Configuration Manual

MSc Research Project
MSCDADJAN24_O

ABIN JOSE
Student ID: x23195681

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Abin Jose……………………………………………………………………………………

Student ID: …x23195681……………………………………………………………………………………

Programme: …MSCDADJAN24_O……………………………… **Year:** ……2024-2025……

Module: …MSc Research Project ………………………………………………………………………..

Supervisor: …Jaswinder Singh………………………………………………………………………………

Submission Due Date: …29/01/2025……………………………………………………………………………………
…

Project Title: …Enhancing Hate Speech Detection in social media using XLNet and Graph Convolutional Networks: Sentiment Analysis

Word Count: ……………1012………………… **Page Count**………09……………………………………

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: ………ABIN JOSE……………………………………………………………………………………

Date: …………29/01/2025……………………………………………………………………………
……

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

Abin Jose
x23195681

1 Introduction

Hate speech on social media has become a ubiquitous problem, causing real-life harm and upending online spaces. Such concerns lead to the exploration of advanced methods for accurately identifying hate speech, which often exhibits context-sensitive language, sarcasm, or coded phrases. We performed a lot of preprocessing such as removing noise, tokenizing text, doing sentiment analysis on datasets from Hatebase and Kaggle. We combine traditional machine learning models (Logistic Regression, SVM) with deep learning architectures (RNN, CNN) and transformer-based models (BERT, XLNet). This Configuration manual includes the steps and system requirements of the complete project.

2 System Requirements

Hardware Requirements

Processor: Minimum 8-core CPU (for example, Intel i7 or AMD Ryzen 7);
Recommended: System with a GPU, supporting NVIDIA CUDA.

Memory: 16 GB RAM minimum; 32 GB RAM recommended for big datasets.

Storage: It is recommended that a minimum of 50 GB free space be available for datasets, logs, and model artifacts. Make sure high-speed SSD storage is used.

GPU: The recommended one is a NVIDIA one with at least 8 GB of VRAM. A model such as the NVIDIA RTX 3060 or a higher model is recommended. Working with a GPU helps a lot while training deep learning models, thus allowing for much faster training and, altogether, better results.

Software Requirements

Operating System: Linux - Ubuntu 20.04 or later, Windows 10/11. (Preferentially use Linux-based systems as their interaction with deep learning frameworks will be better).

Python Version: Python 3.8 or greater. This allows working with most of the available libraries and frameworks.

Package Manager: pip or conda to manage the dependencies efficiently. Conda can be used more effectively for creating isolated environments.

Development Environment: This will be both on Google Colab and Jupyter Notebook. The former provides a free cloud-based GPU runtime, making it very suitable for experimenting and testing when particularly powerful local hardware is not required.

3 Environment Setup

Google Colab

Google Colab is a web-based platform that allows you to run and write Python code in the browser. It supports GPU acceleration, hence making it a very great choice for machine learning and data science projects.

Open Google Colab:

Go to Google Colab.

Click "File > New Notebook" to create a new one.

Set Up GPU Runtime:

From the top menu bar, select Runtime > Change runtime type.

From the "Hardware accelerator" dropdown menu, select GPU. Click Save to save your changes. This will guarantee that your code uses a GPU for faster computations.

