National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Didheemose Pananchickal Sebastian
Student ID: x23176245

School of Computing
National College of Ireland

Supervisor: Sudarshan Deshmukh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Didheemose Pananchickal Sebastian
Student ID:	x23176245
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Sudarshan Deshmukh
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	738
Page Count:	8

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:	DIDHEEMOSE
Date:	11th December 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

Didheemose Pananchickal Sebastian
x23176245

1 Introduction

This is the deployment configuration document of the self-adaptive federated learning system for financial fraud detection. In the project, heterogeneity in resources, scalability, and privacy in financial fraud detection are targeted by using the FL framework. This tutorial gives a detail of every step in order to configure, deploy, and execute the system on cloud infrastructure.

2 Pre-Requisites

2.1 Software Requirements

- Python: Version 3.10.15

```
!python --version

Python 3.10.15
```

Figure 1: Python Version

- Libraries:
  - kneed
  - pandas
  - seaborn
  - tflite-runtime
  - imbalanced-learn
  - typing-extensions (upgrade)
  - tensorflow-federated (upgrade)
  - numpy
  - flask
  - tensorflow-federated (upgrade)
  - typing-extensions (upgrade)

- jupyter
- tensorflow-model-optimization

```
!pip install kneed
!pip install pandas
!pip install seaborn
!pip install tflite-runtime
!pip install imbalanced-learn
!pip install --upgrade typing-extensions
!pip install --upgrade tensorflow-federated
```

Figure 2: Libraries

- Tools:
  - Jupyter Notebook for testing and development.
  - AWS EC2 Instance

2.2 Hardware Requirements

- Server EC2 Instance type: t3.2xlarge.
- Client EC2 Instances: Use 4 instances with different resources (eg. t3.2xlarge, t2.micro).

2.3 Datasets

- Dataset Source: Kaggle Credit Card Fraud Detection dataset.
- Preprocessing: Handle class imbalance, scale features, and encode categorical variables.

3 Configuration Steps

3.1 Server Configuration

1. Launch an EC2 instance with the required specifications.
2. Connect to the EC2 instance using SSH client through system command line (eg. Command prompt, Windows Powershell).
3. Create a python environment.

```
sudo apt install python3.10-venv
python3.10 -m venv myenv
```

Figure 3: To Create Python Environment

4. Activate the environment.

```
source myenv/bin/activate
```

Figure 4: To Activate Python Environment

5. Run the following commands to install Python and required libraries.

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.10

pip install numpy
pip install flask
pip install --upgrade tensorflow-federated
pip install --upgrade typing-extensions
pip install jupyter
pip install tensorflow-model-optimization
```

Figure 5: Python and Libraries

6. Set up the Flask server to communicate with clients for model and weights transfering or upload the server configuration file (global.py file).

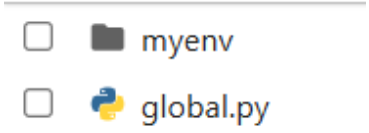

Figure 6: Server File Structure

3.2 Client Configuration

1. Launch 4 EC2 instances with the minium required Specifications.
2. Connect to the EC2 instance using SSH through system command line (eg. Command prompt, Windows Powershell).
3. Create a python environment.
4. Activate the environment.
5. Run the following commands to install Python and required libraries.
6. Place the dataset (.csv), jupyter file (.ipynb) and run_benchmark.py specific to each client on their respective instance.

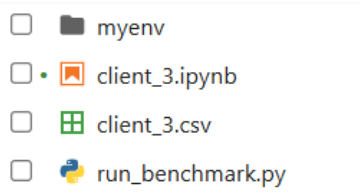

Figure 7: Client File Structure

4 Execution Steps

4.1 Server Execution

1. Start the Flask server either from command line or through jupyter notebook (jupyter notebook prefered) to handle client requests, pruning, quantisation and weights aggregation.

```
python global.py
```

Figure 8: To Start Flask Server

2. Ensure the server is listening on the correct port (eg. 5000).

```
* Serving Flask app 'global'
* Debug mode: off


* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.36.251:5000
Press CTRL+C to quit
```

Figure 9: Server listening on port 5000

3. The server aggregates client weights using Federated Averaging (FedAvg).
4. Update the global model and send it back to the clients for further training.

4.2 Client Execution

1. Start the jupyter notebook (port number should be different for each client)

```
ubuntu@ip-172-31-18-82: ~
(myenv) ubuntu@ip-172-31-18-82:~$ jupyter notebook --no-browser --port=8883
```

Figure 10: Command to Start Jupyter Notebook

2. Access the jupyter notebook on browser (eg. localhost:8883/tree).
3. Execute the (.ipynb) script on each client to trains the local model and save weights in (.h5) format.
4. Upload weights to the server using the /upload_weights endpoint.

