Configuration Manual

MSc Research Project
Artificial Intelligence

Ramanathan Arunachalam
Student ID: x22142401

School of Computing
National College of Ireland

Supervisor: Prof. Abdul Razzaq

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ramanathan Arunachalam
Student ID:	x22142401
Programme:	Artificial Intelligence
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Abdul Razzaq
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	453
Page Count:	9

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:	31st January 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

Ramanathan Arunachalam
x22142401

1 Introduction

The configuration handbook details how to carry out the research topic "Enhancing Fake News Detection with Federated Learning and Word Embedding" step by step. The upcoming sections will explain the details about software and hardware requirements for implementation of this project. By following the steps in order to replicate the outputs that are shown. Machine Learning algorithms such as LSTM, CNN, BERT, ect are discussed in this manual

2 System Requirements

This part outlines the system requirements for successfully performing the project, and it is always necessary to have prior knowledge of the system specification before conducting tests

2.1 System Specification

- **Platform**: Google Colaboratory
- **Runtime**: GPU/TPU
- **RAM**: 12.7GB
- **Disc**: 107.7 GB

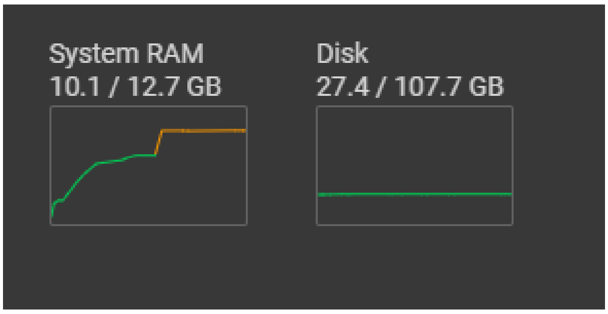


Figure 1: Colab Runtime

3 Code

3.1 Dataset

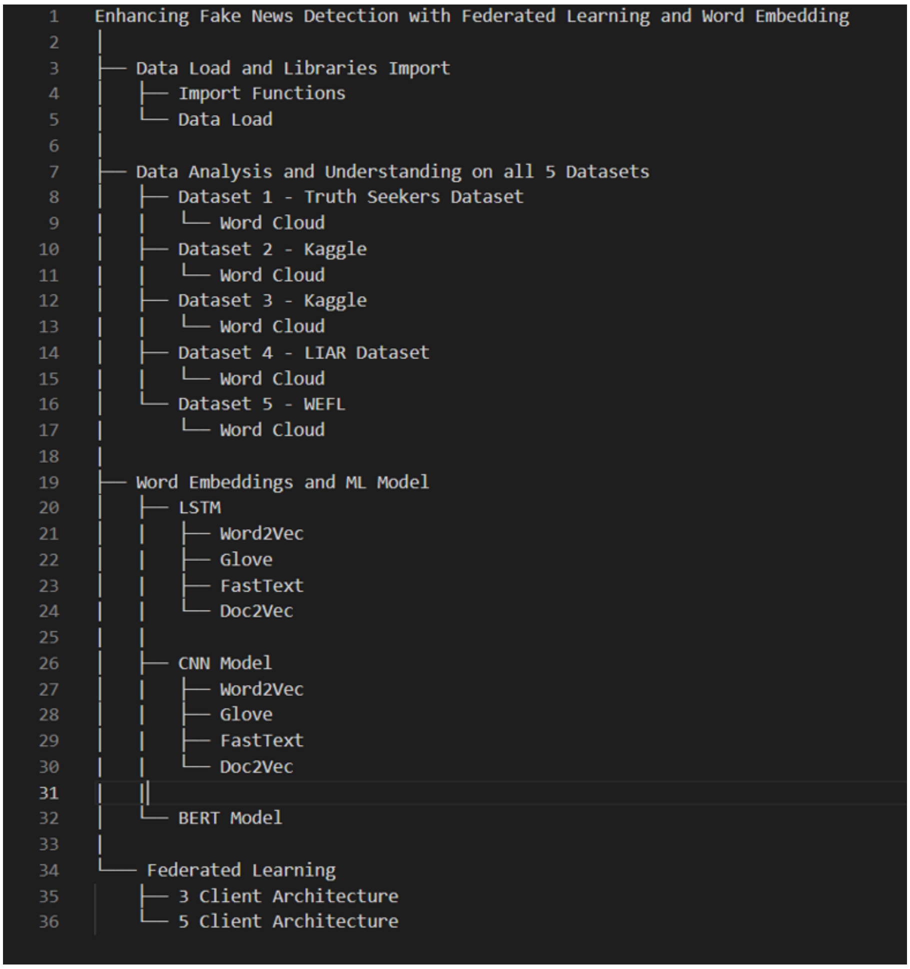
This program uses five data source of different dimensions. The below table show the overview of the data used.

