Workload prediction for cloud services by using a hybrid neural network model-Configuration Mannual

MSc Research Project
MSc in Cloud Computing

Preeti Rawat
Student ID: 20233507

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Preeti Rawat
Student ID:	20233507
Programme:	MSc in Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Shivani Jaswal
Submission Due Date:	15/08/2022
Project Title:	Workload prediction for cloud services by using a hybrid neural network model-Configuration Mannual
Word Count:	761
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:	Preeti Rawat
Date:	18th September 2022

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):	

Workload prediction for cloud services by using a hybrid neural network model-Configuration Mannual

Preeti Rawat
20233507

1 Introduction

This configuration manual helps the reader in understanding the system setup, system requirements, and specification of the hardware and software used during the research. It explains the steps to be followed to run the research project: Workload prediction for cloud services by using a hybrid neural network model.

2 System configuration

2.1 Hardware Configuration

- Model: HP Pavillion Laptop 14 – dv0xxx
- Processor: Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- Operating System: Windows 10
- RAM: 16.0 GB (15.8 GB usable)
- Hard Disk: 476.94 GB

3 Software Installation

3.1 Python Installation

To implement the proposed model and extract the results, python is used. Python can be downloaded from `https://www.python.org/downloads/`. The required version of python is 3.9.13 and is shown in Figure 1.

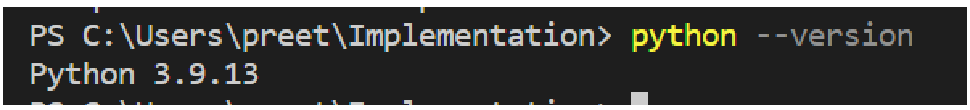


Figure 1: Python's Version

3.2 Required Python Libraries

Libraries shown in Figure 2 are imported while implementing the project.

```
import numpy as np
import sklearn.metrics as metrics
import math
import random
import matplotlib.pyplot as plt
from statistics import mean
import csv
import pandas as pd
import sys
import boto3
from botocore.exceptions import ClientError
import os
from io import StringIO
from scipy.signal import savgol_filter
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
import pywt
```

Figure 2: Imported Python libraries in the Project

matplot library is used to plot the graphs. Pywt library is responsible for carrying out the wavelet transformation. Scipy library is used to include savgol_filter, to perform smoothening of the input series. Sklearn is used for incorporating the SVR algorithm whereas Tensorflow Keras is used for using the LSTM algorithm in the project implementation.

To install certain required python libraries, below are the commands for Windows operating system:

