Configuration Manual

MSc Research Project
MSc Cloud Computing

Anmol Singla
Student ID: x20259891

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anmol Singla		
Student ID:	X20259891		
Programme:	MSc in Cloud Computing	**Year:**	2023
Module:	MSc Research Project		
Lecturer:	Rashid Mijumbi		
Submission Due Date:	14/12/2023		
Project Title:	Optimizing Long-Short Term Memory (LSTM) Algorithm for Enhanced Energy Efficiency and Green Computing in Cloud Environments		
Word Count:	477		
Page Count:	7		

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: Anmol Singla

Date: 14h December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
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.	□

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):	

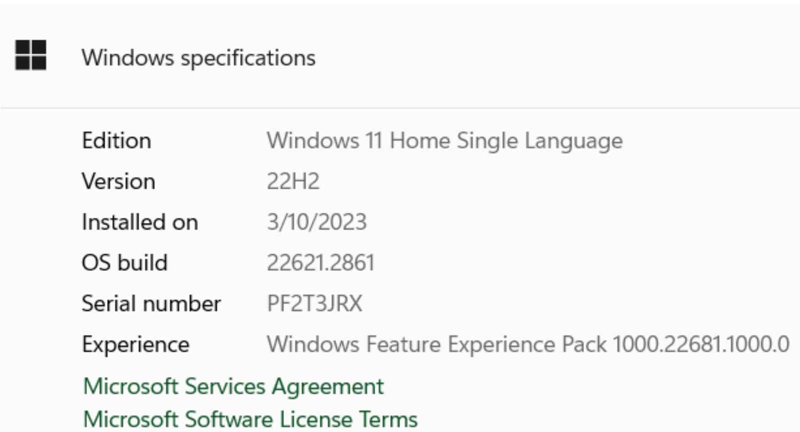
Optimizing Long-Short Term Memory (LSTM) Algorithm for Enhanced Energy Efficiency and Green Computing in Cloud Environments

Anmol Singla
X20259891

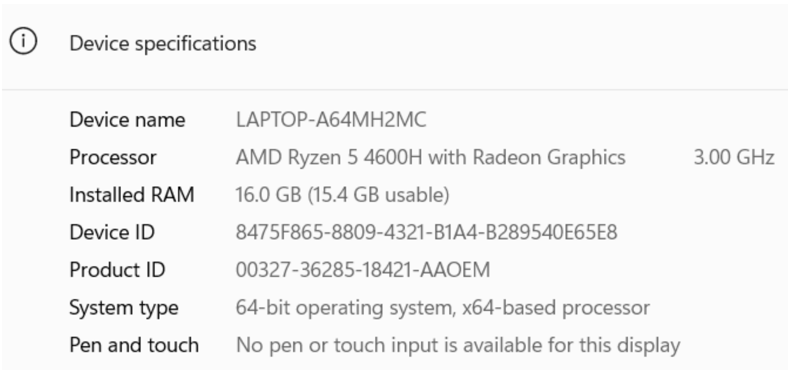
1 Introduction

The purpose of this configuration document is to present the essential information required for conducting the proposed research study. The implementation was executed on a conventional laptop computer. This manual presents a comprehensive set of instructions for replicating the thesis work in a systematic and sequential manner. This document provides an explanation for all of the artifacts that are included and attached to the report. This manual provides an overview of code snippets related to data collection, model building, experiments, and result evaluation.