Dataset Name	Rows	Columns
Truth_Seeker (2023)	134199	8
Kaggle Fake News Data 1	44920	4
LIAR	10239	14
WEFL	72134	4
Kaggle Fake News Data 2	6335	4

Table 1: Datasets Shape

3.2 Data Loading

The below image show a brief view of how the code is structured.



```
Enhancing Fake News Detection with Federated Learning and Word Embedding
│
├── Data Load and Libraries Import
│   ├── Import Functions
│   └── Data Load
│
├── Data Analysis and Understanding on all 5 Datasets
│   ├── Dataset 1 - Truth Seekers Dataset
│   │   └── Word Cloud
│   ├── Dataset 2 - Kaggle
│   │   └── Word Cloud
│   ├── Dataset 3 - Kaggle
│   │   └── Word Cloud
│   ├── Dataset 4 - LIAR Dataset
│   │   └── Word Cloud
│   └── Dataset 5 - WEFL
│       └── Word Cloud
│
├── Word Embeddings and ML Model
│   ├── LSTM
│   │   ├── Word2Vec
│   │   ├── Glove
│   │   ├── FastText
│   │   └── Doc2Vec
│   │
│   ├── CNN Model
│   │   ├── Word2Vec
│   │   ├── Glove
│   │   ├── FastText
│   │   └── Doc2Vec
│   │
│   └── BERT Model
│
└── Federated Learning
    ├── 3 Client Architecture
    └── 5 Client Architecture
```

Figure 2: Code Workflow

1. Access the data from your specified location by mounting the Google Drive in Google Colab. Unzip the dataset saved on the drive to the chosen directory.

2. Import the required libraries, if throws an error please pip install few dependencies and restart the session.

```python
# Import essential libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report

# Import data preprocessing libraries
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud

# Import text processing libraries
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Import language models and word embedding libraries
import nltk
nltk.download('stopwords')
nltk.download('punkt')
import gensim
import gensim.utils
from gensim.models import Word2Vec, FastText, Doc2Vec, KeyedVectors

# Import text preprocessing and modeling libraries
from gensim.parsing.preprocessing import preprocess_string
from gensim.models.doc2vec import TaggedDocument, Doc2Vec
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense, Dropout

# Import deep learning libraries
import tensorflow as tf
from tensorflow.keras.layers import LSTM
import gensim.downloader as api
```

Figure 3: Import Statement

3. Load all the 5 data as dataframes.

```python
dataset1 = "/content/drive/MyDrive/Projects/MSc Thesis/TruthSeeker2023/TruthSeeker2023/Truth_Seeker_Model_Dataset.csv" #new
dataset2t = "/content/drive/MyDrive/Projects/MSc Thesis/DatasetFakeNews KaggleNew/True.csv" #kagle
dataset2f = "/content/drive/MyDrive/Projects/MSc Thesis/DatasetFakeNews KaggleNew/Fake.csv" #kagle
dataset3 = "/content/drive/MyDrive/Projects/MSc Thesis/Kaggle.csv" #k
dataset4 = "/content/drive/MyDrive/Projects/MSc Thesis/liar_dataset/train.tsv" #liar
dataset5 = "/content/drive/MyDrive/Projects/MSc Thesis/WELFake_Dataset.csv" #WEFL
```