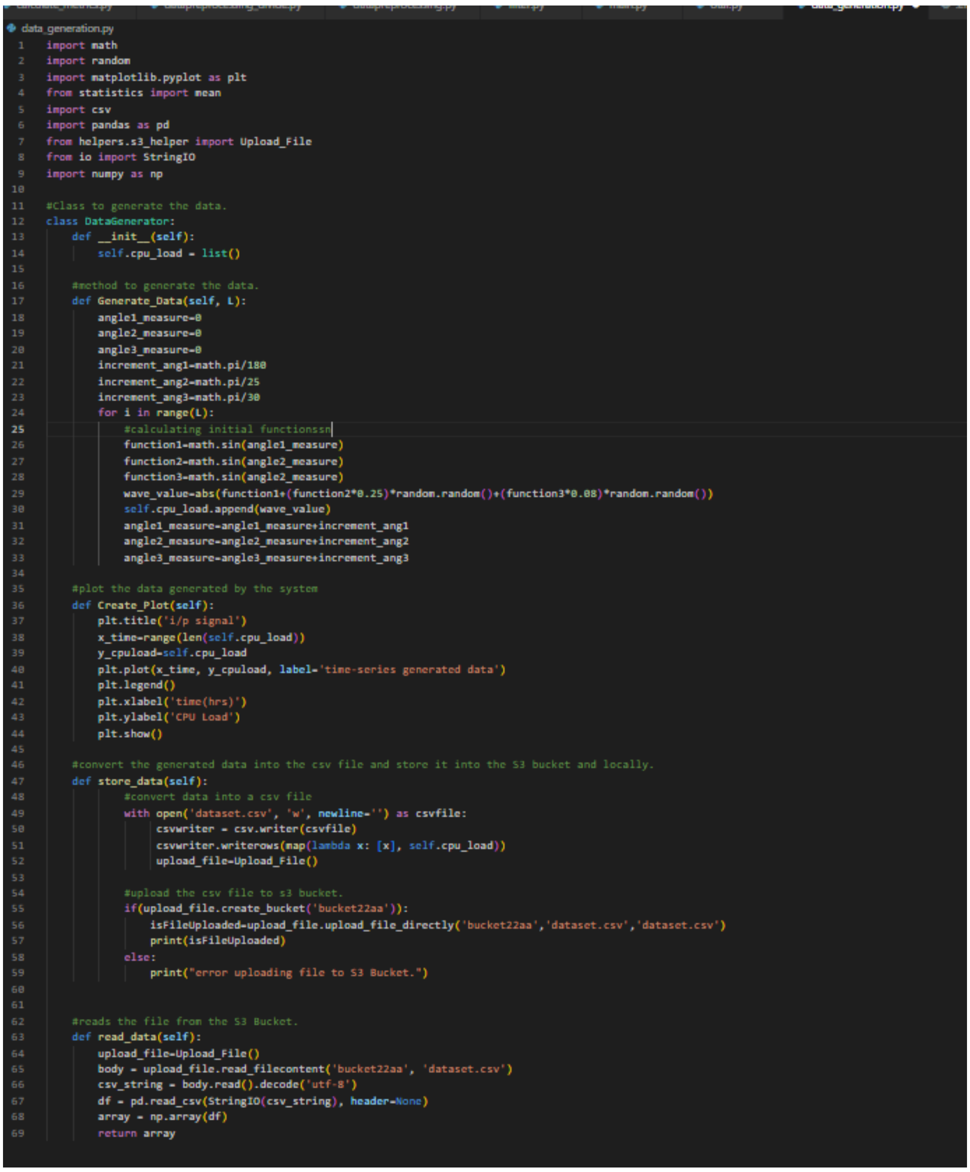
- python -m pip install -upgrade pip
- python -m pip install matplotlib
- python -m pip install numpy
- python -m pip install pandas
- python -m pip install sklearn
- python -m pip install tensorflow

- python -m pip install pywt
- python -m pip install boto3

4 Implementation and Analysis

4.1 Data Generation and Storage

For the research, synthetic data is generated which includes pseudo-randomness. The proposed model trains this data and predicts the workload for the next time slot. Code for data generation and storage is present in the data_generation.py file and shown in Figure 3. Generated data is stored in the dataset.csv file, which in turn is stored in the S3 bucket so that later on data can be fetched from the S3 Bucket directly.



```
data_generation.py
import math
import random
import matplotlib.pyplot as plt
from statistics import mean
import csv
import pandas as pd
from helpers.s3_helper import Upload_File
from io import StringIO
import numpy as np

#Class to generate the data.
class DataGenerator:
    def __init__(self):
        self.cpu_load = list()

    #method to generate the data.
    def Generate_Data(self, L):
        angle1_measure=0
        angle2_measure=0
        angle3_measure=0
        increment_ang1=math.pi/180
        increment_ang2=math.pi/25
        increment_ang3=math.pi/30
        for i in range(L):
            #calculating initial functionssn
            function1=math.sin(angle1_measure)
            function2=math.sin(angle2_measure)
            function3=math.sin(angle2_measure)
            wave_value=abs(function1+(function2*0.25)*random.random()+(function3*0.08)*random.random())
            self.cpu_load.append(wave_value)
            angle1_measure=angle1_measure+increment_ang1
            angle2_measure=angle2_measure+increment_ang2
            angle3_measure=angle3_measure+increment_ang3

    #plot the data generated by the system
    def Create_Plot(self):
        plt.title('i/p signal')
        x_time=range(len(self.cpu_load))
        y_cpuload=self.cpu_load
        plt.plot(x_time, y_cpuload, label='time-series generated data')
        plt.legend()
        plt.xlabel('time(hrs)')
        plt.ylabel('CPU Load')
        plt.show()

    #convert the generated data into the csv file and store it into the S3 bucket and locally.
    def store_data(self):
            #convert data into a csv file
            with open('dataset.csv', 'w', newline='') as csvfile:
                csvwriter = csv.writer(csvfile)
                csvwriter.writerows(map(lambda x: [x], self.cpu_load))
                upload_file=Upload_File()

            #upload the csv file to s3 bucket.
            if(upload_file.create_bucket('bucket22aa')):
                isFileUploaded=upload_file.upload_file_directly('bucket22aa','dataset.csv','dataset.csv')
                print(isFileUploaded)
            else:
                print("error uploading file to S3 Bucket.")


    #reads the file from the S3 Bucket.
    def read_data(self):
        upload_file=Upload_File()
        body = upload_file.read_filecontent('bucket22aa', 'dataset.csv')
        csv_string = body.read().decode('utf-8')
        df = pd.read_csv(StringIO(csv_string), header=None)
        array = np.array(df)
        return array
```