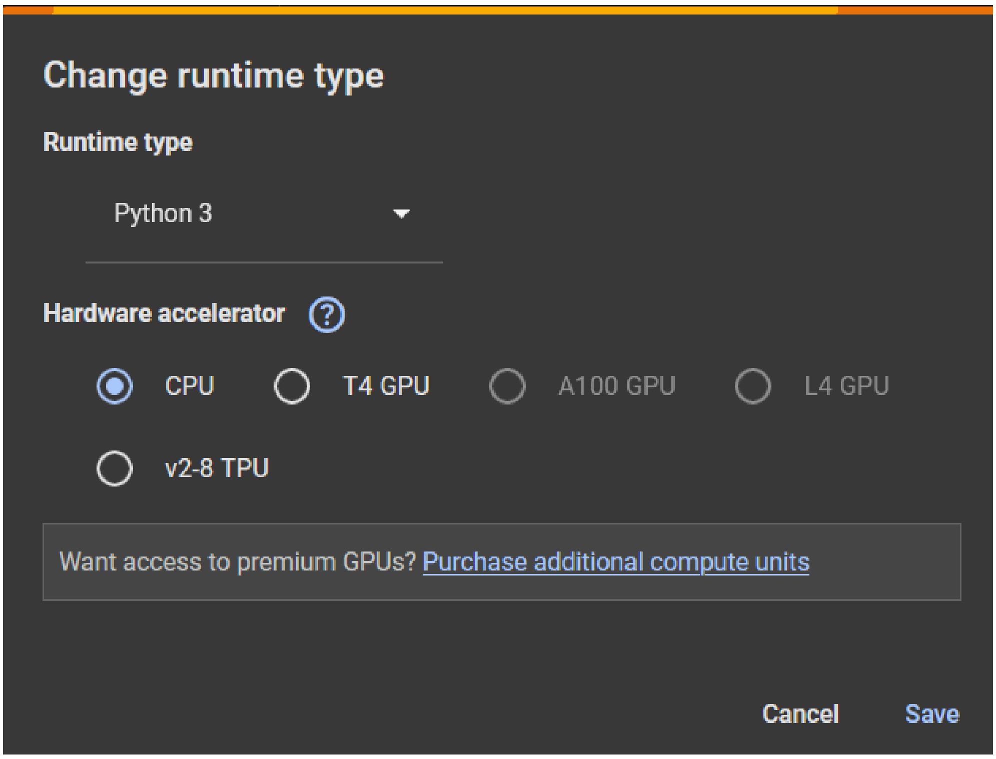

Figure 1. Runtime Types in Google Colab

Install Dependencies:

You can install the required libraries using the following command in a cell in your Colab notebook from a requirements.txt file:

!pip install -r requirements.txt

Alternatively, libraries can be installed individually with a pip install command. Follow the proper installation of every dependency, including TensorFlow and PyTorch, as well as scikit-learn, to name a few dependencies.

Key dependencies:

pandas, numpy: These are required for data manipulation and preprocessing.

Scikit-learn: useful in implementation for basic machine learning.

Tensorflow, torch: For building and training deep learning models.

VaderSentiment is used for sentiment analysis.

matplotlib, seaborn: for plotting and result analysis.

Upload Your Dataset:
Upload your dataset using Colab's upload functionality.

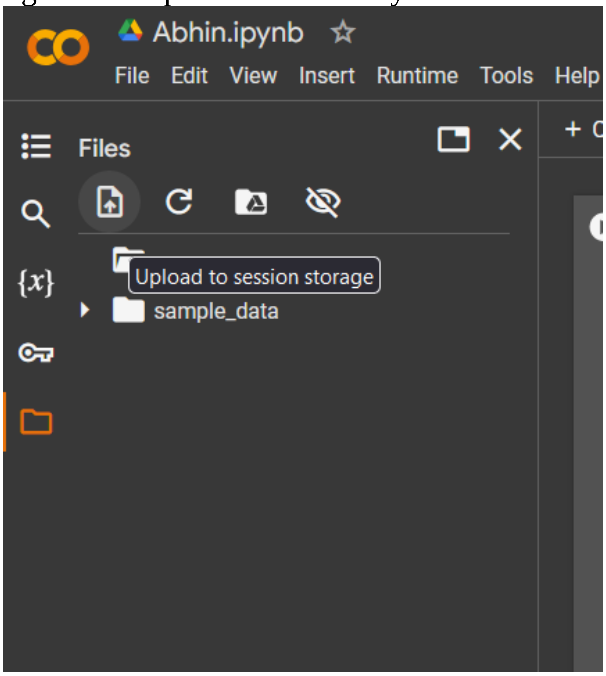

Figure 2. Option to Upload files in Colab

Alternatively, you may mount your Google Drive, and then access the files from there:

from google.colab import drive drive.mount('/content/drive')