Figure 4: Data Load

4. Data Cleaning is important because we are handling text data and need for clutter free word corpus is important for analysis.

```python
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

def preprocess_text(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)

    tokens = word_tokenize(text)  # Tokenize the text
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]  # Remove stop words

    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]  # Stemming

    return tokens

```

Figure 5: Data Clean Function

3.3 Modeling

The code first does LSTM modeling with for embeddings and then do CNN model with same four embedding on 5 different datasets.

```python
def word2vec_lstm_accuracy_loss(dataset_X, dataset_Y):

    word2vec_model = gensim.models.Word2Vec(
        dataset_X, vector_size=100, window=5, min_count=1, workers=4
    )
    word2vec_model.save("word2vec_model.bin")

    # Create sequence of word indices
    word_index = {w: word2vec_model.wv.key_to_index[w] for w in word2vec_model.wv.index_to_key}
    X = [[word_index[word] for word in sent if word in word_index] for sent in dataset_X]

    # Pad sequences
    max_len = max([len(x) for x in X])
    X_1 = pad_sequences(X, maxlen=max_len, padding='post')

    # Convert labels to numpy array
    y = np.array(dataset_Y)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_1, y, test_size=0.2, random_state=42)

    model = Sequential()
    model.add(Embedding(len(word_index) + 1, 100, input_length=max_len))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2, kernel_regularizer=l2(0.01)))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Train the model
    model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test), verbose=1)
    # Evaluate the model

    loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
    print(f'Accuracy: {accuracy*100:.2f}%')
    print(f'Loss: {loss}')

```

Figure 6: LSTM

This is how the function call is made for each function

```
glove_lstm(df1['preprocessed_statement'], df1['target'])
```

```
glove_lstm(df2['preprocessed_news'], df2['label'])
```

```
glove_lstm(df3["preprocessed_Text"], df3['new_label'])
```

```
glove_lstm(df4['preprocessed_statement'], df4['new_label'])
```

```
glove_lstm(df5['preprocessed_title_text'], df5['label'])
```

Figure 7: Word2Vec Function Call for different datasets

These are the results are as follows

```
word2vec_lstm_accuracy_loss(df1['preprocessed_statement'], df1['target'])

Epoch 1/5
3355/3355 [==============================] - 298s 87ms/step - loss: 0.0527 - accuracy: 0.9899 - val_loss: 0.0030 - val_accuracy: 0.9998
Epoch 2/5
3355/3355 [==============================] - 277s 82ms/step - loss: 0.0067 - accuracy: 0.9995 - val_loss: 0.0059 - val_accuracy: 0.9998
Epoch 3/5
3355/3355 [==============================] - 285s 85ms/step - loss: 0.0080 - accuracy: 0.9996 - val_loss: 0.0244 - val_accuracy: 0.9984
Epoch 4/5
3355/3355 [==============================] - 282s 84ms/step - loss: 0.0040 - accuracy: 0.9999 - val_loss: 0.0025 - val_accuracy: 0.9998
Epoch 5/5
3355/3355 [==============================] - 299s 89ms/step - loss: 0.0055 - accuracy: 0.9999 - val_loss: 0.0025 - val_accuracy: 0.9999
839/839 [==============================] - 9s 11ms/step - loss: 0.0025 - accuracy: 0.9999
Accuracy: 99.99%
Loss: 0.002490422921255231
```

