

# Configuration Manual

MSc Research Project  
Data Analytics

Jackay Lohano  
Student ID: 23212896

School of Computing  
National College of Ireland

Supervisor:     Hamilton V. Niculescu

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Jackay Lohano  
**Student ID:** x23212896  
**Programme:** Data Analytics **Year:** 2024  
**Module:** MSc Research Project  
**Lecturer:** Hamilton V. Niculescu  
**Submission Due Date:** 12-12-2024  
**Project Title:** Automated Detection of Fake News in Urdu Language Using Pre-Trained Transformer Models  
**Word Count:** 1661 **Page Count:** 10

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:** Jackay Lohano  
**Date:** 12-12-2024

**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

Jackay Lohano  
x23212896

## 1 Introduction

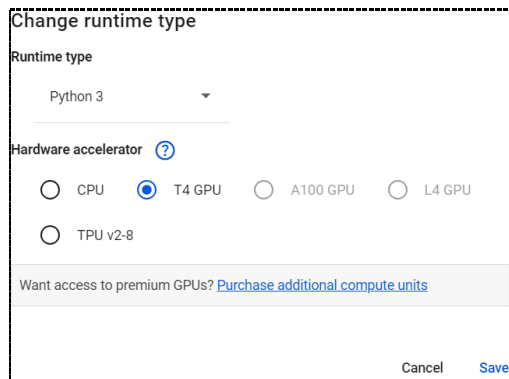
This configuration manual serves as step-by-step guide to replicate the results of the research. Section 2 shows the installation and configuration process of tools, packages and frameworks used in this study. Section 3 illustrates the application those tools and frameworks in order to reproduce results obtained in this research.

## 2 Installation and Configuration

This research uses Google Collaboratory as integrated development environment which provides Jupyter notebook service with free access to computing resources such as GPU and TPU. Although, there are some paid versions i.e. Colab Pro+ which provide more compute units, this research uses free resource of T4 GPU which is enough for this research experiment. According to Nvidia T4 delivers up to 40X higher performance than CPUs<sup>1</sup>. Table 1 lists specification of T4 GPU.

T4 GPU	Specification
Turning Tensor Cores	320 Cores
Nvidia Cuda Cores	2,560 Cores
Memory	16 GB GDDR6

**Table 1: T4 GPU specification**



**Figure 1: T4 GPU runtime**

---

<sup>1</sup> <https://www.nvidia.com/en-us/data-center/tesla-t4/>

Make sure that T4 GPU is selected as runtime in your Google Colab notebook file as illustrated in Figure 1. Python 3 is used as coding language in this study, as can be seen in Figure 1. All the experiments are programmed in Python language.

To access the dataset files in Google Colab Notebook, first Google drive must be mounted as shown in Figure 2. This command links the Colab with Google drive to use the data files.

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

**Figure 2: Mounting google drive**

## 2.1 Tools for data processing

To process Urdu text, this research uses LughaatNLP library. Figure 3 shows the command to install this library. This command installs all the necessary packages required to use this library. This library provides various functions to pre-process Urdu text such as normalization, lemmatization and stop words removal.

```
[ ] !pip install LughaatNLP
```

```
Collecting LughaatNLP
  Downloading LughaatNLP-1.0.6-py3-none-any.whl.metadata (39 kB)
Collecting python-Levenshtein (from LughaatNLP)
  Downloading python-Levenshtein-0.26.1-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (from LughaatNLP) (2.17.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from LughaatNLP) (1.26.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from LughaatNLP) (1.5.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from LughaatNLP) (1.13.1)
Collecting gtts (from LughaatNLP)
  Downloading gtts-2.5.4-py3-none-any.whl.metadata (4.1 kB)
Collecting SpeechRecognition (from LughaatNLP)
  Downloading SpeechRecognition-3.11.0-py2.py3-none-any.whl.metadata (28 kB)
Collecting pydub (from LughaatNLP)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
```