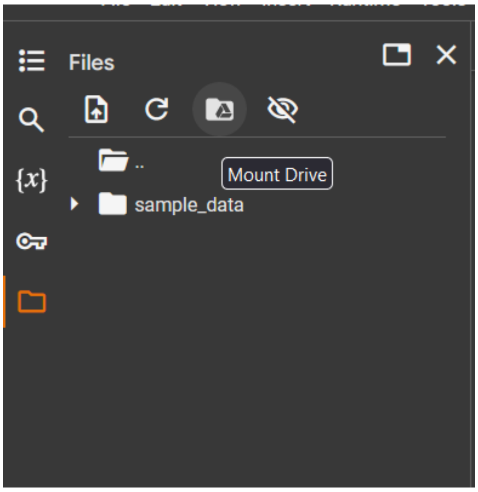

Figure 3. Option to mount google drive in Colab

4 Testing the Setup

First, one should ensure that the environment is set up correctly before commencing the project. Running a few tests helps confirm everything is functional.

1. Verify Package Installation:

Add a cell in your notebook to check the installed packages and their versions:

```
!pip list
```

This command outputs the list of installed libraries so you can check if they are present or not.

2. Run Preliminary Tests:

Then add a test cell that checks required libraries and dependencies have been installed:

```
# Sample test script

import tensorflow as tf

print("TensorFlow Version:", tf.__version__)
```

This will display the version number of TensorFlow if it has been installed correctly.

3. GPU Verification:

To make sure that the GPU is available and accessible, run the following code:

```
import tensorflow as tf

print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

This should output the list of available GPUs. If no GPU is detected, please refer to the runtime settings or hardware configuration.

If you go through the steps above, your Google Colab environment will be ready to execute the hate speech detection project without any problems. Remember to save your work frequently in Colab.

5 Data Preparation

	A	B	C	D	E	F	G
1		count	hate_spee	offensive_	neither	class	tweet
2	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. & as a man you should always take the trash out...
3	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn bad for cuffin dat hoe in the 1st place!!
4	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby4life: You ever fuck a bitch and she start to cry? You be confused as shit
5	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she look like a tranny
6	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you hear about me might be true or it might be faker than the bitch who told it to ya 
7	5	3	1	2	0	1	!!!!!!!!!!!!!!!!!"@T_Madison_x: The shit just blows me..claim you so faithful and down for somebody but still fucking with hoes! 😂😂😂"
8	6	s	0	3	0	1	!!!!!!"@__BrighterDays: I can not just sit up and HATE on another bitch .. I got too much shit going on!"
9	7	3	0	3	0	1	!!!!“@selfiequeenbri: cause I'm tired of you big bitches coming for us skinny girls!!”

Figure 4: Dataset sample

The Dataset used in the project contains more than 20000 observations and and 7 variables.

```python
def clean_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#', '', text)
    text = re.sub(r'[^A-Za-z\s]', '', text)
    return text.lower()

df['cleaned_text'] = df['tweet'].apply(clean_text)

df['tokens'] = df['cleaned_text'].apply(lambda x: x.split())

df = df.dropna(subset=['cleaned_text', 'class'])

label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['class'])

X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'], df['label'], test_size=0.2, random_state=42)

print("Training and testing data prepared!")

df.to_csv('preprocessed_hate_speech_data.csv', index=False)
```

Figure 5: Data cleaning and Preprocessing

The figure 5 shows how the data is prepossessed and cleaned.

```python
import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

df = pd.read_csv('preprocessed_hate_speech_data.csv')

analyzer = SentimentIntensityAnalyzer()

def get_sentiment_score(text):
    sentiment = analyzer.polarity_scores(text)
    return sentiment['compound']

df['sentiment_score'] = df['cleaned_text'].apply(get_sentiment_score)

print(df[['cleaned_text', 'sentiment_score']].head())

X_train, X_test, y_train, y_test = train_test_split(
    df[['cleaned_text', 'sentiment_score']],
    df['label'],
    test_size=0.2,
    random_state=42
)

df.to_csv('hate_speech_with_sentiment.csv', index=False)

print("Sentiment scores added and dataset updated!")
```

Figure 6: sentiment scores

6 Experiments

```python
log_reg_model = LogisticRegression(max_iter=200)
log_reg_model.fit(X_train, y_train)

y_pred = log_reg_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"F1 Score: {f1:.4f}")

import joblib
joblib.dump(log_reg_model, 'logistic_regression_hate_speech_model_tfidf.pkl')
```

Figure 7: Logistic regression model

```python
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train, y_train)
svm_preds = svm_model.predict(X_test)

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)
```

Figure 8: SVM and Random Forest

Multiplicative algorithms provide robustness and can be adapted to various areas of interest. The first one is a comparison established by performance benchmarking with traditional machine learning models such as Logistic Regression and SVM.