Figure 8: LSTM-Word2Vec

```
glove_lstm(df1['preprocessed_statement'], df1['target'])

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/5
3355/3355 [==============================] - 595s 174ms/step - loss: 0.2330 - accuracy: 0.9098 - val_loss: 0.0714 - val_accuracy: 0.9760
Epoch 2/5
3355/3355 [==============================] - 581s 173ms/step - loss: 0.1325 - accuracy: 0.9596 - val_loss: 0.0413 - val_accuracy: 0.9928
Epoch 3/5
3355/3355 [==============================] - 580s 173ms/step - loss: 0.1159 - accuracy: 0.9649 - val_loss: 0.0462 - val_accuracy: 0.9889
Epoch 4/5
3355/3355 [==============================] - 580s 173ms/step - loss: 0.1096 - accuracy: 0.9669 - val_loss: 0.0350 - val_accuracy: 0.9936
Epoch 5/5
3355/3355 [==============================] - 583s 174ms/step - loss: 0.1059 - accuracy: 0.9684 - val_loss: 0.0303 - val_accuracy: 0.9947
839/839 [==============================] - 10s 12ms/step - loss: 0.0303 - accuracy: 0.9947
Accuracy: 99.47%
```

Figure 9: LSTM-Glove

```
[64] fasttext_lstm_model(df4['preprocessed_statement'], df4['new_label'])

Found 8711 unique tokens.
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/5
256/256 [==============================] - 1042s 4s/step - loss: 0.6717 - accuracy: 0.5717 - val_loss: 0.6683 - val_accuracy: 0.5654
Epoch 2/5
256/256 [==============================] - 1025s 4s/step - loss: 0.6671 - accuracy: 0.5826 - val_loss: 0.6801 - val_accuracy: 0.5679
Epoch 3/5
256/256 [==============================] - 1013s 4s/step - loss: 0.6650 - accuracy: 0.5854 - val_loss: 0.6670 - val_accuracy: 0.5659
Epoch 4/5
256/256 [==============================] - 1018s 4s/step - loss: 0.6656 - accuracy: 0.5817 - val_loss: 0.6713 - val_accuracy: 0.5669
Epoch 5/5
256/256 [==============================] - 1014s 4s/step - loss: 0.6638 - accuracy: 0.5878 - val_loss: 0.6664 - val_accuracy: 0.5752
64/64 [==============================] - 18s 278ms/step - loss: 0.6664 - accuracy: 0.5752
Validation Accuracy: 0.5752
<keras.src.engine.sequential.Sequential at 0x7e381a26ae30>
```

Figure 10: LSTM-FastText

Similarly as next step we are calling an CNN function.

```python
def word2vec_cnn_accuracy_loss(dataset_X, dataset_Y):

    word2vec_model = gensim.models.Word2Vec(
        dataset_X, vector_size=100, window=5, min_count=1, workers=4
    )

    # Create sequence of word indices
    word_index = {w: word2vec_model.wv.key_to_index[w] for w in word2vec_model.wv.index_to_key}
    X = [[word_index[word] for word in sent if word in word_index] for sent in dataset_X]

    word_index = {w: word2vec_model.wv.key_to_index[w] for w in word2vec_model.wv.index_to_key}
    print('Found %s word vectors.' %len(word_index))

    # Convert labels to numpy array
    y = dataset_Y

    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(dataset_X)

    word_index = tokenizer.word_index
    vocabulary_size = len(tokenizer.word_index) + 1
    print("Vocabulary Size :", vocabulary_size)

    embedding_matrix = np.zeros((vocabulary_size, EMBEDDING_DIM))
    for word, i in word_index.items():
      embedding_vector = embeddings_index.get(word)
      if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

    BATCH_SIZE = 32
    EPOCHS = 5
    MAX_SEQUENCE_LENGTH = 30
```

Figure 11: CNN Function

The below are the outputs of CNN

```
model_A.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0

 embedding_1 (Embedding)     (None, 30, 100)           281200

 dropout (Dropout)           (None, 30, 100)           0

 conv1d (Conv1D)             (None, 24, 128)           89728

 conv1d_1 (Conv1D)           (None, 18, 128)           114816

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense_1 (Dense)             (None, 512)               66048

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 1)                 513

=================================================================
Total params: 552305 (2.11 MB)
Trainable params: 271105 (1.03 MB)
Non-trainable params: 281200 (1.07 MB)
_________________________________________________________________
```