Figure 3: Data Generation Script

4.2 AWS credentials update

To use the AWS S3 service for the storage of the generated dataset, update the value shown in Figure 4 for the user's AWS account in the .env file. Since visual studio code is

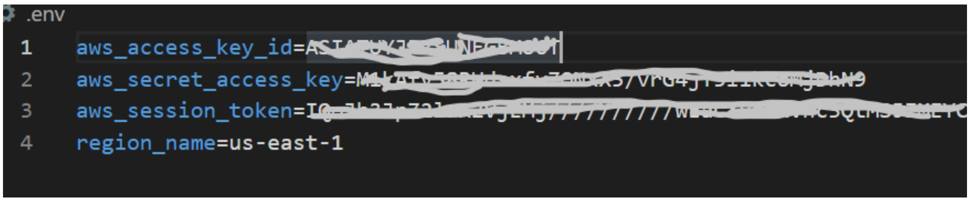


Figure 4: AWS credential's Setting in .env file

used for the development, so, to connect visual studio code to AWS, get credentials file, and update AWS account credentials. Credentials file is shown in Figure 5

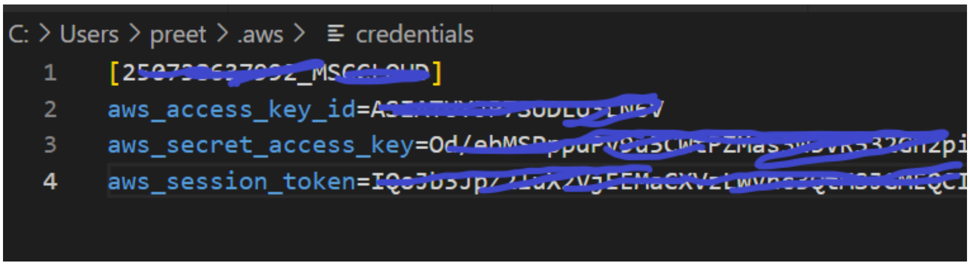


Figure 5: AWS credential's Setting in Credentials file

Credentials shown in Figure 4 and Figure 5 are token based and they usually expire after some hours, so they need to be updated after expiration.

4.3 Steps to run the project

After installing the prerequisite libraries and setting up the AWS credentials, the user can run the project by using the below command:

- python main.py

when this command is run, data is generated by running the data_generation.py script. The plot of the input signal is shown in Figure 6.

Simple LSTM Model: After synthetic data is generated, simple LSTM is applied to the input signal, and metrics are calculated for that. Code in the lstmmodel.py file is executed for training the processed input signal and the file is shown in Figure 7. The results of running simple LSTM are shown in Figure 8 in the command line.

