National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Alex Sunny
Student ID: x23304600

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Alex Sunny
Student ID:	x23304600
Programme:	MSc Data Analytics **Year:** 2024-2025
Module:	MSc Research Practicum
Lecturer:	Arjun Chikkankod
Submission Due Date:	11/08/2025
Project Title:	Towards Trustworthy Healthcare AI: Multi-Method Explainable AI for Transparent Adverse Drug Reaction Detection from Patient-Generated Text
Word Count:	711 **Page Count:** 5

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:	Alex Sunny
Date:	11/08/2025

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

Alex Sunny
x23304600

1 Introduction

This document provides all the details of the hardware and software platforms and environments used to realize the research objective to integrated explainable AI (XAI) in Adverse Drug Reaction (ADR) detection systems. As part of this study multiple Machine Learning (ML) and Deep Learning (DL) models and XAI techniques were implemented using Python on the Google Colaboratory environment.

2 System Configuration

To implement the project code and evaluate the results, Python programming was used along with its numerous libraries to implement the XAI techniques and the ADR detection models. The details of the cloud-based computing environment and software setup that was used is as follows.

2.1 Hardware Configuration

All the model executions and experiments were implemented on the cloud-based Google Colaboratory, given its flexibility of computing power, GPU resources and extensive libraries for support.
Runtime Type: Python 3.11.13
System RAM: 12.67 GB
Disk Storage: 108 GB temporary storage
Compute units: Based on availability and need
Network: Cloud infrastructure provided by Google

2.2 Software Configuration

Programming language: Python 3.11.13
CUDA Version: 12.5 (for GPU needs)
Storage: Google Drive API for dataset storage and access
Browser: Google Chrome Version 138.0.7204.184 (to access the Colab environment)

All the code training, development, evaluation and integration of explainability were conducted in the Google Colab interface for a scalable, consistent and GPU accelerated workflow.

3 Project Implementation

3.1 Dataset Collection and Preparation

Dataset source: US Food and Drug Administration (FDA) Drug Reviews Dataset (2024)

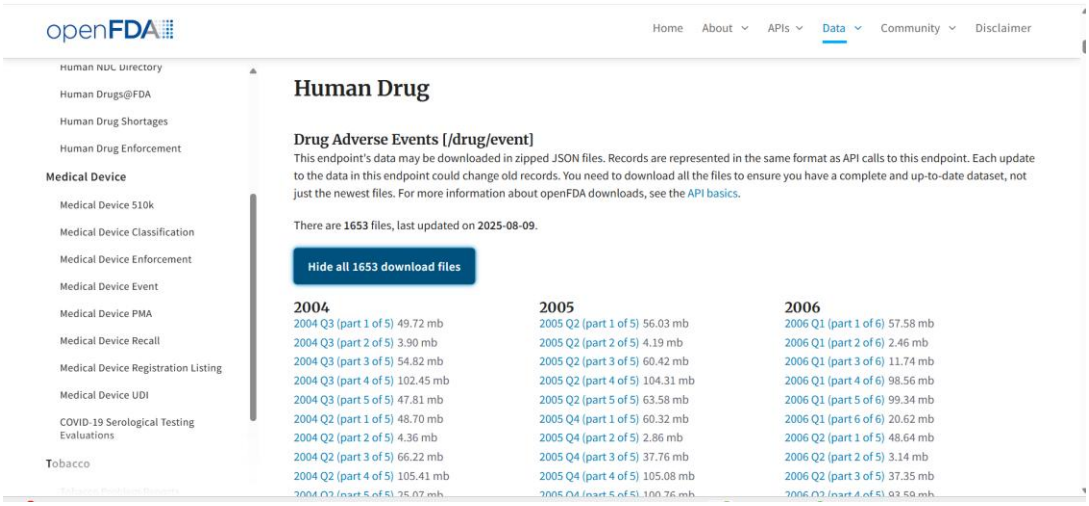

Fig 1. Dataset source – US Open FDA

Format: Excel file
Size: 161,297 records
Storage location: Google Drive Folder

Dataset Structure:
Features

1) urlDrugName – Name of the drug
2) rating - Rating or score from 1 to 10 provided by the patient
3) effectiveness – how effective the patient found the treatment drug
4) benefitsReview – Textual review of the benefits
5) sideeffectsReview – Side effects text review
6) commentsReview – Any general comments given by the patient
7) condition – The medical condition being treated
8) sideEffects – Category of the side effect

Data Mounting and Loading
The dataset was first mounted to a folder in Google Drive.

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

     Mounted at /content/drive
```

Fig 2. Mounting Google Drive

Then it was loaded for processing into the Colab environment.