Figure 12: Parameters in CNN

```
%%time
history_A = model_A.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, validation_data=(X_test, y_test), callbacks=[es, reduce_lr])

Epoch 1/5
1123/1123 [==============================] - 8s 6ms/step - loss: 0.2683 - accuracy: 0.8850 - val_loss: 0.1511 - val_accuracy: 0.9414 - lr: 0.0010
Epoch 2/5
1123/1123 [==============================] - 7s 6ms/step - loss: 0.1600 - accuracy: 0.9372 - val_loss: 0.1306 - val_accuracy: 0.9521 - lr: 0.0010
Epoch 3/5
1123/1123 [==============================] - 6s 5ms/step - loss: 0.1399 - accuracy: 0.9445 - val_loss: 0.1156 - val_accuracy: 0.9566 - lr: 0.0010
Epoch 4/5
1123/1123 [==============================] - 7s 6ms/step - loss: 0.1234 - accuracy: 0.9514 - val_loss: 0.1017 - val_accuracy: 0.9624 - lr: 0.0010
Epoch 5/5
1123/1123 [==============================] - 6s 6ms/step - loss: 0.1143 - accuracy: 0.9558 - val_loss: 0.0962 - val_accuracy: 0.9647 - lr: 0.0010
CPU times: user 36.6 s, sys: 2.51 s, total: 39.1 s
Wall time: 34.5 s
```

Figure 13: CNN Training

Then we do BERT Modeling

```python
# Split dataset into train and test
train_texts, test_texts, train_labels, test_labels = train_test_split(
    df1['preprocessed_statement'].values.tolist(),
    df1['target'].values.tolist(),
    test_size=0.2,
    random_state=42
)
```

```python
# Initialize BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```python
# Tokenize text and convert labels to tensors
train_encodings = tokenizer(train_texts, truncation=True, padding=True)
test_encodings = tokenizer(test_texts, truncation=True, padding=True)

train_labels = torch.tensor(train_labels)
test_labels = torch.tensor(test_labels)
```

Figure 14: BERT Model

After all these code we choose the best model among these and do the Federated Learning.

```python
# Create a global model
global_model = SimpleModel()
```

```python
client1_data = doc2vec_ds1
client2_data = doc2vec_ds2
client3_data = glove_ds1

```

```python
# Federated learning iterations
losses = []  # For storing losses across rounds
for round_num in range(3):  # Assuming 3 rounds of federated learning
    print(f"Round {round_num + 1}:")
    # Client updates
    for client_data in [client1_data, client2_data, client3_data]:
        local_model = SimpleModel()
        local_model.load_state_dict(global_model.state_dict())  # Initialize with the global model
        optimizer = optim.SGD(local_model.parameters(), lr=0.01)
        loss = client_train(local_model, optimizer, client_data)
        losses.extend(loss)

        # Update global model using a weighted average
        global_state_dict = global_model.state_dict()
        local_state_dict = local_model.state_dict()
        for key in global_state_dict.keys():
            global_state_dict[key] += local_state_dict[key] / 3  # Assuming equal weight for each client

        global_model.load_state_dict(global_state_dict)
```

Figure 15: Federated Learning

Similarly a 5 clinet architecture is done

```python
client1_data = doc2vec_ds1
client2_data = doc2vec_ds2
client3_data = glove_ds1
client4_data = glove_ds3
client5_data = fasttext_df2

#dtype=torch.float

