

Configuration Manual

MSc Research Project
Cloud Computing

Moiz Ahmed Nurul Hasan Khan
Student ID: 22201785

School of Computing
National College of Ireland

Supervisor: Prof Aqeel Kazmi

National College of Ireland
MSc Project Submission Sheet



School of Computing

Moiz Ahmed Nurul Hasan Khan

Student Name:

Student ID: x22201785

Programme: MSc in Cloud Computing **Year:** 2023-2024

Module: Research in Computing

Lecturer: Prof Aqeel Kazmi

Submission Due Date: 16th September 2024

Project Title: Minimizing Cold Starts in Serverless Environments with Predictive Optimization Approach Using Bi-LSTM and Genetic Algorithms

Word Count: 959 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Moiz Ahmed Nurul Hasan Khan

Date: 16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Moiz Ahmed Nurul Hasan Khan
Student ID: 22201785

1 Introduction

In this guidebook you will see how to deploy and run an optimized cold start solution in a stateless serverless environment using AWS Lambda. The primary goal is to apply the Bi-LSTM approach for building a machine learning algorithm. In order to overcome the cold start problem described, genetic algorithm is introduced to optimize the handling of Lambda function invocation. A dashboard is also provided to monitor the machine learning model and its training in the serverless mode of an application with a user-friendly interface. Through this guide, users will be able to understand how to configure, run and manage the system.

2 System Configuration

This section outlines the requirements of Hardware and Software.

2.1 Hardware Requirements

- **Brand and Model:** MacBook Pro or Dell Precision
- **Processor:** Apple M2 Max or Intel Core i7/i9 processor
- **Memory:** 16 GB RAM to handle data processing
- **Storage:** 250 GB of SSD storage for quick access to the files and data processing.
- **Internet Connection:** Stable and fast internet connection for accessing cloud services.

2.2 Software requirements

- **Operating System:** Windows 10/11, macOS, or Linux
- **Programming Languages:** Python 3, JavaScript
- **Machine Learning and Data Processing Libraries:** TensorFlow, NumPy, Pandas, Matplotlib, Scikit-learn.
- **AWS Account for deploying and managing serverless architecture**
- **Integrated Development Environment (IDE):** VS code or Jupiter Notebook

3 Project Implementation

3.1 Environment setup

3.1.1 Setup AWS Account

- Create or Login to your AWS account to access the services such as Lambda, S3, SageMaker and CloudWatch

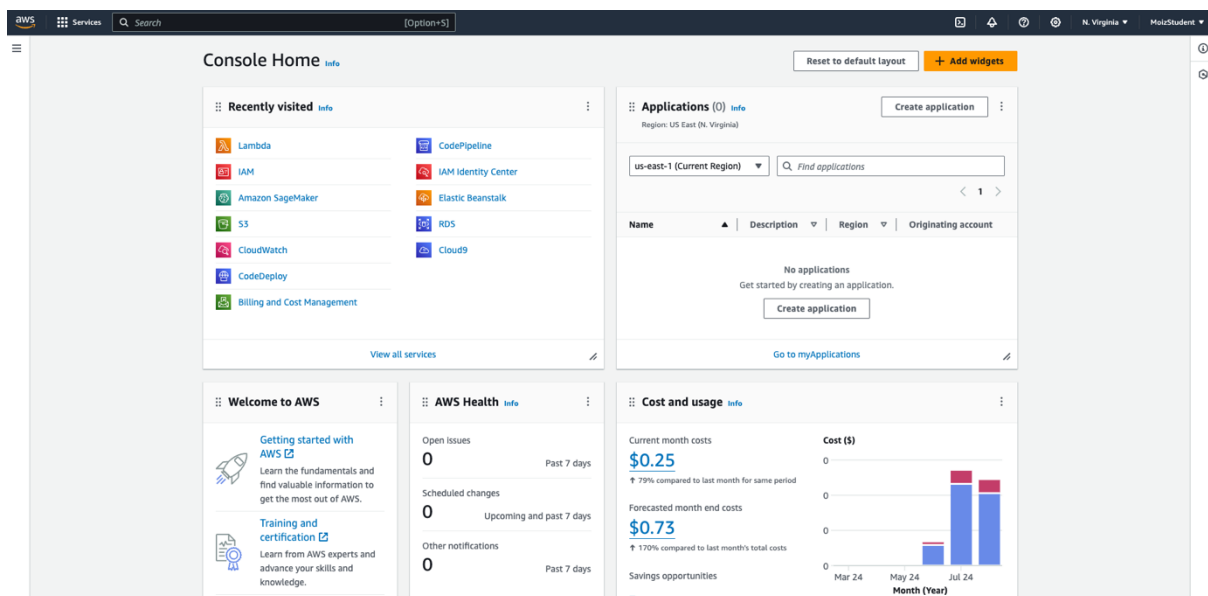


Figure 1: Creation of AWS Account

3.2 Packages & Libraries Used

- **NumPy:** For math operations and array manipulation.
- **Pandas:** For data handling and analysis.
- **TensorFlow/ Keras:** For building and training neural network models.
- **Matplotlib/Seaborn:** For graph and creating visualizations.
- **Scikit-learn:** For machine learning tasks like data pre-processing and evaluation.
- **Boto3:** AWS SDK for Python
- **Sequential, Dense, LSTM, Bi-Directional:** Components and layers in Keras for building neural network architectures.

