
- **Rendezvous Server:** Validate communication between Lambda functions and test deployment latency.
- **Local Inference:** Run the astronomy data pipeline locally to establish baseline execution metrics.
- **Distributed Inference:** Use FMI with Step Functions to execute at scale and compare results against local benchmarks.

Fig 1: CAI framework design on AWS State Machine.



Performance Measurement

25 AWS experiments tested combinations of data prefix size (10MB–100MB) and file limits (2–10).

Performance scaled with file size: Increasing from 10MB → 100MB boosted throughput from ~23.7 MB/s to ~39 MB/s.

Larger file limits improved parallelism: File limit = 10 consistently achieved higher sample rates (up to 237.35 samples/sec).

Best throughput was achieved in Experiments 21–25 (100MB, file limit = 10), peaking at 38.96 MB/s with 2.28s batch time.

Memory scaled with input size: From 7.1 GB (10MB prefix) to ~71 GB (100MB prefix) — manageable with serverless scalability.

Results confirm Step Functions can be tuned for high-throughput, low-latency inference by optimizing data partitioning and parallel file processing.

Experiment	Data Prefix	File Limit	Avg CPU Time (s)	Avg CPU Memory (MB)	Avg Exec Time per Batch (s)	Avg Throughput (MB/s)	Avg Samples/s
Experiment1	100MB	2	25.34	14295.35	0.33	5.09	201.66
Experiment2	10MB	4	3.56	7151.39	3.57	23.70	144.36
Experiment3	10MB	2	3.56	7151.39	3.57	23.70	144.36
Experiment4	10MB	4	3.56	7151.39	3.57	23.70	144.36
Experiment5	10MB	6	3.56	7151.39	3.57	23.70	144.36
Experiment6	10MB	8	3.56	7151.39	3.57	23.70	144.36
Experiment7	25MB	10	7.28	17864.66	2.92	31.04	189.07
Experiment8	25MB	2	7.28	17864.66	2.92	31.04	189.07
Experiment9	25MB	4	7.28	17864.66	2.92	31.04	189.07
Experiment10	25MB	6	7.28	17864.66	2.92	31.04	189.07
Experiment11	50MB	8	11.89	35579.80	2.40	37.05	225.67
Experiment15	50MB	10	11.89	35579.80	2.40	37.05	225.67
Experiment16	75MB	2	18.37	53229.60	2.47	37.21	226.64
Experiment17	75MB	4	18.37	53229.60	2.47	37.21	226.64
Experiment18	75MB	6	18.37	53229.60	2.47	37.21	226.64
Experiment19	75MB	8	18.37	53229.60	2.47	37.21	226.64
Experiment20	75MB	10	18.37	53229.60	2.47	37.21	226.64
Experiment21	100MB	2	22.35	70800.08	2.28	38.96	237.35
Experiment22	100MB	4	22.35	70800.08	2.28	38.96	237.35
Experiment23	100MB	6	22.35	70800.08	2.28	38.96	237.35
Experiment24	100MB	8	22.35	70800.08	2.28	38.96	237.35
Experiment25	100MB	10	22.35	70800.08	2.28	38.96	237.35

Batch Size Analysis

- Execution time remained constant across batch sizes within the same dataset group:
- Example: For 10MB/100MB/1, mean execution time about 3.57s for all batch sizes (32, 64, 128, 512).
- In the 25MB/10GB/1 dataset, larger batch sizes did not reduce mean execution time, all configs about 2.92s.
- Results for 100MB/1G/3 show execution time as 0.05s, likely due to a very small or cached workload (or misconfiguration).
- No meaningful variance in min/max execution time across batch sizes, suggesting batch size had minimal impact on performance in the given context.
- Implication: In distributed or parallel workflows, performance may be more sensitive to data chunking and I/O bottlenecks than batch size alone

[2]:

	dataset	batch_size	mean_execution_time	min_execution_time	max_execution_time
0	100MB/1G/3	32	0.050000	0.050000	0.050000
1	100MB/1G/3	64	0.050000	0.050000	0.050000
2	100MB/1G/3	128	0.050000	0.050000	0.050000
3	100MB/1G/3	512	0.050000	0.050000	0.050000
4	10MB/100MB/1	32	3.569510	3.329745	4.489512
5	10MB/100MB/1	64	3.569510	3.329745	4.489512
6	10MB/100MB/1	128	3.569510	3.329745	4.489512
7	10MB/100MB/1	512	3.569510	3.329745	4.489512
8	25MB/10GB/1	32	2.915121	2.018642	10.759740
9	25MB/10GB/1	64	2.915121	2.018642	10.759740
10	25MB/10GB/1	128	2.915121	2.018642	10.759740
11	25MB/10GB/1	512	2.915121	2.018642	10.759740

Cost Analysis

- Costs scale with memory and request duration: larger partitions use more memory and run longer, increasing total cost.
- Most cost-effective config: 25MB partitions
  - highest request count (1600) at only $0.16, due to balanced memory and execution time.
- Highest cost: 100MB partitions at $0.38, driven by 60.2 GB of total memory usage and longer run time (22.84s).
- Trade-off identified: 75MB offered near-peak throughput with moderate cost ($0.30) — good balance for scalable performance.
- Optimizing partition size is key to minimizing cost while maintaining throughput.

partition	Requests	Duration_s	Memory_GB	Cost ($)
10MB	100	3.56	7.0	NaN
25MB	1600	7.28	17.4	0.16
50MB	410	11.89	34.7	0.20
75MB	685	18.37	52.0	0.30
100MB	615	22.84	60.2	0.38

Summary

- Local CPU-based runs provided a baseline for batch performance, with batch size 128 offering the best throughput (~11.03 Mbps).
- AWS Step Functions enabled massive scalability, with peak throughput reaching about 39 MB/s
  - **Over 30× faster than local execution.**
- Cost-effective and high-performing setups were identified:
  - 25MB partition: best for low-cost inference ($0.16 total).
  - 75MB partition: optimal throughput-to-cost trade-off ($0.30).
- Batch size had minimal impact on performance in AWS workflows, as shown in the Batch Size Analysis:
- Execution times remained constant across batch sizes for the same dataset.
- Performance bottlenecks were more influenced by data prefix size and I/O limitations.
- Memory scaled predictably with input size (7 GB -71 GB), handled efficiently by serverless compute.
- Overall, the project demonstrated that cloud-native inference pipelines can achieve low-latency, high-throughput, and cost-effective scalability when tuned with:
  - Proper partitioning
  - Concurrency (file limits)
  - Dataset design (chunk size)