Proposed model: After generating the synthetic time series and smoothing it using SG filters, it is divided into low and high-frequency components using wavelet transformation, and the code shown in wave_transform.py is executed for that. Code of wave_transform is shown in Figure 9. After that, low-frequency components are trained by the SVR model to predict the next 5 values, while high-frequency components are modeled by LSTM to predict the next 5 values. After this by using inverse wavelet

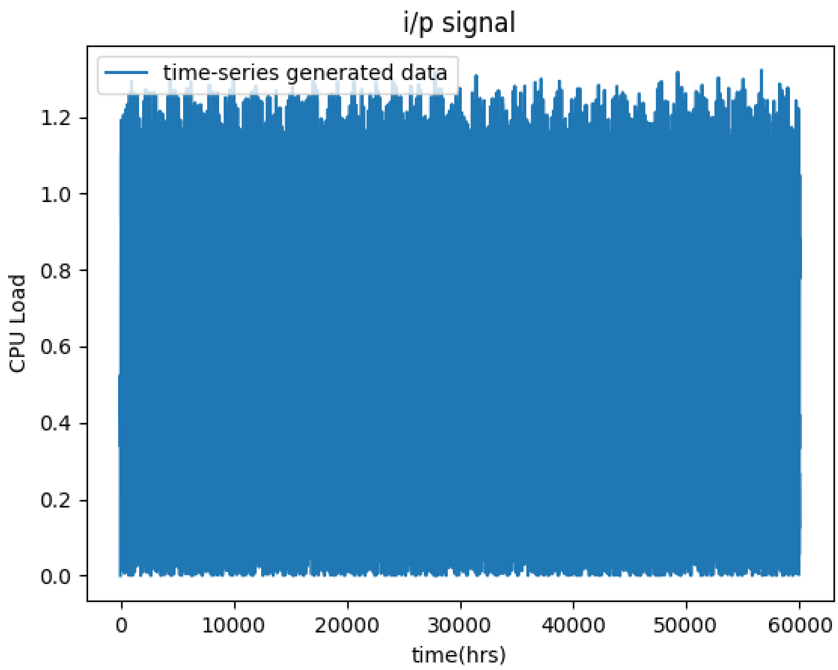


Figure 6: Generated Data

```
lstmmodel.py
import pandas as pd
import numpy as np
import math
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM

#LSTM model to be applied for the high-frequency components in the input signal.
class LSTMModel:
    #predicts the CPU load value for the test set.
    def predict(self,trainX, trainY,testX):
        model = Sequential()
        trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1],1))
        testX = np.reshape(testX, (testX.shape[0], testX.shape[1],1))

        model.add(LSTM(trainY.shape[1], input_shape=(trainX.shape[1],1)))
        model.add(Dense(trainY.shape[1],activation='softmax'))
        model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
        print(model.summary())
        model.fit(trainX, trainY, epochs=5, batch_size=37, verbose = 2, shuffle = False)
        # PREDICTION
        trainPredict = model.predict(trainX)
        testPredict = model.predict(testX)
        return testPredict
```

Figure 7: LSTM algorithm Script

transformation, high and low-frequency components are combined and predicted 10 values are returned. The proposed model is mentioned in the proposed_model.py file and in Figure 10. The performance metrics are calculated after main.py is run. Results are displayed in the command line for the LSTM+SVR hybrid model and show in Figure 11. Metrics reveal that the LSTM+SVR hybrid model performs better compared to

```
PS C:\Users\preet\Implementation> python main.py
2022-08-10 21:32:45.872311: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2022-08-10 21:32:45.872742: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
bucket already exists.
None
2022-08-10 21:32:53.959811: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
2022-08-10 21:32:53.960309: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2022-08-10 21:32:53.962905: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: LAPTOP-NECC9JEH
2022-08-10 21:32:53.963375: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: LAPTOP-NECC9JEH
2022-08-10 21:32:53.963924: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU in
 in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10)                480

 dense (Dense)               (None, 10)                110

=================================================================
Total params: 590
Trainable params: 590
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
109/109 - 3s - loss: 0.3945 - accuracy: 0.0547 - 3s/epoch - 24ms/step
Epoch 2/5
109/109 - 3s - loss: 0.1034 - accuracy: 0.1063 - 3s/epoch - 24ms/step
Epoch 3/5
109/109 - 2s - loss: 0.0370 - accuracy: 0.1470 - 2s/epoch - 22ms/step
Epoch 4/5
109/109 - 2s - loss: 0.0284 - accuracy: 0.1310 - 2s/epoch - 19ms/step
Epoch 5/5
109/109 - 2s - loss: 0.0246 - accuracy: 0.1410 - 2s/epoch - 16ms/step
125/125 [==============================] - 1s 4ms/step
63/63 [==============================] - 1s 4ms/step
MSE of LSTM is 0.022196922053912455
RMSE of LSTM is 0.14898631498870107
R-Squared value of LSTM is 0.784956219754695
```