The image shows a Jupyter Notebook interface. At the top, the title bar says "Minimizingcoldstart" and "Last Checkpoint: a few seconds ago (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar has icons for file operations, running, and saving. The code cell contains the following Python code:

```
In [1]: # Importing the necessary Libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Bidirectional
from datetime import datetime, timedelta
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import boto3
import pandas as pd
```

Figure 2: Importing the Libraries

4 Phases

This section has the detailed Implementation methodology for the project

4.1 Data Generation

- The Dataset used in this study is system generated as the machine learning mode is trained on the real invocations, so the study utilizes a synthetic dataset.

The image shows a VS Code editor with a Python script named "lambda_function.py". The script defines a function "generate_synthetic_data" that generates synthetic data records. The code is as follows:

```
def generate_synthetic_data(num_records):
    start_time = datetime(2024, 6, 1, 0, 0, 0)
    function_ids = ['FuncA', 'FuncB', 'FuncC', 'FuncD']

    data = []
    current_time = start_time

    for i in range(num_records):
        function_id = random.choice(function_ids)

        # Introduce variations in behavior for different functions
        if function_id == 'FuncA':
            cpu_usage = np.random.normal(30, 5)
            memory_usage = np.random.normal(200, 50)
        elif function_id == 'FuncB':
            cpu_usage = np.random.normal(20, 5)
            memory_usage = np.random.normal(150, 30)
        elif function_id == 'FuncC':
            cpu_usage = np.random.normal(40, 5)
            memory_usage = np.random.normal(250, 60)
        else: # FuncD
            cpu_usage = np.random.normal(25, 5)
            memory_usage = np.random.normal(100, 20)

        request_size = np.random.normal(500, 100)
        response_time = cpu_usage * 2 + np.random.normal(50, 10) # Some correlation between CPU usage and response time
        cold_start = np.random.choice([0, 1], p=[0.9, 0.1]) # chance of a cold start

        cpu_usage = max(10, min(cpu_usage, 50))
        memory_usage = max(64, min(memory_usage, 256))
        request_size = max(100, min(request_size, 1000))
        response_time = max(100, min(response_time, 300))

        data.append([
            current_time,
            function_id,
            cpu_usage,
            memory_usage,
            request_size,
            response_time,
            cold_start
        ])

        current_time += timedelta(seconds=1)

    columns = ['Timestamp', 'FunctionID', 'CPUUsage (%)', 'MemoryUsage (MB)', 'RequestSize (KB)', 'ResponseTime (ms)', 'ColdStart (0/1)']
```

Figure 3: Synthetic Dataset Generation

- Dataset that is generated is stored in the S3 bucket named as “historical.csv” as showed in Figure 4.