**Figure 3: Installation of LughaatNLP library**

Different python libraries are used in this study which are installed to perform various functions. Table 2 lists the name of Python library and their purpose of use. Figure 4 shows the commands used in code to import these necessary libraries.

```
import pandas as pd
import re
import numpy as np
from LughaatNLP import LughaatNLP
urdu_text_processing = LughaatNLP()
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

**Figure 4: Importing necessary libraries**

Library	Purpose
Pandas	For data manipulation such as loading file
Numpy	For numerical computation
re	For regular expression used to remove special characters and punctuation from Urdu text
Matplotlib.pyplot	For visualizing graphs and plots
Seaborn	For visualizing graphs and plots
LughatNLP	For Urdu text preprocessing
Scikit learn	For importing classification report, confusion matrix, accuracy score and train_test split for model.
Tqdm	For tracking progress of model training.
Torch	Pytorch Library to work with tensors
Wordcloud	Used for visualizing word frequency in Urdu text

**Table 2: Python libraries imported in this study**

Spacy library available in Python was used to tokenize Urdu text in separate strings. To install this library, command was used as shown in Figure 5. To generate word cloud to show most frequent words in dataset, WordCloud library was used, which was imported in notebook using the command shown in Figure 5.

```
!pip install spacy

Show hidden output

import pandas as pd
import matplotlib.pyplot as plt
import spacy
from wordcloud import WordCloud
```

**Figure 5: Tools for word cloud generation**

## 2.2 Framework for model implementation

Hugging face provides a range of pretrained transformer models to work with. To use transformer models in our notebook file, transformer library is installed as shown in Figure 6. This installs all the required packages to work with transformers.

```
!pip install transformers torch

Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.3)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transf
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from tra
```

**Figure 6: Hugging Face transformer library**

### 2.2.1 mBERT

Every transformer has its model and tokenizer which are imported from Hugging face library. For mBERT base model implementation, BertTokenizer and BertForSequenceClassification are imported from Hugging face library. AdamW optimizer is also imported from transformer

library. Dataset and DataLoader are also imported to convert our dataset into PyTorch dataset used by model and DataLoader library is used to efficiently feed data into model in batches. Python's torch library provides a deep learning framework for neural network. Scikit learn library provides a method to divide data into train and test. Python's tqdm library is used to monitor progress of model training. All the commands to import these libraries are shown in Figure 7.

```
import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

**Figure 7: Tools for mBERT model**

## 2.2.2 DistilmBERT

Similarly for implementation of DistilmBERT, Hugging Face provides libraries which are imported from transformer module such as DistilBertTokenizer and DistilBertForSequenceClassification, as shown in Figure 8.

```
import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, AdamW
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

**Figure 8: Tools for DistilmBERT model**

## 2.2.3 mT5

To work with Mt5 model, the model and tokenizer are imported in Notebook from Hugging Face transformer library. MT5Tokenizer is tokenizer for multilingual T5 and MT5ForConditionalGeneration is model for mT5. Figure 9 shows the code snippet to import these tools from Hugging Face transformers library.

```
from transformers import MT5Tokenizer, MT5ForConditionalGeneration
import torch
import pandas as pd
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

**Figure 9: Tools for mT5 model**

### 3 Application of tools

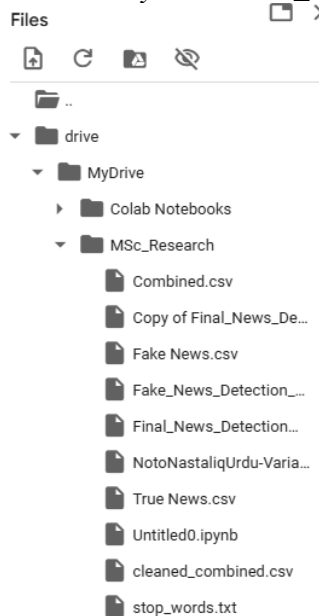
This section provides step by step guide on application of tools and frameworks used in this research.