```

```python
# Create a global model
global_model = SimpleModel()
```

```python
# Federated learning iterations
losses = []  # For storing losses across rounds
for round_num in range(3):  # Assuming 3 rounds of federated learning
    print(f"Round {round_num + 1}:")
    # Client updates
    for client_data in [client1_data, client2_data, client3_data]:
        local_model = SimpleModel()
        local_model.load_state_dict(global_model.state_dict())  # Initialize with the global model
        optimizer = optim.SGD(local_model.parameters(), lr=0.001)  # Lower learning rate
        loss = client_train(local_model, optimizer, client_data)
        losses.extend(loss)

        # Update global model using a weighted average
        global_state_dict = global_model.state_dict()
        local_state_dict = local_model.state_dict()
        for key in global_state_dict.keys():
            global_state_dict[key] += local_state_dict[key] / 3  # Assuming equal weight for each client

        global_model.load_state_dict(global_state_dict)
```

Figure 16: Federated Learning 5 Client Architecture

The Federated Learning gave this two loss function

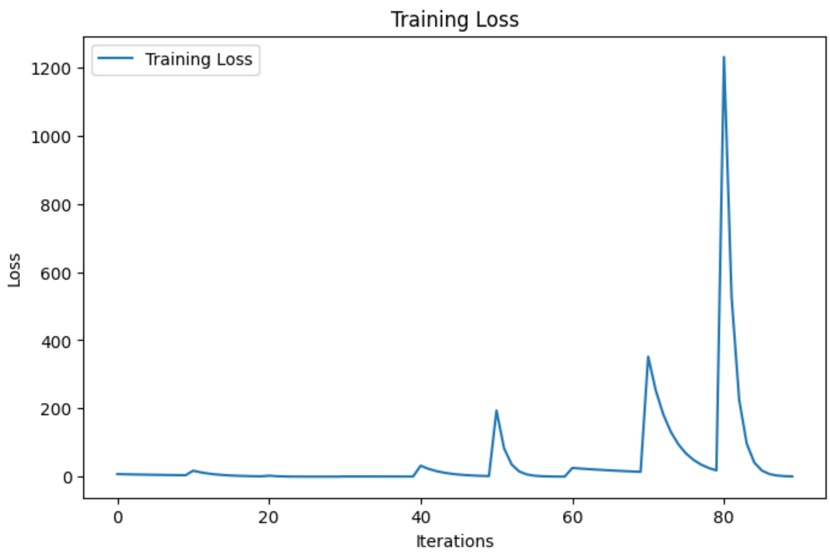


Figure 17: Federated Learning 3 Client Architecture

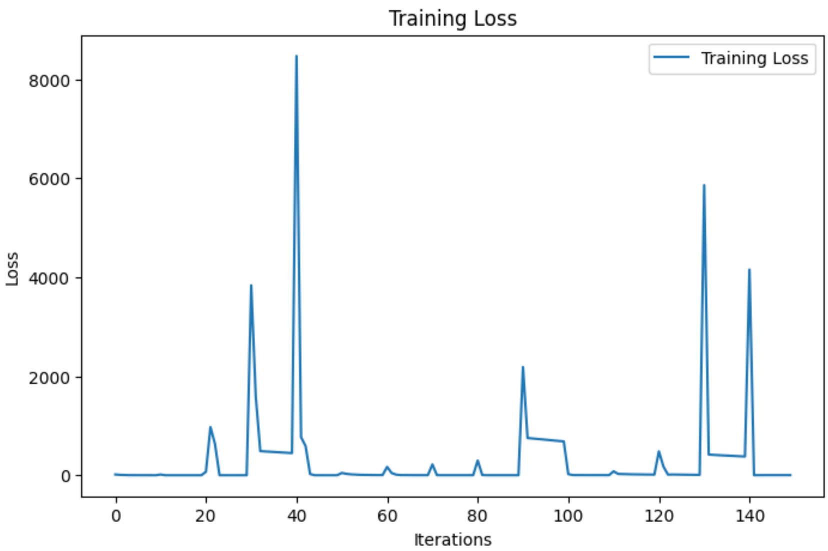


Figure 18: Federated Learning 5 Client Architecture

These are the steps followed in executing this code.