National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Siranjeevi Muthusamy
Student ID: x22241647

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Siranjeevi Muthusamy
Student ID:	x22241647
Programme:	MSc Cloud Computing **Year:** 2023-2024
Module:	MSc Research Project
Lecturer:	Yasantha Samarawickrama
Submission Due Date:	16-09-2024
Project Title:	Federated Deep Learning for Privacy Preserving Collaborative Data Sharing and Fault Recovery in Connected Vehicles
Word Count:	6851 **Page Count:** 19

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: 

Date: **16th Sep 2024**

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):	

Configuration Manual

Siranjeevi Muthusamy
Student ID: x22241647

1 Introduction

To help the researchers and other academic purposes, this manual provides step by step approach to setup the project. This includes system requirements, environment setup, programming languages, libraries and other dependencies that are essential for the project to run.

2 System Specifications

AWS Studio Notebook Instance:

- vCPUs: 4
- Memory: 16 GiB
- Networking: 25Gbps
- Storage: EBS
- GPUs: 1 NVIDIA T4 Tensor Core GPU
    - GPU Memory: 16 GiB GDDR6
    - CUDA Cores: 2,560
    - Tensor Cores: 320

3 Essential Software Packages and Libraries

Initialize a new studio notebook instance and proceed with the installation of following packages and libraries

1. **Programming Language**
    a. Python
    b. Version: 3.10
2. **TensorFlow Federated (TFF)**
    a. Installation: !pip install --quiet --upgrade tensorflow_federated
    b. Version: >= 2.13 < 2.18
3. **Nest Asyncio**
    a. Installation: !pip install --quiet --upgrade nest_asyncio
4. **TensorBoard Extensions**
    a. Command: %reload_ext tensorboard

3.1 Import Libraries for Data Processing, Models and Visualization

a. Numpy, Pandas, Matplotlib, Seaborn

a. Commands**:**

i. import numpy as np
ii. import pandas as pd
iii. import seaborn as sns
iv. import matplotlib.pyplot as plt

b. Scikit-learn (sklearn)

a. **Import:** from sklearn.metrics import accuracy_score

c. TensorFlow

a. Import: import tensorflow as tf

d. TensorFlow Federated (TFF)

a. Import: import tensorflow_federated as tff

4. Dataset

- **MNIST**
- **Import:** (x_train,y_train),(x_test,y_test) = tf.keras.datasets.mnist.load_data()

5. AWS SageMaker Setup

- Create a new SageMaker domain if not available already
- Create a new SageMaker notebook with the following configuration,
  - Choose the "Python 3 (TensorFlow 2.13 Python 3.10 CPU Optimized)" kernel.
  - Select the "ml.c5.xlarge" instance type.
- Run the notebook instance.

6. Data Preparation

```
import tensorflow as tf

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape data for CNN
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

7. Model Implementation

```
def create_keras_model():
    return tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
```

```
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

8. Federated Learning Setup

```
import tensorflow_federated as tff
# Preprocess data for FL
def preprocess(dataset):
    def batch_format_fn(element):
        return (tf.reshape(element['pixels'], [-1, 28, 28, 1]),
                tf.reshape(element['label'], [-1, 1]))
    return dataset.batch(20).map(batch_format_fn)

# Create TFF types
sample_batch = tf.nest.map_structure(
    lambda x: x.numpy(), next(iter(preprocess(train_data.take(1)))))
input_spec = sample_batch[0].shape, tf.TensorSpec(shape=[None, 1],
dtype=tf.int64)

def model_fn():
    keras_model = create_keras_model()
    return tff.learning.from_keras_model(
        keras_model,
        input_spec=input_spec,
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

# Create federated training process
iterative_process = tff.learning.build_federated_averaging_process(
    model_fn,
    client_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=0.02),
    server_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=1.0))
```

9. Running Experiments

```
NUM_ROUNDS = 5
NUM_CLIENTS = 10

for round_num in range(NUM_ROUNDS):
    state, metrics = iterative_process.next(state, federated_train_data)
    print(f'Round {round_num}')
    print(metrics)
```

10. Recovery Strategies

```
# Recovery strategies
```

```
def federated_recovery(x_test, y_test):
    return global_accuracy

def start_from_scratch(x_train, y_train, x_test, y_test):
    model, history = train_local_model(x_train, y_train, x_test, y_test)
    return history.history['val_accuracy']

def continue_training(model, x_train, y_train, x_test, y_test):
    _, history = train_local_model(x_train, y_train, x_test, y_test)
    return history.history['val_accuracy']

# Simulate recovery
x_recovery, y_recovery = create_scenario_datasets('all')
federated_recovery_acc = [federated_recovery(x_recovery, y_recovery)] * 5
scratch_recovery_acc = start_from_scratch(x_train, y_train, x_recovery,
y_recovery)
continue_recovery_acc = continue_training(local_model_even, x_train, y_train,
x_recovery, y_recovery)
```

References

LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). MNIST handwritten digit database. Retrieved from http://yann.lecun.com/exdb/mnist/

Google Research. (2019). TensorFlow Federated: Machine Learning on Decentralized Data. Retrieved from https://www.tensorflow.org/federated