#### 3.1 Dataset

The dataset used in this research is downloaded from Github repository<sup>2</sup>. This repository has three csv files. Combined.csv contains both fake and real news content whereas Fake News.csv contains only fake news items and Real News.csv contains only real news items. In this study, author has used Combined.csv as it contains both fake and real news. A folder named “MSc\_Research” is created where all data and code files are placed, shown in Figure 10. Dataset file is loaded into Colab notebook with help of Pandas library using following command:

*# loading dataset*

*combine\_df = pd.read\_csv('/content/drive/MyDrive/MSc\_Research/Combined.csv')*



**Figure 10: Project directory**

Matplotlib provides functions to plot graph. For example, Figure 11 shows the distribution of true and fake news based on class label using matplotlib.pyplot library.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
combine_df['Label'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Distribution of True and Fake News')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```

**Figure 11: Barplot of news distribution**

---

<sup>2</sup> <https://github.com/Sheetal83/Ax-to-Grind-Urdu-Dataset>

## 3.2 Preprocessing

An instance of *LughaatNLP* is created named *urdu\_text\_preprocessing* which is used to call various methods for Urdu text preprocessing such as normalization, lemmatization and stop words removal, as shown in Figure 12. All these methods are defined in a function called *preprocess\_text()*. The special characters and punctuation are removed using regular expression which is also shown in Figure.

```
# loading dataset
combine_df = pd.read_csv('/content/drive/MyDrive/MSc_Research/Combined.csv')

def preprocess_text(text):

    # To normalize text
    normalized_text = urdu_text_processing.normalize(text)

    # To lemmatize text
    lemmatized_text = urdu_text_processing.lemmatize_sentence(normalized_text)

    # remove stop words
    filtered_text = urdu_text_processing.remove_stopwords(lemmatized_text)

    # remove special characters and punctuation
    filtered_text = re.sub(r'^\w\s۹-۰۵-۱', '', filtered_text)

    return filtered_text
```

Figure 12: Text preprocessing using *LughaatNLP*

To remove stop words from Urdu for generating word cloud, text file containing all Urdu stop words is downloaded from Github<sup>3</sup>, as shown in Figure 13. This file is uploaded to the “MSc\_Research” folder to use in code as can be seen in Figure 10.

```
with open('/content/drive/MyDrive/MSc_Research/stop_words.txt', 'r', encoding='utf-8') as file:
    stop_words_list = [line.strip() for line in file]

print(stop_words_list)
```

ہوگئے، ہوں، ہوئے، ہوں، ہی، ہیں، وغیرہ، ہے، ی، یا، ہیں، ہوں، یہ، بی، پہل، ے

Figure 13: Urdu stop words

Figure 14 shows the code snippet for application of *Spacy* and *WordCloud* library in the code. *Spacy* provides support for Urdu language processing by specifying *spacy.blank('ur')*. Urdu text is tokenized using *Spacy* library as show in Figure 14. These Urdu tokens are compared with stop words list to remove the stop words from the Urdu text. The word cloud is generated using *WordCloud* method as shown in Figure 14. The Urdu font for word cloud is downloaded from Google Fonts<sup>4</sup> and was saved in “MSc\_Research” folder. Figure 14 also shows the application of that font file.

<sup>3</sup> <https://github.com/SyedMuhammadMuhsinKarim/Urdu-Stop-Words>

<sup>4</sup> <https://fonts.google.com/selection>



```

import spacy
nlp = spacy.blank('ur')

def tokenize(text):
    doc = nlp(text)
    tokens = [token.text for token in doc] # Convert each token to a string
    return tokens

all_text = ' '.join(
    word for text in combine_df['News Items'] for word in tokenize(text) if word not in stop_words_list
)

# Generate the word cloud
wordcloud = WordCloud(font_path='/content/drive/MyDrive/MSc_Research/NotoNastaliqUrdu-VariableFont_wght.ttf',
    background_color='white', width=800, height=400).generate(all_text)

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Most Common Words in Urdu News Texts')
plt.show()

