

# 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



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

<b>Signature:</b>	
<b>Date:</b>	31st January 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
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

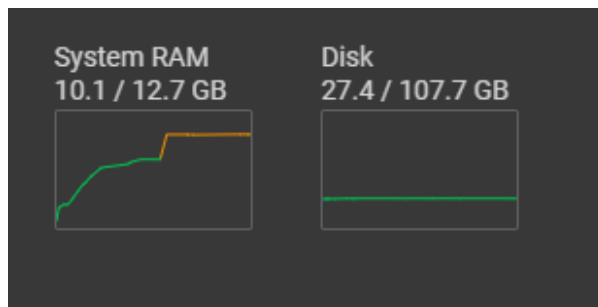


Figure 1: Colab Runtime

## 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.

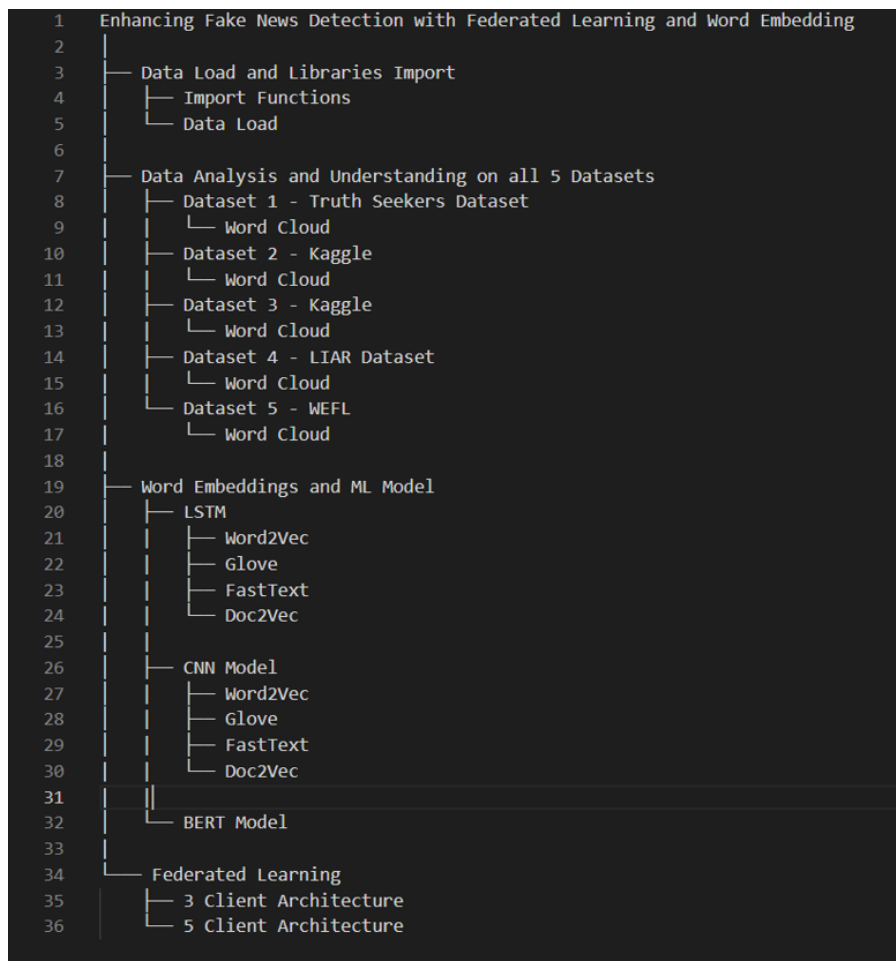


Figure 2: Code Workflow

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.

```
1 # Import essential libraries
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.metrics import classification_report
6
7 # Import data preprocessing libraries
8 import numpy as np
9 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
10 from sklearn.model_selection import train_test_split
11 from wordcloud import WordCloud
12
13 # Import text processing libraries
14 import re
15 from nltk.corpus import stopwords
16 from nltk.stem.porter import PorterStemmer
17
18 # Import language models and word embedding libraries
19 import nltk
20 nltk.download('stopwords')
21 nltk.download('punkt')
22 import gensim
23 import gensim.utils
24 from gensim.models import Word2Vec, FastText, Doc2Vec, KeyedVectors
25
26 # Import text preprocessing and modeling libraries
27 from gensim.parsing.preprocessing import preprocess_string
28 from gensim.models.doc2vec import TaggedDocument, Doc2Vec
29 from tensorflow.keras.preprocessing.sequence import pad_sequences
30 from tensorflow.keras.models import Sequential
31 from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense, Dropout
32
33 # Import deep learning libraries
34 import tensorflow as tf
35 from tensorflow.keras.layers import LSTM
36 import gensim.downloader as api
```