```
try:
    # Send weights file via a POST request to the server
    url = 'http://54.75.107.93:5000/upload_weights'
    files = {'file': open('model_weights.weights.h5', 'rb')}
    data = {'sample_size': len(X_train)}
    response = requests.post(url, files=files, data=data)
    print(response.text)  # Check if the upload was successful
except Exception as e:
    print(f"Error uploading weights: {str(e)}")
```

```
Weights uploaded
```

Figure 11: Weights Uploading

5. Ensure the script includes necessary changes for unique client IDs and datasets (eg. client_1.ipynb, client_1.csv)

4.3 Benchmarking

1. Run the run_benchmark.py script on each client to determine computational and network capabilities

```
python run_benchmark.py
```

Figure 12: Benchmarking

2. Upload benchmark results to the server for classification and appropriate model adjustments.

```
Benchmark Results:
CPU Time: 0.03313279151916504 seconds
Total Memory: 33263824896 bytes
Available Memory: 31828672512 bytes
Download Time: 0.09282326698303223 seconds
Data Size: 25888 bytes
Client classified as high-capacity.
```

Figure 13: Example of Benchmark Results

5 Adaptive Techniques

5.1 Pruning and Quantization

1. Enable pruning to reduce model complexity by eliminating less important weights for low-resource clients based on client benchmark results..

2. Apply quantization to optimize resource utilization for medium resource clients based on client benchmark results.

5.2 Client Classification

Use benchmark results (e.g., cpu_time, memory) from the run_benchmark.py script to classify clients.

- High Capacity: Full model.
- Medium Capacity: Pruned model (30%).
- Low Capacity: Pruned and quantized model.

5.3 Communication

Clients fetch the updated model from the server using the /get_model endpoint, which provides the best-suited model based on their classification.

```
try:
    url = 'http://54.75.107.93:5000/get_model'
    response = requests.post(url, json=benchmark_results)
    if response.status_code == 200:
        # Save the model file locally
        with open('server_model.h5', 'wb') as f:
            f.write(response.content)
        print("Model downloaded and saved as 'server_model.h5'")
    else:
        print(f"Failed to download the model. Status code: {response.status_code}")
except Exception as e:
    print(f"Error downloading or loading the model: {str(e)}")
Model downloaded and saved as 'server_model.h5'
```

Figure 14: Example of /get_model endpoint

6 System Architecture

- Server - Client Interaction:
  - Clients runs a benchmark test and request for updated model from server by passing benchmark results.
  - The server check the client resource to check whether the Pruning and Quantization required or not before sending the model.
  - Clients loads the model received server and train locally and share weights with the server through HTTP request.
  - The server aggregates and redistributes the global model for next training round.

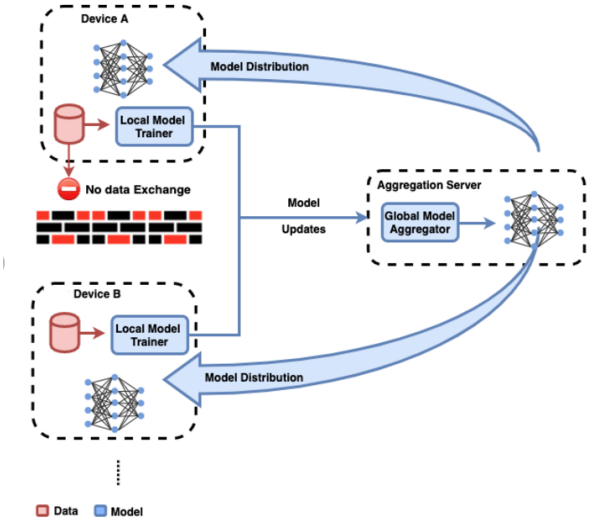

Figure 15: Federated Learning System Architecture

6.1 Results Verification

1. Evaluate model performance using:
   - Accuracy, Precision, Recall, and F1 Score.
   - Confusion Matrix: Compare True Positives, False Positives, etc.
2. Resource usage: Monitor CPU, memory, and network bandwidth.

6.2 Troubleshooting

Table 1: Troubleshooting

Issue	**Solution**
Benchmark results not transmitted correctly	Verify the network connection and server URL.
Model not updated on the server	Ensure all clients upload weights successfully.
High resource usage on clients	Use pruning and quantization.
Flask server fails to start	Confirm Python and Flask installation.

7 Additional Notes

1. Endpoints:
   - /upload_weights: To upload client weights.
   - /get_model: To fetch updated model based on benchmarks.
2. System Logs:
   - Monitor the server logs for weight aggregation status.
   - Ensure no missing client uploads before aggregation.