```

**Figure 14: Application of Spacy and WordCloud libraries**

### 3.3 Model implementation

To use data for model, it is first converted into proper format. As shown in Figure 15, NewsDataset custom class uses PyTorch Dataset class which is already imported. It preprocesses the data such as tokenization, padding and truncation. Dataset is prepared into suitable format of PyTorch tensors for model input.

```

# Load and preprocess the dataset
class NewsDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_length,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'label': torch.tensor(label, dtype=torch.long)
        }

```

**Figure 15: Load and preprocess dataset class**

This code snippet in Figure 16 shows the application of DataLoader package and train\_test\_split library that we imported. The dataset is split into training, validation and testing set using train\_test\_split method. These data are processed by DataLoader class to create data loaders for efficient processing of data in batches while training and evaluating model.

```
# Function to load data from CSV and prepare train/test datasets
def load_data(file_path, tokenizer, batch_size=16):
    # Load and preprocess data
    data = pd.read_csv(file_path)
    data['Label'] = data['Label'].str.strip().str.lower()
    data['Label'] = data['Label'].str.capitalize()

    data = data[['News Items', 'Label']].rename(columns={'News Items': 'text', 'Label': 'label'})
    data['label'] = data['label'].map({'Fake': 0, 'True': 1})
    data = data.dropna(subset=['label'])

    # return train_loader, test_loader
    # Split data into train, validation, and test sets
    train_data, temp_data = train_test_split(data, test_size=0.4, random_state=42, stratify=data['label'])
    val_data, test_data = train_test_split(temp_data, test_size=0.5, random_state=42, stratify=temp_data['label'])

    # Create dataset objects for train, validation, and test sets
    train_dataset = NewsDataset(train_data['text'].values, train_data['label'].values, tokenizer)
    val_dataset = NewsDataset(val_data['text'].values, val_data['label'].values, tokenizer)
    test_dataset = NewsDataset(test_data['text'].values, test_data['label'].values, tokenizer)

    # Create data loaders for train, validation, and test sets
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size)
    test_loader = DataLoader(test_dataset, batch_size=batch_size)

    return train_loader, val_loader, test_loader
```

**Figure 16: Preparing data loaders for model**

```
# Training loop
def train_model(model, train_loader, val_loader, test_loader, epoch, lr):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)
    optimizer = AdamW(model.parameters(), lr=lr)
    training_loss = []
    validation_accuracy = []
    training_accuracy = []

    for epoch in range(epochs):
        model.train()
        total_train_loss = 0
        correct_predictions = 0
        total_predictions = 0

        # Training loop
        for batch in tqdm(train_loader, desc=f"Training Epoch {epoch + 1}"):
            optimizer.zero_grad()
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss
            total_train_loss += loss.item()
            loss.backward()
            optimizer.step()
```

**Figure 17: Training model**

This code snippet in Figure 17 illustrates class created to train the model. It shows application of tqdm library which tracks the training performance of model over all epochs. The code also shows command to use GPU processor instead of CPU processor. As discussed, this study uses T4 GPU then this command activates the GPU and shifts all operations to GPU (Cuda). It can also be seen that AdamW is used as optimizer which we already have imported.

### 3.3.1 mBERT Implementation

Model name from Hugging Face library for mBERT base version is “*bert-base-multilingual-cased*”. This exact string can be found on Hugging Face website<sup>5</sup>. This pretrained mBERT base model is initialized with BertForSequenceClassification class as shown in Figure 18. The function load\_data() is already explained above which takes the csv file, in this case cleaned Urdu dataset and uses BERT tokenizer to tokenize the Urdu text and prepares train loader, validation loader and test loader. The train\_model() function trains the model for 10 epochs using train data, validates the model using validation data.

```
# Load tokenizer and model
model_name = "bert-base-multilingual-cased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Prepare data loaders
train_loader, val_loader, test_loader = load_data('/content/drive/MyDrive/MSc_Research/cleaned_combined.csv', tokenizer)

# Train the model
training_loss, validation_accuracy, training_accuracy = train_model(model, train_loader, val_loader, test_loader, epochs=10)
```