2 Hardware Requirements



Windows Specification



Device Specification

3 Software Requirement

For creating the script following tools and softwares were required

Programming Language	Python3.6
Tools	Google Collab

4 Dataset Collection

The dataset has been taken from the online public repositry i.e. kaggle:

https://www.kaggle.com/datasets/abdurraziq01/cloud-computing-performance-metrics

5 Importing libraries

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from IPython.display import display, Math
```

Importing required Libraries

6 Importing Dataset

```
df = pd.read_csv('vmCloud_data.csv')
```

```
df.head()
```

	vm_id	timestamp	cpu_usage	memory_usage	network_traffic	power_consumption	num_executed_instructions	execution_time	energy_efficiency	ta
0	c5215826-6237-4a33-9312-72c1df909881	2023-01-25 09:10:54	54.881350	78.950861	164.775973	287.808986	7527.0	69.345575	0.553589	
1	29690bc6-1f34-403b-b509-a1ecb1834fb8	2023-01-26 04:46:34	71.518937	29.901883	NaN	362.273569	5348.0	41.396040	0.349856	
2	2e55abc3-5bad-46cb-b445-a577f5e9bf2a	2023-01-13 23:39:47	NaN	92.709195	203.674847	231.467903	5483.0	24.602549	0.796277	
3	e672e32f-c134-4fbc-992b-34eb63bef6bf	2023-02-09 11:45:49	54.488318	88.100960	NaN	195.639954	5876.0	16.456670	0.529511	
4	f38b8b50-6926-4533-be4f-89ad11624071	2023-06-14 08:27:26	42.365480	NaN	NaN	359.451537	3361.0	55.307992	0.351907	

7 Data pre-processing and transformation

```
null_values = df.isnull().sum()
print(null_values)
```

```
vm_id                          200638
timestamp                      200666
cpu_usage                      199038
memory_usage                   200510
network_traffic                199481
power_consumption              200271
num_executed_instructions      199686
execution_time                 199827
energy_efficiency              200042
task_type                      199962
task_priority                  199433
task_status                    200306
dtype: int64
```

Missing values

```
In [6]: # Creating New Dataframe
new_df = df[['cpu_usage', 'memory_usage', 'network_traffic', 'power_consumption', 'num_executed_instructions', 'execution_time',
```

```
In [7]: # Fill numerical columns with the median
numerical_columns = ['cpu_usage', 'memory_usage', 'network_traffic', 'power_consumption',
                     'num_executed_instructions', 'execution_time', 'energy_efficiency']
for col in numerical_columns:
    df[col] = new_df[col].fillna(new_df[col].median())

# Fill categorical columns with the mode
categorical_columns = ['task_type', 'task_priority', 'task_status']
for col in categorical_columns:
    new_df[col] = new_df[col].fillna(new_df[col].mode()[0])
```

Handling missing values

8 Label Encoding

```
In [12]: from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()

# List of non-numerical variables
non_numerical_vars = ['task_type', 'task_priority', 'task_status']

# Apply Label Encoding to each non-numerical column
for col in non_numerical_vars:
    new_df[col] = le.fit_transform(new_df[col])

# Compute the correlation matrix
correlation_matrix = new_df.corr()

# Display the correlation matrix
print(correlation_matrix)
```

Label Encoding

9 Data Visualization

```
In [13]: import seaborn as sns
         import matplotlib.pyplot as plt
         # Plotting the heatmap
         plt.figure(figsize=(12, 8))
         sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
         plt.title('Correlation Matrix Heatmap')
         plt.show()
```

Correlation Matrix

```
In [14]: plt.figure(figsize=(10, 6))
         sns.histplot(df['cpu_usage'], bins=30, kde=True)
         plt.title('CPU Usage Distribution')
         plt.xlabel('CPU Usage')
         plt.ylabel('Frequency')
         plt.show()
```

CPU Usage Distribution

```
In [15]: plt.figure(figsize=(10, 6))
         sns.boxplot(x='task_priority', y='execution_time', data=df)
         plt.title('Execution Time for Different Task Priorities')
         plt.xlabel('Task Priority')
         plt.ylabel('Execution Time')
         plt.show()
```

Execution time for different task priorities

```
In [16]: plt.figure(figsize=(12, 7))
         df.groupby('task_type')['energy_efficiency'].mean().plot(kind='bar')
         plt.title('Average Energy Efficiency by Task Type')
         plt.xlabel('Task Type')
         plt.ylabel('Average Energy Efficiency')
         plt.show()
```

Average Energy Efficiency by Task Type