Figure 8: Output of Simple LSTM model

the simple LSTM model.

```python
# wave_transform.py
import pywt
#import pywavelets
class Wave_Transform:
    #divide the single into two components
    def divide(self, signal):
        component_lowfreq, component_highfreq=pywt.dwt(signal, 'haar')
        return component_lowfreq,component_highfreq

    #inverse dwt to regain original signal
    def combine(self, component_lowfreq, component_highfreq):
        signal=pywt.idwt(component_lowfreq, component_highfreq, 'haar')
        return signal
```

Figure 9: Wavelet Transform Script

```python
proposed_model.py
from datapreprocessing_divide import Divide_Signal
from wave_transform import Wave_Transform
from svrmodel import SVRModel
from lstmmodel import LSTMModel
import numpy as np
from filter import Smoothing

#Class to apply the proposed model that consist of wavelet transformation, SVR and LSTM.
class Proposed_Model:

    #method that return the predicted 10 values for input 90 values for every row.
    def predict(self, datarows):
        #perform wavelet transformation and divide input signal into low and high frequency components
        ds=Divide_Signal(datarows)
        ds.split_signals()

        #create svr model object
        svr= SVRModel(ds.lowfreq_test_x)
        #finding predictions for low frequency components for the test dataset
        lowfreq_predictions=svr.predictions_by_svr()
        #combining test input set with predicted values.
        lowfreq_combined=np.concatenate((ds.lowfreq_test_x, lowfreq_predictions), axis=1)

        #create LSTM model
        lstm=LSTMModel()
        #applying lstm on the high frequency components of test dataset
        highfreq_predicted=lstm.predict(ds.highfreqcomp_X_train,ds.highfreqcomp_Y_train,ds.highfreqcomp_X_test)
        #combining predictions with test input set.
        highfreq_combined=np.concatenate((ds.highfreqcomp_X_test,highfreq_predicted), axis=1)

        #perform inverse wavelet transformation to combine high and low frequency components
        signal_combine=list()
        wt=Wave_Transform()
        for i in range(highfreq_combined.shape[0]):
            wv=wt.combine(lowfreq_predictions[i], highfreq_predicted[i])
            signal_combine.append(wv)
        signal_combine=np.array(signal_combine)
        #return last ten predicted values for every row.
        selected_values=signal_combine[:, -10:]
        smoothing=Smoothing()
        smoothened_output=smoothing.filter_input(selected_values)
        return selected_values
```

Figure 10: LSTM+SVR hybrid model Script

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 5)                 140

 dense_1 (Dense)             (None, 5)                 30

=================================================================
Total params: 170
Trainable params: 170
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
109/109 - 2s - loss: 7.4948e-04 - accuracy: 0.1940 - 2s/epoch - 17ms/step
Epoch 2/5
109/109 - 1s - loss: 7.3861e-04 - accuracy: 0.2170 - 761ms/epoch - 7ms/step
Epoch 3/5
109/109 - 1s - loss: 7.3191e-04 - accuracy: 0.2265 - 753ms/epoch - 7ms/step
Epoch 4/5
109/109 - 1s - loss: 7.2696e-04 - accuracy: 0.2360 - 736ms/epoch - 7ms/step
Epoch 5/5
109/109 - 1s - loss: 7.2252e-04 - accuracy: 0.2422 - 749ms/epoch - 7ms/step
125/125 [==============================] - 1s 2ms/step
63/63 [==============================] - 0s 2ms/step
MSE of Proposed Model is 0.011953328008985367
RMSE of Proposed Model is 0.10933127644450771
R-Squared value of Proposed Model is 0.8840300808389097
PS C:\Users\preet\Implementation>
```

Figure 11: Output of LSTM+SVR model