```python
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['cleaned_text'])

X_seq = tokenizer.texts_to_sequences(df['cleaned_text'])
X_padded = pad_sequences(X_seq, maxlen=max_len)
y_encoded = df['label']

X_train, X_test, y_train, y_test = train_test_split(X_padded, y_encoded, test_size=0.2, random_state=42)

model_rnn = Sequential()
model_rnn.add(Embedding(max_words, 128, input_length=max_len))
model_rnn.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model_rnn.add(Dense(1, activation='sigmoid'))

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

model_rnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2, verbose=1)

rnn_preds = (model_rnn.predict(X_test) > 0.5).astype("int32")
print("RNN Accuracy:", accuracy_score(y_test, rnn_preds))
```

Figure 9: RNN

```python
from keras.layers import Conv1D, GlobalMaxPooling1D

model_cnn = Sequential()
model_cnn.add(Embedding(5000, 128, input_length=max_len))
model_cnn.add(Conv1D(64, kernel_size=5, activation='relu'))
model_cnn.add(GlobalMaxPooling1D())
model_cnn.add(Dense(1, activation='sigmoid'))

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

model_cnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.2, verbose=1)

cnn_preds = (model_cnn.predict(X_test) > 0.5).astype("int32")
print("CNN Accuracy:", accuracy_score(y_test, cnn_preds))
```

Figure 10: CNN

Deep learning architectures are particularly popular for text classification, with RNNs used to capture sequential patterns and CNNs to capture local patterns in text

```
)
# Define optimizer
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=3e-5)

# Compile the model
bert_model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Train the model
history = bert_model.fit(
    train_encodings['input_ids'],
    y_train,
    epochs=3,
    batch_size=64,
    validation_split=0.1
)

# Predictions
bert_preds = bert_model.predict(test_encodings['input_ids']).logits
bert_preds = tf.argmax(bert_preds, axis=1)

# Evaluation
print("BERT Multi-Class Accuracy:", accuracy_score(y_test, bert_preds))
print("\nClassification Report:")
print(classification_report(y_test, bert_preds))
```

Figure 11: BERT

BERT and XLNet use their advanced contextual comprehension to address such nuances in hate speech using transformer-based structures.

```
from transformers import XLNetTokenizer, TFXLNetForSequenceClassification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'], df['label'], test_size=0.2, random_state=42)
xlnet_tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased')

train_encodings = xlnet_tokenizer(list(X_train), truncation=True, padding=True, max_length=100, return_tensors='tf')
test_encodings = xlnet_tokenizer(list(X_test), truncation=True, padding=True, max_length=100, return_tensors='tf')
xlnet_model = TFXLNetForSequenceClassification.from_pretrained('xlnet-base-cased', num_labels=3)
xlnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
history = xlnet_model.fit(
    train_encodings['input_ids'],
    y_train,
    epochs=3,
    batch_size=256,
    validation_split=0.1
)
xlnet_preds = xlnet_model.predict(test_encodings['input_ids']).logits
xlnet_preds = tf.argmax(xlnet_preds, axis=1)

print("XLNet Multi-Class Accuracy:", accuracy_score(y_test, xlnet_preds))
print("\nClassification Report:")
print(classification_report(y_test, xlnet_preds))
```

Figure 12: XLNET

The modular design makes the system scalable and adaptable, with new models easily integrated. Due to their computational needs, platforms like Google Colab and AWS are usually enough to train and test these models.

References

Keras Team. (2023). Keras documentation. Retrieved from https://keras.io

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

TensorFlow Developers. (2023). TensorFlow documentation. Retrieved from https://www.tensorflow.org

Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. Scotts Valley, CA: CreateSpace.

Waskom, M., et al. (2021). Seaborn: Statistical data visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021