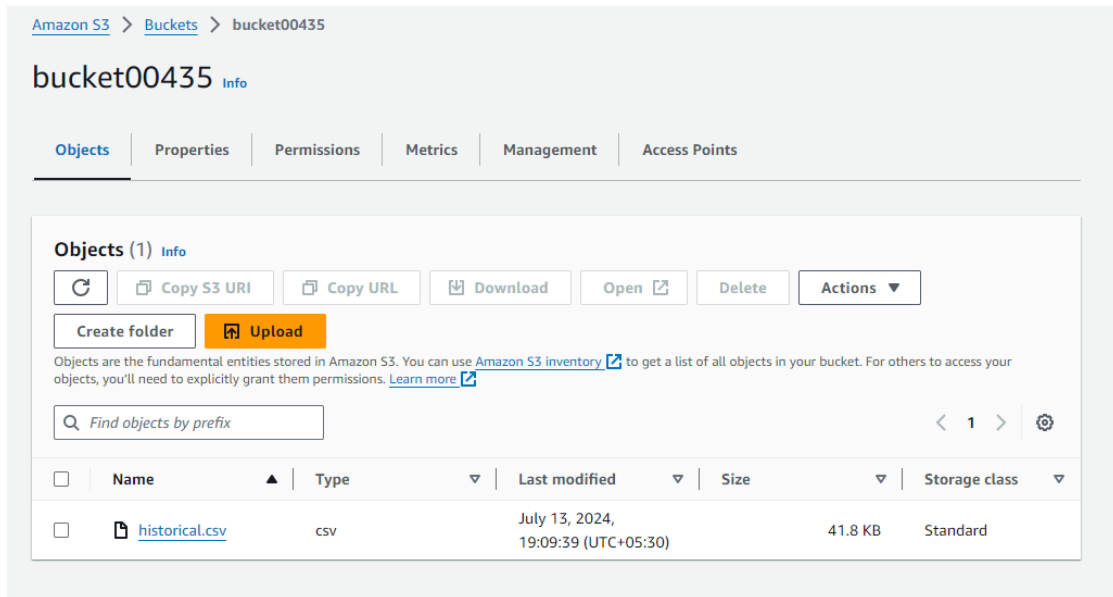


Figure 4: Dataset Stored Location

- From S3 bucket csv file is being downloaded in Amazon Sagemaker in Figure 5.

```
In [2]: import boto3
import pandas as pd

s3_client = boto3.client('s3')
bucket_name = 'bucket00435'
file_key = 'historical.csv'

# Download the file from S3
s3_client.download_file(bucket_name, file_key, 'historical.csv')

# Load the data into a pandas DataFrame
df = pd.read_csv('historical.csv')
print(df.head())
```

	Timestamp	FunctionID	CPUUsage (%)	MemoryUsage (MB)	\
0	2024-06-01 00:00:00	FuncA	30.561682	227.286844	
1	2024-06-01 00:00:01	FuncA	29.381101	199.124056	
2	2024-06-01 00:00:02	FuncA	34.583019	110.234565	
3	2024-06-01 00:00:03	FuncC	45.594580	256.000000	
4	2024-06-01 00:00:04	FuncC	43.940998	256.000000	

	RequestSize (KB)	ResponseTime (ms)	ColdStart (0/1)
0	422.467181	106.783783	0
1	567.574967	105.540158	0
2	562.496880	121.818300	0
3	472.139546	138.608779	0
4	714.367864	150.896196	0

Figure 5: Dataset Information

4.2 Data Pre-processing

- Loading the dataset from S3 into a Pandas DataFrame to perform pre-processing. Perform cleaning, removing outliers and transformations to prepare the data for model training.

```
In [12]: #outlier Detection
import matplotlib.pyplot as plt
import seaborn as sns

# Box plot for each numerical column
plt.figure(figsize=(15, 10))
sns.boxplot(data=df[['CPUUsage (%)', 'MemoryUsage (MB)', 'RequestSize (KB)', 'ResponseTime (ms)']])
plt.show()
```

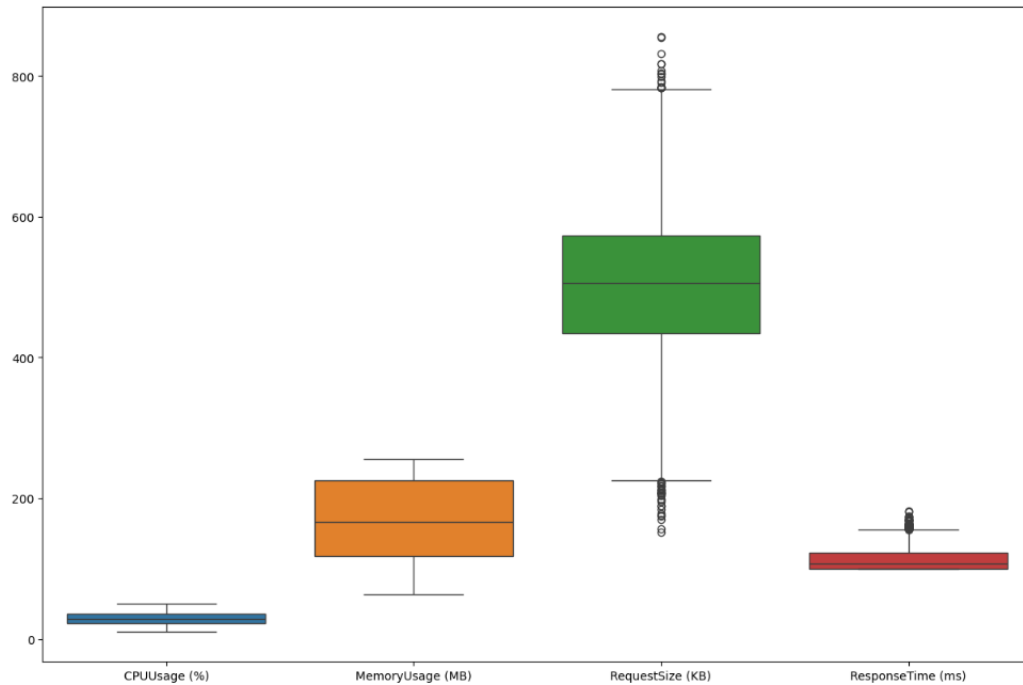


Figure 6: Outlier Detection and Cleaning

```
In [16]: # Identify outliers
outliers = (df[['CPUUsage (%)', 'MemoryUsage (MB)', 'RequestSize (KB)', 'ResponseTime (ms)']] < (Q1 - 1.5 * IQR)) |
df_outliers = df[outliers.any(axis=1)]
print(df_outliers.head(3))
```