```
In [17]: # Creating a pivot table for the heatmap
         pivot_table = df.pivot_table(values=['cpu_usage', 'memory_usage'], index='task_priority', aggfunc='mean')

         plt.figure(figsize=(10, 6))
         sns.heatmap(pivot_table, annot=True, cmap='viridis')
         plt.title('Average CPU and Memory Usage by Task Priority')
         plt.show()
```

Average CPU and Memory usage

10 Data Splitting and Model Building

```python
In [19]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import numpy as np

# Data Preprocessing
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(small_df[['cpu_usage', 'memory_usage', 'network_traffic', 'power_consumption']])

# Create sequences from data
def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(len(data)-seq_length-1):
        x = data[i:(i+seq_length)]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 5  # Example sequence length
X, y = create_sequences(scaled_data, seq_length)

# Splitting the Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Data Splitting

```python
# Building the LSTM Model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(seq_length, X.shape[2])))
model.add(Dense(1))

# Compiling the Model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the Model
model.fit(X_train, y_train, epochs=50, validation_split=0.1, batch_size=64)

# Evaluation
predictions = model.predict(X_test)
```

Model Building

```python
In [20]: from keras.models import Sequential
from keras.layers import LSTM, Dense

def create_lstm_model(units=50, optimizer='adam'):

    model = Sequential()
    model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

```python
In [21]: from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV

model = KerasRegressor(build_fn=create_lstm_model, verbose=0)

param_grid = {
    'epochs': [10],
    'batch_size': [32],
    'units': [50],
    'optimizer': ['adam']
}

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, error_score='raise')
grid_result = grid.fit(X_train, y_train)
```

Creating Model

```
In [27]: from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# 'model_optimized' is trained LSTM model
# Generate predictions
y_pred = model_optimized.predict(X_test)

# Reshape y_pred to ensure it is a 2D array with one column
y_pred = np.reshape(y_pred, (-1, 1))

# Reshape y_test to have the same format as y_pred
y_test_single_output = y_test[:, 0].reshape(-1, 1)

# Calculate MSE
mse = mean_squared_error(y_test_single_output, y_pred)
print("MSE on Test Set:", mse)

# Plotting the training loss (MSE)
plt.plot(history.history['loss'], label='Training Loss')
if 'val_loss' in history.history:
    plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss (MSE) Over Epochs')
plt.ylabel('MSE Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

Finding model loss over epochs

```
In [28]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

def build_model(input_shape):
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))

    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

model = build_model(input_shape=(X_train.shape[1], X_train.shape[2]))
```

Building Custom LSTM

```
In [34]: print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)

X_test shape: (3999, 5, 4)
y_test shape: (15996, 1)

In [35]: y_pred = model.predict(X_test)
print("y_pred shape:", y_pred.shape)

125/125 [==============================] - 1s 4ms/step
y_pred shape: (3999, 1)

In [36]: y_pred = y_pred.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

print("Reshaped y_pred shape:", y_pred.shape)
print("Reshaped y_test shape:", y_test.shape)

Reshaped y_pred shape: (3999, 1)
Reshaped y_test shape: (15996, 1)

In [38]: # Truncate y_test to match the shape of y_pred
y_test_truncated = y_test[:3999]

# Calculate MSE
mse = mean_squared_error(y_test_truncated, y_pred)
print("MSE on Test Set:", mse)

MSE on Test Set: 0.07308321178428932
```

Reshaping data

```
In [39]: import matplotlib.pyplot as plt

         plt.plot(history.history['loss'], label='Training Loss')
         plt.plot(history.history['val_loss'], label='Validation Loss')
         plt.title('Model Loss (MSE) Over Epochs')
         plt.ylabel('MSE Loss')
         plt.xlabel('Epoch')
         plt.legend()
         plt.show()
```

Visualizing Model Loss over Epochs