Figure 18: mBERT tools application

### 3.3.2 DistilBERT Implementation

The pre-trained multilingual distilled version of BERT model imported from Hugging Face library is “*distilbert-base-multilingual-cased*”. The exact string can be found on their website<sup>6</sup>. This pretrained model is initialized with DistilBERTForSequenceClassification as can be seen in Figure 19. Model tokenizer DistilBertTokenizer takes the cleaned Urdu text and converts into token. The number of labels is 2 because model predicts two classes, fake and real news.

```
# Load tokenizer and model
model_name = "distilbert-base-multilingual-cased"
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
model = DistilBertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Load data and prepare DataLoader
train_loader, val_loader, test_loader = load_data('/content/drive/MyDrive/MSc_Research/cleaned_combined.csv', tokenizer)

# Train the model
training_loss, validation_accuracy = train_model(model, train_loader, val_loader, test_loader, epochs=10)
```

Figure 19: DistilBERT tools application

---

<sup>5</sup> <https://huggingface.co/google-bert/bert-base-multilingual-cased>

<sup>6</sup> <https://huggingface.co/distilbert/distilbert-base-multilingual-cased>

### 3.3.3 mT5 Implementation

The application of imported tools from Hugging Face library for multilingual T5 (mT5) is shown in Figure 20. The pretrained model “google/mt5-small” is initialized by MT5ForConditionalGeneration class. The exact model’s name can be found on Hugging face website<sup>7</sup>. MT5Tokenizer tokenizes the Urdu text into input embeddings.

```
model_name = "google/mt5-small"
tokenizer = MT5Tokenizer.from_pretrained(model_name)
model = MT5ForConditionalGeneration.from_pretrained(model_name)
```

Figure 20: mT5 implementation

One thing to note here is that initially batch size was kept 16 for DataLoader for mT5 implementation. However, it showed error “OutOfMemoryError” as shown in Figure 21 which means 16GB RAM allowed by T4 GPU was exhausted to process lot of data simultaneously. When batch size was reduced to 8, error was rectified.

```
0% | 0/757 [00:00<?, ?it/s]

Traceback (most recent call last)
<ipython-input-68-7faedca479cc> in <cell line: 1>()
----> 1 train_model(model, tokenizer, train_loader, val_loader, epochs=2, lr=5e-5)

3 frames
/usr/local/lib/python3.10/dist-packages/torch/autograd/graph.py in _engine_run_backward(t_outputs, *args, **kwargs)
    823     unregister_hooks = _register_logging_hooks_on_whole_graph(t_outputs)
    824     try:
--> 825         return Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass
    826             t_outputs, *args, **kwargs
    827         ) # Calls into the C++ engine to run the backward pass

OutOfMemoryError: CUDA out of memory. Tried to allocate 978.00 MiB. GPU 0 has a total capacity of 14.75 GiB of which 815.06 MiB is free. Process 4177 has 13.95 GiB memory in use. Of the allocated memory 13.64 GiB is allocated by PyTorch, and 175.76 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting
PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation. See documentation for Memory Management
(https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

Figure 21: RAM Error

## 3.4 Evaluation

Figure 22 shows application of libraries imported for evaluation. The method classification\_report() prints the classification report of predictions made by model. The method confusion\_matrix outputs the confusion matrix which shows classification results in matrix format between actual and predict labels.

```
def display_metrics_and_confusion_matrix(labels, preds, label_names):
    print("Classification Report:")
    print(classification_report(labels, preds, target_names=label_names))

    cm = confusion_matrix(labels, preds, labels=label_names)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_names, yticklabels=label_names)
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.title("Confusion Matrix")
    plt.show()
```

Figure 22: Model evaluation metrics

<sup>7</sup> <https://huggingface.co/google/mt5-small>