Timestamp	FunctionID	CPUUsage (%)	MemoryUsage (MB)	\
2024-06-01 00:00:19	FuncC	50.000000	256.0	
2024-06-01 00:01:28	FuncC	46.738806	256.0	
2024-06-01 00:03:08	FuncC	50.000000	256.0	

Timestamp	RequestSize (KB)	ResponseTime (ms)	ColdStart (0/1)
2024-06-01 00:00:19	493.808821	167.073657	1
2024-06-01 00:01:28	691.935942	156.010319	1
2024-06-01 00:03:08	463.646208	161.613317	0

Figure 7: Identified Outlier

4.3 Training Dummy Machine Learning Model

- Run the code in the Jupiter notebook named “Minimizingcoldstart.ipynb” in AWS Sagemaker for the results.
- Ensure the accuracy of the dummy model and generation of “initial_results.csv” in your s3 bucket

```
In [32]: # Build the model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Bidirectional, LSTM, Dense
model = Sequential()
model.add(Bidirectional(LSTM(50, return_sequences=True), input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Bidirectional(LSTM(50)))
model.add(Dense(1, activation='relu'))
optimizer = Adam(clipvalue=1.0)
model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

2024-08-08 22:49:42.477424: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:282] failed call to cuIni
t: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected

In [33]: # Train the model
history = model.fit(X_train, y_train, epochs=8, batch_size=10, validation_data=(X_test, y_test))

Epoch 1/8
560/560 [=====] - 37s 12ms/step - loss: 0.9137 - accuracy: 0.8989 - val_loss: 1.0099 - val_
accuracy: 0.9064
Epoch 2/8
560/560 [=====] - 4s 7ms/step - loss: 0.4112 - accuracy: 0.9013 - val_loss: 0.3194 - val_a
ccuracy: 0.9064
Epoch 3/8
560/560 [=====] - 3s 6ms/step - loss: 0.3232 - accuracy: 0.9023 - val_loss: 0.3144 - val_a
ccuracy: 0.9064
Epoch 4/8
560/560 [=====] - 4s 6ms/step - loss: 0.3224 - accuracy: 0.9023 - val_loss: 0.3137 - val_a
ccuracy: 0.9064
Epoch 5/8
560/560 [=====] - 4s 7ms/step - loss: 0.3225 - accuracy: 0.9023 - val_loss: 0.3200 - val_a
ccuracy: 0.9064
Epoch 6/8
560/560 [=====] - 4s 6ms/step - loss: 0.3232 - accuracy: 0.9023 - val_loss: 0.3231 - val_a
ccuracy: 0.9064
Epoch 7/8
560/560 [=====] - 4s 7ms/step - loss: 0.3225 - accuracy: 0.9023 - val_loss: 0.3213 - val_a
ccuracy: 0.9064
Epoch 8/8
560/560 [=====] - 4s 7ms/step - loss: 0.3319 - accuracy: 0.9023 - val_loss: 0.6732 - val_a
ccuracy: 0.9064

44/44 [=====] - 2s 2ms/step
Model Accuracy: 90.64%
```

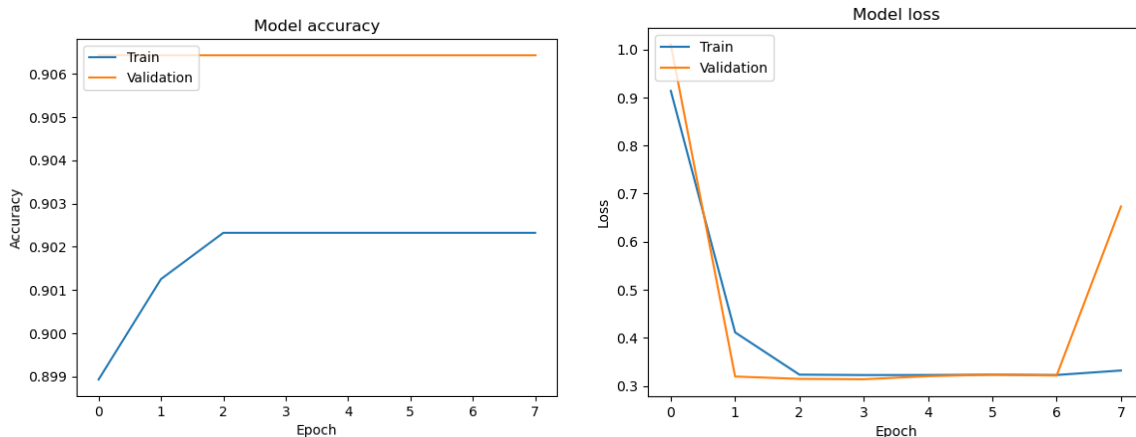


Figure 8: Model Training

4.4 Train Optimized Machine Learning Model

- Browse recent invocations for the “gendata” lambda functions to manually save it as csv
- View it in CloudWatch logs and Run the query Figure