Figure 3: Import Statement

3. Load all the 5 data as dataframes.

```
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/lian_dataset/train.tsv" #lian
dataset5 = "/content/drive/MyDrive/Projects/MSc Thesis/WELFake_Dataset.csv" #WEL
```

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.

```

1 import re
2 from nltk.tokenize import word_tokenize
3 from nltk.corpus import stopwords
4 from nltk.stem import PorterStemmer
5
6 def preprocess_text(text):
7     text = str(text).lower()
8     text = re.sub('\[.?\]\|', '', text)
9     text = re.sub('https?://www.\S+', '', text)
10    text = re.sub('<.*>+', '', text)
11    text = re.sub('\n', '', text)
12    text = re.sub('\w*\d\w*', '', text)
13
14    tokens = word_tokenize(text) # Tokenize the text
15    stop_words = set(stopwords.words('english'))
16    tokens = [word for word in tokens if word not in stop_words] # Remove stop words
17
18    stemmer = PorterStemmer()
19    tokens = [stemmer.stem(word) for word in tokens] # Stemming
20
21    return tokens
22

```

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.

```

1 def word2vec_lstm_accuracy_loss(dataset_X, dataset_Y):
2
3     word2vec_model = gensim.models.Word2Vec(
4         dataset_X, vector_size=100, window=5, min_count=1, workers=4
5     )
6     word2vec_model.save("word2vec_model.bin")
7
8     # Create sequence of word indices
9     word_index = {w: word2vec_model.wv.key_to_index[w] for w in word2vec_model.wv.index_to_key}
10    X = [[word_index[word] for word in sent if word in word_index] for sent in dataset_X]
11
12    # Pad sequences
13    max_len = max([len(x) for x in X])
14    X_1 = pad_sequences(X, maxlen=max_len, padding='post')
15
16    # Convert labels to numpy array
17    y = np.array(dataset_Y)
18
19    # Split data into training and testing sets
20    X_train, X_test, y_train, y_test = train_test_split(X_1, y, test_size=0.2, random_state=42)
21
22    model = Sequential()
23    model.add(Embedding(len(word_index) + 1, 100, input_length=max_len))
24    model.add(SpatialDropout1D(0.2))
25    model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2, kernel_regularizer=l2(0.01)))
26    model.add(Dense(1, activation='sigmoid'))
27
28    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
29
30    # Train the model
31    model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test), verbose=1)
32    # Evaluate the model
33
34    loss, accuracy = model.evaluate(X_test, y_test, verbose=1)
35    print(f'Accuracy: {accuracy*100:.2f}%')
36    print(f'Loss: {loss}')
37

```

Figure 6: LSTM

This is how the function call is made for each function

```

1 glove_lstm(df1['preprocessed_statement'], df1['target'])
1 glove_lstm(df2['preprocessed_news'], df2['label'])
1 glove_lstm(df3["preprocessed_Text"], df3['new_label'])
1 glove_lstm(df4['preprocessed_statement'], df4['new_label'])
1 glove_lstm(df5['preprocessed_title_text'], df5['label'])

```

Figure 7: Word2Vec Function Call for different datasets

These are the results are as follows

```

1 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: 0.002490422021255231

```

Figure 8: LSTM-Word2Vec

```

1 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 270ms/step - loss: 0.6664 - accuracy: 0.5752
Validation Accuracy: 0.5752
<keras.src.engine.sequential.Sequential at 0x7c381a2cae30>

```

Figure 10: LSTM-FastText

Similarly as next step we are calling an CNN function.

```

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 (GlobalMaxPooling1D)	(None, 128)	0
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 [=====] - 8s 6ms/step - loss: 0.2683 - accuracy: 0.8850 - val_loss: 0.1511 - val_accuracy: 0.9414 - lr: 0.0010
Epoch 2/5 [=====] - 7s 6ms/step - loss: 0.1600 - accuracy: 0.9372 - val_loss: 0.1306 - val_accuracy: 0.9521 - lr: 0.0010
Epoch 3/5 [=====] - 6s 5ms/step - loss: 0.1399 - accuracy: 0.9445 - val_loss: 0.1156 - val_accuracy: 0.9566 - lr: 0.0010
Epoch 4/5 [=====] - 7s 6ms/step - loss: 0.1234 - accuracy: 0.9514 - val_loss: 0.1017 - val_accuracy: 0.9624 - lr: 0.0010
Epoch 5/5 [=====] - 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

```

1 # Split dataset into train and test
2 train_texts, test_texts, train_labels, test_labels = train_test_split(
3     df['preprocessed_statement'].values.tolist(),
4     df['target'].values.tolist(),
5     test_size=0.2,
6     random_state=42
7 )