```
# Load dataset
df = pd.read_excel("/content/drive/MyDrive/Research practicum dataset/drugLibTrain_raw.xls")
```

Fig 3. Loading the dataset into a dataframe

All the essential libraries were installed.

Dataset repository - https://open.fda.gov/data/faers/

```
import pandas as pd
import numpy as np
import re
import string
import nltk
import spacy
import contractions
import emoji
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
```

```
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional, SpatialDropout1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.utils import class_weight
```

Fig 4. Essential libraries

3.2 Python Libraries Utilized

Multiple libraries were imported and installed for data processing, model building and XAI implementation.

1) NumPy – To create arrays and perform the mathematical calculations
2) Pandas – To clean, analyse and manipulate the drug review data frame
3) Scikit-learn – To perform multiple functionalities like
   sklearn.ensemble: For random forest implementation
   sklearn.model_selection: For train-test split and cross validation
   sklearn.inspection: For partial dependence plots
4) TensorFlow – To implement the LSTM model
5) XGBoost – To implement the XGBoost classifier
6) NLTK- For text pre-processing and stopword removal
7) spaCy – For lemmatization and negation handling
8) XAI Libraries
   LIME – For implementing the LIME explainer for local explanations
   SHAP – For implementing the SHAP explainer
   TreeInterpreter – For explaining the ensemble ML models

3.3 Model Development

The Random Forest, XGBoost and LSTM models were developed.
A train-test split of 80% and 20% respectively was performed. The hold-out validation with early stopping was implemented. The Adam optimizer was used for the LSTM model.

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report

# Random Forest
rf_model = RandomForestClassifier(class_weight='balanced', n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)
print("Random Forest:\n", classification_report(y_test, rf_preds))

# XGBoost
xgb_model = XGBClassifier(scale_pos_weight=(len(y_train[y_train == 0]) / len(y_train[y_train == 1])), eval_metric='logloss')
xgb_model.fit(X_train, y_train)
xgb_preds = xgb_model.predict(X_test)
print("XGBoost:\n", classification_report(y_test, xgb_preds))
```

Fig 5. Random Forest and XGboost models

```
model = Sequential([
    # Embedding layer with masking for padded sequences
    Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen, mask_zero=True),

    # Spatial dropout for embedding layer
    SpatialDropout1D(0.3),

    # First bidirectional LSTM layer (returns sequences)
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.3, recurrent_dropout=0.2)),

    # Second bidirectional LSTM layer (final output)
    Bidirectional(LSTM(32, return_sequences=False, dropout=0.3, recurrent_dropout=0.2)),

    # Dense layers with dropout for classification
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

# Compile model with custom optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(
    loss='binary_crossentropy',
```

Fig 6. LSTM model

4 Steps for Recreating the Implementation

4.1 Environment setup

1) Access Google Colab through any web browser
2) Create a new Python 3 Notebook
3) Mount the Google Drive folder
4) Install all required libraries

4.2 Data Preparation

1) Upload the drug reviews dataset to Google Drive folder
2) Load the dataset suing pandas
3) Execute the pre-processing pipeline
4) Apply negation handling with spaCy
5) Generate the TF-IDF features and combine them with the numerical features

4.3 Model Implementation

1) Implement the three models (RF, XGBoost and LSTM)
2) Evaluate model performance using the metrics
3) Save the trained models for XAI integration

4.4 XAI Integration

1) Configure the XAI methods (LIME, SHAP, Integrated Gradients, TreeInterpreter and Partial Dependence Plots) with appropriate parameters
2) Generate the explanations for sample predictions

3) Evaluate the XAI methods using the quantitative metric (faithfulness, fidelity and complexity)
4) Create any visualization needed for interpretation analysis.

```
!pip install lime

from lime.lime_text import LimeTextExplainer
import numpy as np

class_names = ['No ADR', 'ADR']

# Wrap model prediction function to accept raw text and return probability for each class
def predict_proba(texts):
    sequences = tokenizer.texts_to_sequences(texts)
    padded = pad_sequences(sequences, maxlen=200)
    probs = model.predict(padded)
    # Return probabilities for both classes: [prob_no_adr, prob_adr]
    probs_2class = np.hstack([1 - probs, probs])
    return probs_2class

explainer = LimeTextExplainer(class_names=class_names)

# Pick a sample text to explain
idx = 0
text_instance = df["lemmatized_review"].iloc[idx]

exp = explainer.explain_instance(text_instance, predict_proba, num_features=10)
```

Fig 7. LIME Implementation

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.sequence import pad_sequences

def integrated_gradients(inputs_emb, sub_model, baseline=None, steps=50):
    if baseline is None:
        baseline = np.zeros_like(inputs_emb).astype(np.float32)

    interpolated_inputs = [baseline + (float(i) / steps) * (inputs_emb - baseline) for i in range(steps + 1)]
    grads = []
    for x in interpolated_inputs:
        x = tf.convert_to_tensor(x)
        with tf.GradientTape() as tape:
            tape.watch(x)
            pred = sub_model(x)
        grad = tape.gradient(pred, x)
        if grad is not None:
            grads.append(grad.numpy())
        else:
            grads.append(np.zeros_like(x.numpy()))

    avg_grads = np.average(grads[:-1], axis=0)
    integrated_grads = (inputs_emb - baseline) * avg_grads
    return integrated_grads
```

Fig 8. Integrated Gradients Implementation