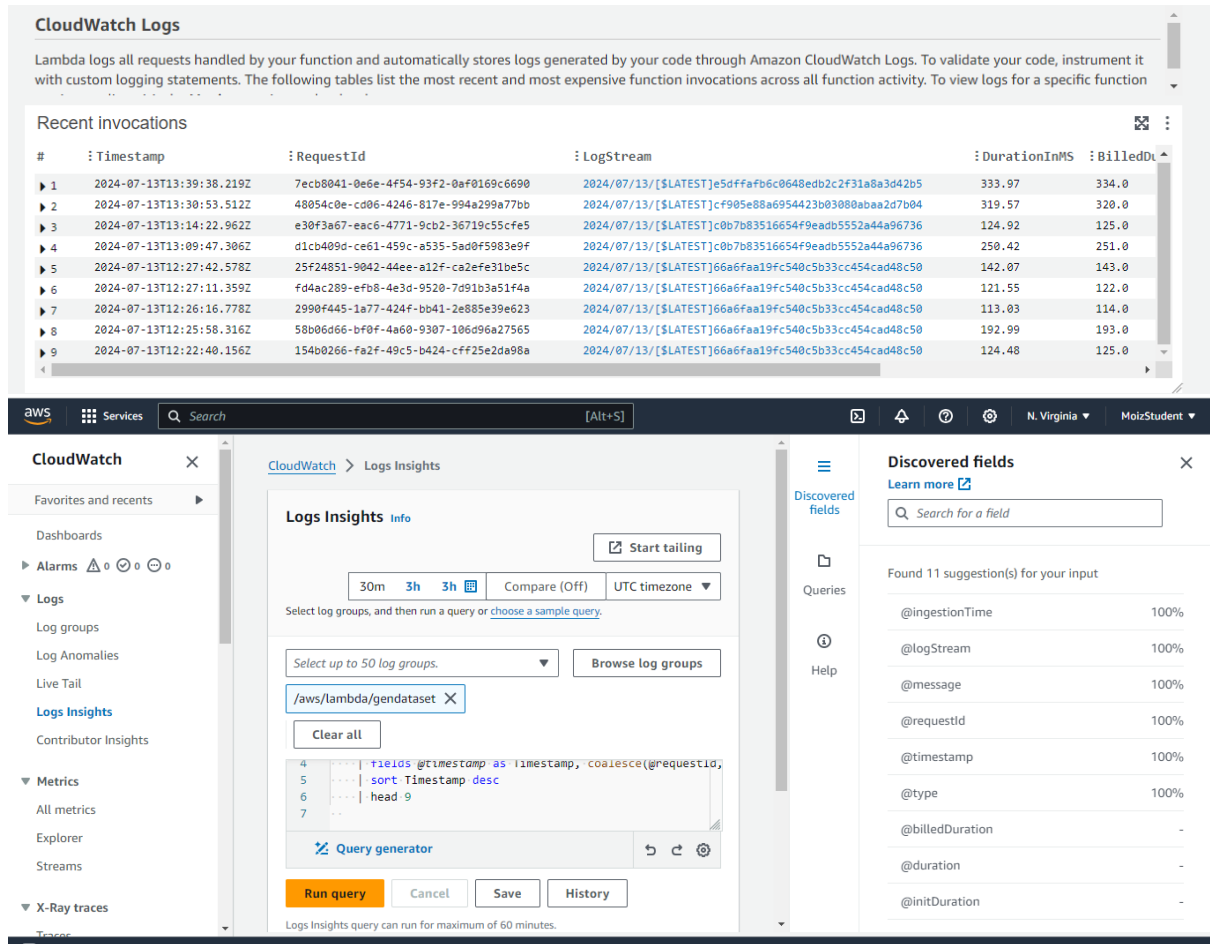


Figure 9: CloudWatch Log

- Export the data in csv format to your Computer

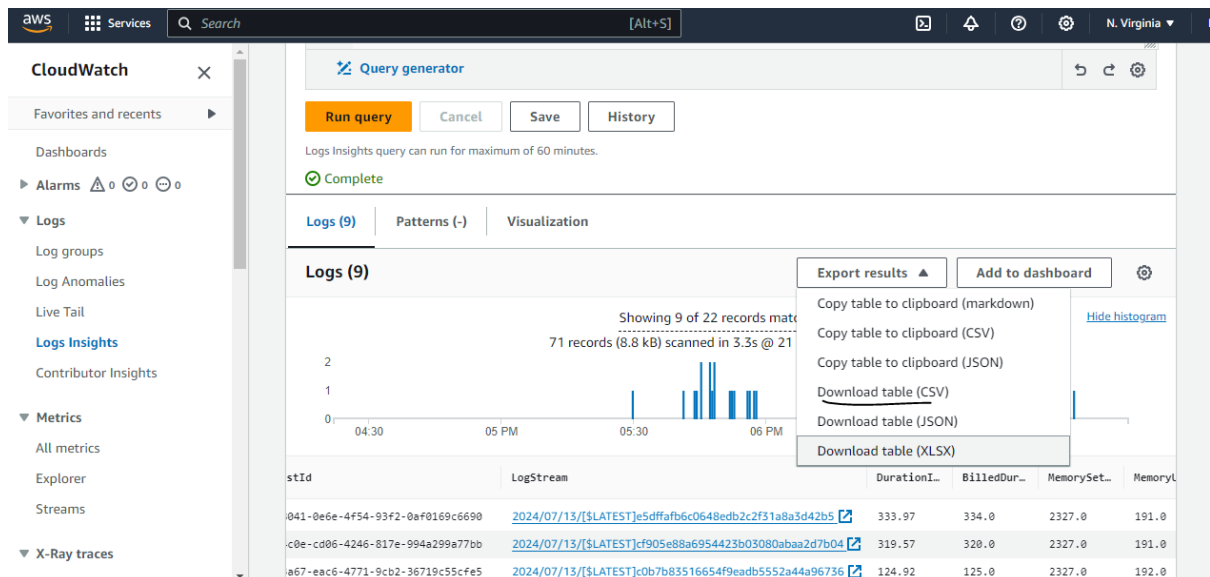


Figure 10

- Upload the csv to your S3 Bucket

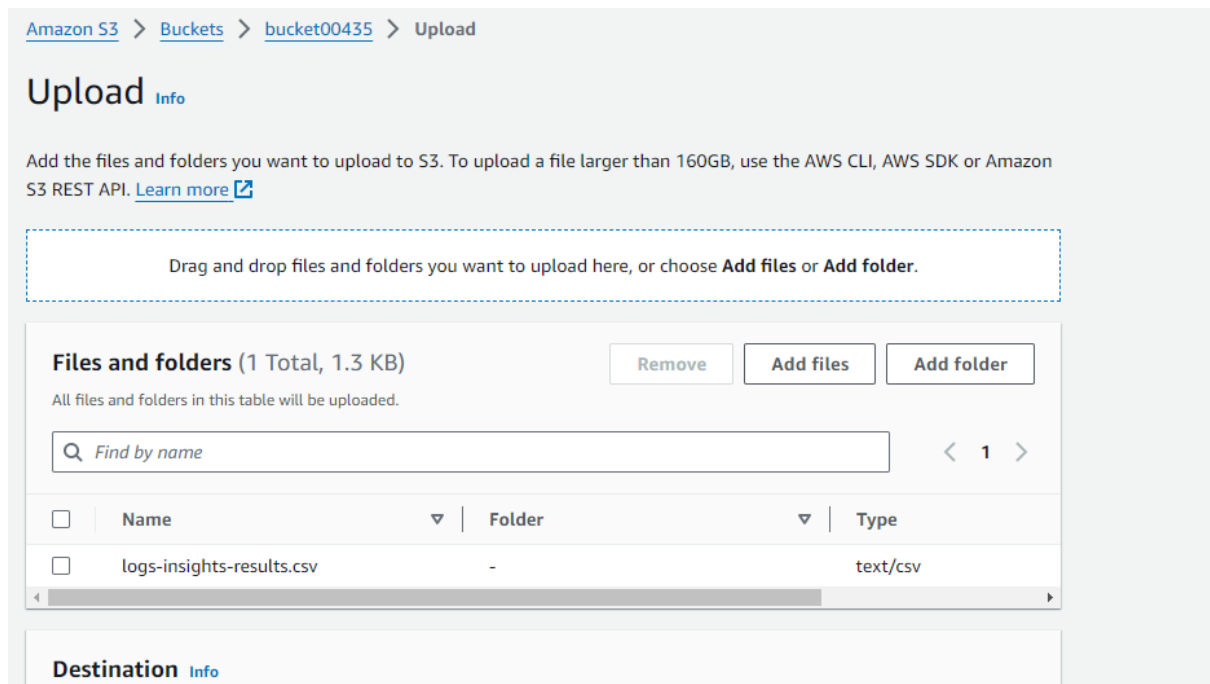


Figure 11

- Ensure the file has been uploaded successfully to S3

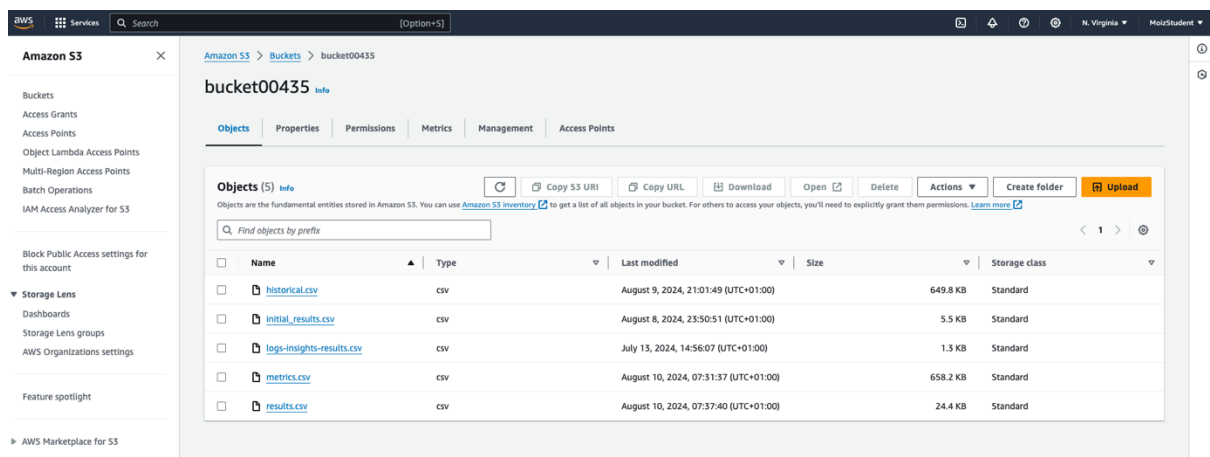


Figure 12

- Run all the cells in “Optimizing Cold” Start notebook in sagemaker

The screenshot shows a Jupyter notebook interface with the title 'OptimizingCold Start'. The notebook has two cells. The first cell, labeled 'In [137]:', contains code for importing various libraries including pandas, numpy, tensorflow, keras, sklearn, and matplotlib. The second cell, labeled 'In [140]:', contains code for downloading a file named 'metrics.csv' from an S3 