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

1 # Tokenize text and convert labels to tensors
2 train_encodings = tokenizer(train_texts, truncation=True, padding=True)
3 test_encodings = tokenizer(test_texts, truncation=True, padding=True)
4
5 train_labels = torch.tensor(train_labels)
6 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.

```

1 # Create a global model
2 global_model = SimpleModel()

1 client1_data = doc2vec_ds1
2 client2_data = doc2vec_ds2
3 client3_data = glove_ds1
4

1 # Federated learning iterations
2 losses = [] # For storing losses across rounds
3 for round_num in range(3): # Assuming 3 rounds of federated learning
4     print(f"Round {round_num + 1}:")
5     # Client updates
6     for client_data in [client1_data, client2_data, client3_data]:
7         local_model = SimpleModel()
8         local_model.load_state_dict(global_model.state_dict()) # Initialize with the global model
9         optimizer = optim.SGD(local_model.parameters(), lr=0.01)
10        loss = client_train(local_model, optimizer, client_data)
11        losses.extend(loss)
12
13        # Update global model using a weighted average
14        global_state_dict = global_model.state_dict()
15        local_state_dict = local_model.state_dict()
16        for key in global_state_dict.keys():
17            global_state_dict[key] += local_state_dict[key] / 3 # Assuming equal weight for each client
18
19        global_model.load_state_dict(global_state_dict)

```

Figure 15: Federated Learning

Similarly a 5 client architecture is done

```

1 client1_data = doc2vec_ds1
2 client2_data = doc2vec_ds2
3 client3_data = glove_ds1
4 client4_data = glove_ds3
5 client5_data = fasttext_df2
6
7 #dtype=torch.float
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 16: Federated Learning 5 Client Architecture

The Federated Learning gave this two loss function

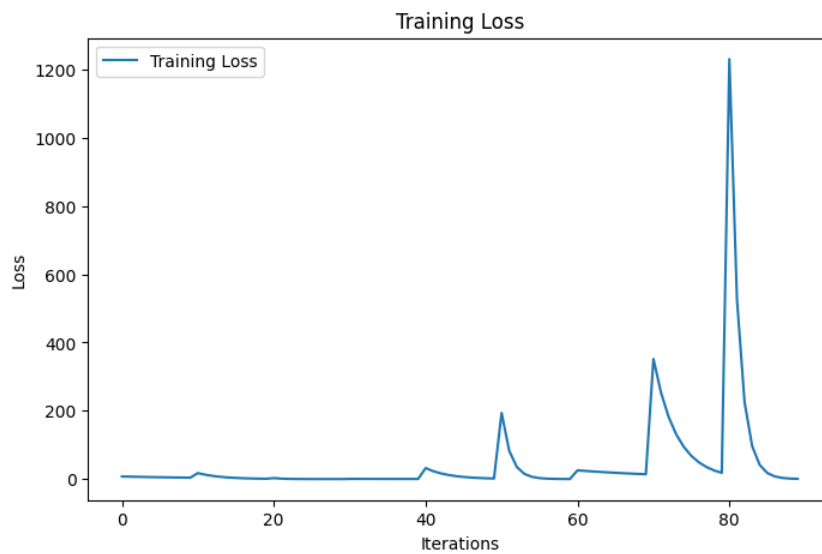


Figure 17: Federated Learning 3 Client Architecture

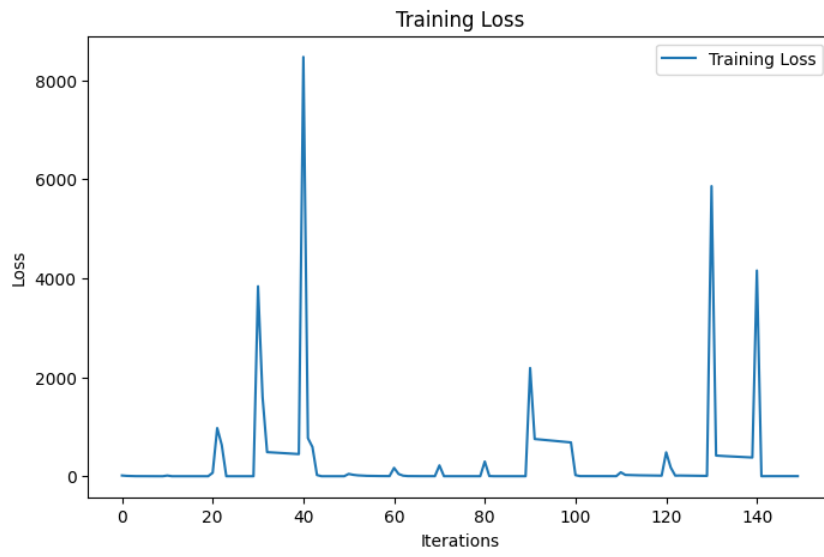


Figure 18: Federated Learning 5 Client Architecture

These are the steps followed in executing this code.