bucket and loading it into a pandas DataFrame. The output of the second cell shows the first five rows of the DataFrame, which include columns for Timestamp, ProcessID, StartTime, EndTime, MemoryUtilization, CPUUtilization, ProcessingSpeed, ReadBytes, WrittenBytes, IncomingBytes, OutgoingBytes, and DiskIO.

```
In [137]: # Importing the necessary Libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Bidirectional
from datetime import datetime, timedelta
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import boto3
import uuid

In [140]: s3_client = boto3.client('s3')
bucket_name = 'bucket00435'
file_key = 'metrics.csv'

# Download the file from S3
s3_client.download_file(bucket_name, file_key, 'metrics.csv')

# Load the data into a pandas DataFrame
df = pd.read_csv('metrics.csv')
print(df.head())
```

	Timestamp	ProcessID	StartTime	EndTime
0	31-07-2024 13:33	8596	31-07-2024 13:33	31-07-2024 13:33
1	31-07-2024 13:33	8596	31-07-2024 13:33	31-07-2024 13:33
2	31-07-2024 13:33	8596	31-07-2024 13:33	31-07-2024 13:33
3	31-07-2024 13:33	8596	31-07-2024 13:33	31-07-2024 13:33
4	31-07-2024 13:33	8596	31-07-2024 13:33	31-07-2024 13:33

	MemoryUtilization	CPUUtilization	ProcessingSpeed	ReadBytes
0	29.66	4.52	826860	932149
1	31.72	4.57	826952	932689
2	33.37	4.61	828464	935466
3	35.21	4.65	828927	935914
4	37.26	4.69	829390	936362

	WrittenBytes	IncomingBytes	OutgoingBytes	DiskIO
0	753398	143289	0.28	63
1	753398	143289	0.28	63
2	753398	143473	0.58	63
3	753398	143481	0.58	63
4	753398	144593	0.58	63

Figure 13

- Test the optimization function in lambda to pre-warm the main function with the help of Genetic Algorithm

The screenshot shows the AWS Lambda console with a function named 'lambda_function'. The code is written in Python and implements a genetic algorithm. It includes functions for selection, crossover, mutation, and a main genetic algorithm function. The code is as follows:

```
1 import boto3
2 import json
3 from numpy.random import randint, rand
4
5 # Initialize AWS SDK clients
6 cloudwatch_logs = boto3.client('logs')
7
8 # Genetic algorithm functions
9 def onemax(x):
10     return -sum(x)
11
12 def selection(pop, scores, k=3):
13     selection_ix = randint(len(pop))
14     for ix in randint(0, len(pop), k-1):
15         if scores[ix] < scores[selection_ix]:
16             selection_ix = ix
17     return pop[selection_ix]
18
19 def crossover(p1, p2, r_cross):
20     c1, c2 = p1.copy(), p2.copy()
21     if rand() < r_cross:
22         pt = randint(1, len(p1)-2)
23         c1 = p1[:pt] + p2[pt:]
24         c2 = p2[:pt] + p1[pt:]
25     return [c1, c2]
26
27 def mutation(bitstring, r_mut):
28     for i in range(len(bitstring)):
29         if rand() < r_mut:
30             bitstring[i] = 1 - bitstring[i]
31
32 def genetic_algorithm(objective, n_bits, n_iter, n_pop, r_cross, r_mut):
33     pop = [randint(0, 2, n_bits).tolist() for _ in range(n_pop)]
34     best, best_eval = 0, objective(pop[0])
35     for gen in range(n_iter):
36         scores = [objective(c) for c in pop]
37         for i in range(n_pop):
38             if scores[i] < best_eval:
39                 best, best_eval = pop[i], scores[i]
40                 print("<td>New best f(%s) = %.3f" % (gen, pop[i], scores[i]))
41         selected = [Selection(pop, scores) for _ in range(n_pop)]
42         children = []
43         for i in range(0, n_pop, 2):
44             p1, p2 = selected[i], selected[i+1]
45             for c in crossover(p1, p2, r_cross):
46                 mutation(c, r_mut)
47                 children.append(c)
48         pop = children
49     return [best, best_eval]
```

Figure 14

- Optimized cold start time and Original Invocation Time

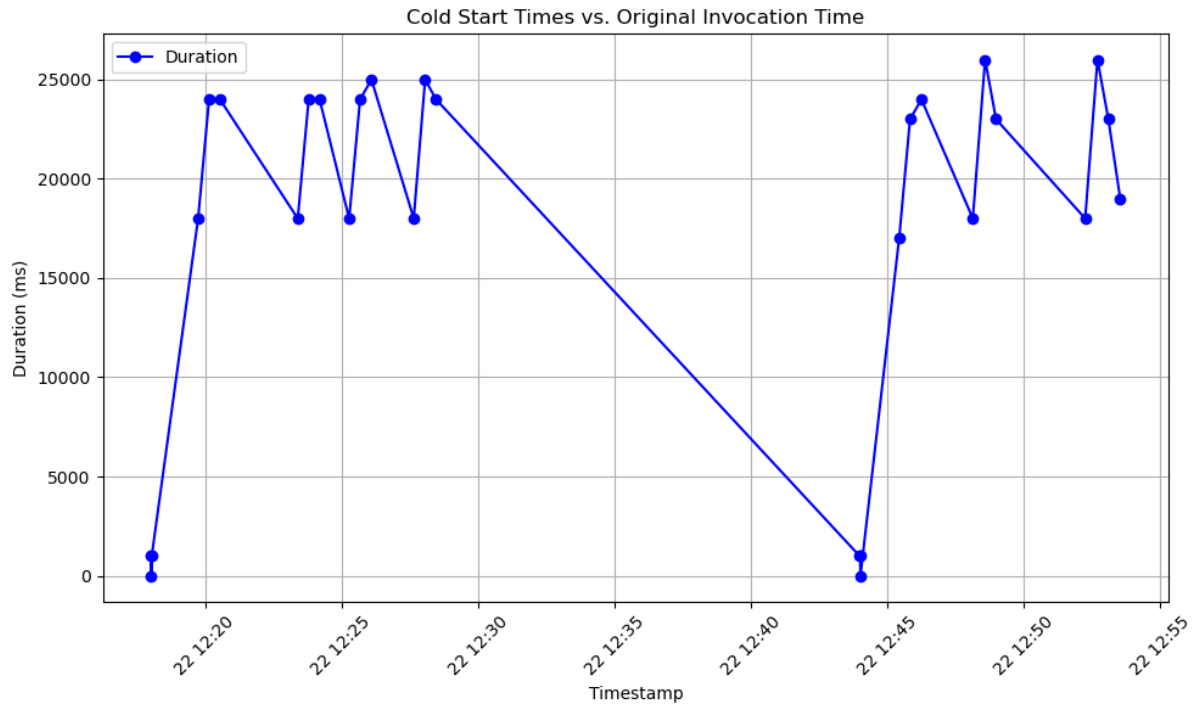


Figure 15

References

Amazon Web Services, Inc. (n.d.). *AWS Lambda Documentation*. Retrieved from <https://docs.aws.amazon.com/lambda/>

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357-362. Retrieved from <https://numpy.org/>

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95. Retrieved from <https://matplotlib.org/>