National College of Ireland

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Metehan Dilekmen
Student ID: 23308907

School of Computing
National College of Ireland

Supervisor: Arundev Vamadevan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Metehan Dilekmen		
Student ID:	23308907		
Programme:	MSc in Artificial Intelligence	**Year:**	2024/2025
Module:	MSc Research Project		
Lecturer:	Arundev Vamadevan		
Submission Due Date:	11/08/2025		
Project Title:	Classifying, Understanding, and Mimicking Driver Behaviour from Sensor Data Using LSTM-Based Models for Autonomous Driving		
Word Count:	700	**Page Count: 6**	

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: Metehan Dilekmen

Date: 10.08.2025

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

Metehan Dilekmen
Student ID: 23308907

1 Introduction

Your first section. Change the header and label to something appropriate. This document explains the environment setup required to run an LSTM-based model project that classifies and mimics driver behaviour. The hardware, software, data sets, and model training steps are explained step by step. By following these steps, it is possible to set up and run a system that recognises aggressive, normal, and drowsy driving behaviour from scratch.

2 System Specifications

Figure 1. Computer hardware information

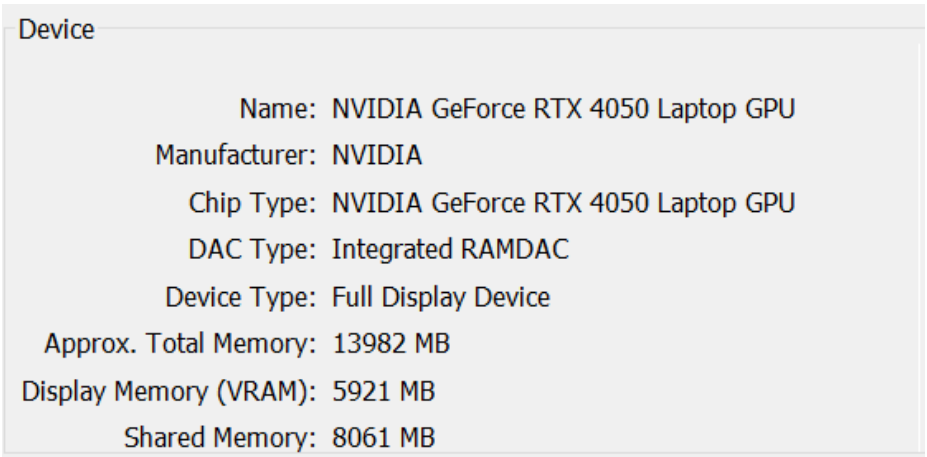

Figure 2. GPU specifications

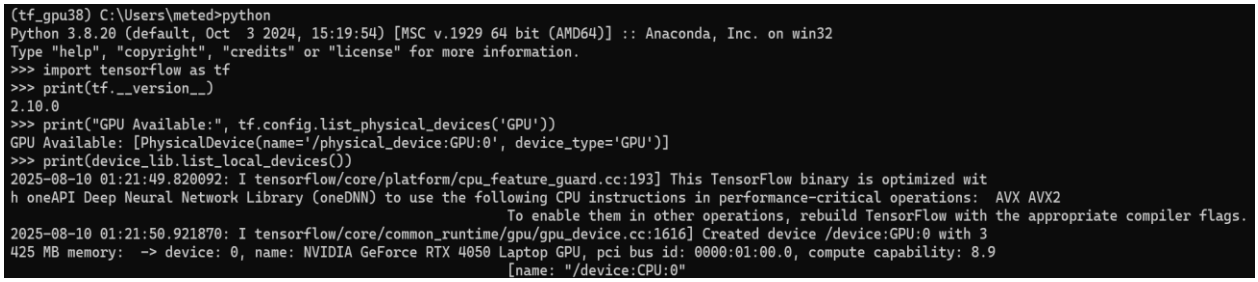

Figure 3: Python environment and TensorFlow GPU configuration output

Figure 1. This image displays the basic specifications of the computer. It is running Windows 11 Home operating system, an Intel Core i7-13700H processor, and 16 GB of RAM.

Figure 2. This image shows the graphics card used in the computer. It has an NVIDIA GeForce RTX 4050 Laptop GPU with 5921 MB VRAM capacity. This GPU was used to shorten the training time of the model and speed up the processes.

Figure 3. This image shows that TensorFlow recognises the GPU in the Python environment. Python 3.8.20 and TensorFlow 2.10 were used. When setting up this environment, the CUDA and cuDNN libraries suitable for the TensorFlow version were also installed. Thus, the project was prepared to run smoothly with GPU support.

3 Dataset specification

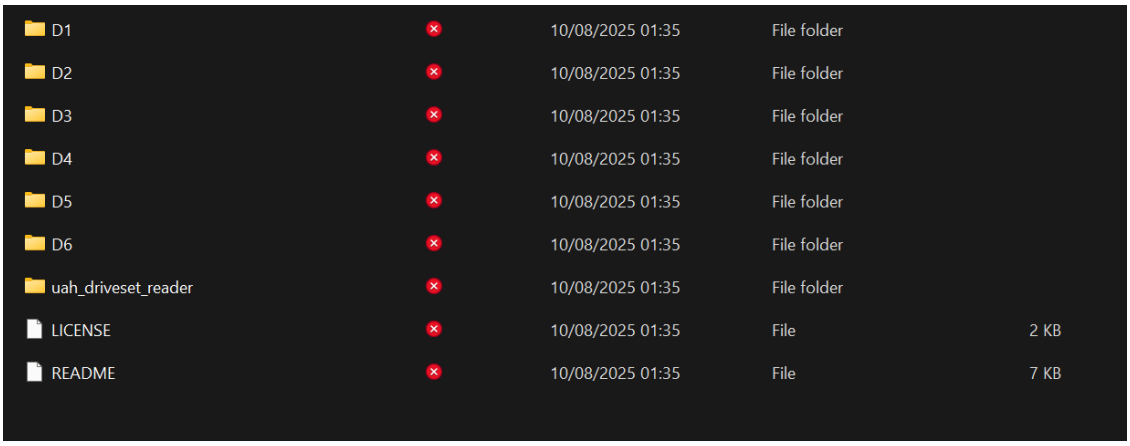

Figure 4. Drivers

Figure 5. Driving Records for a Sample Driver

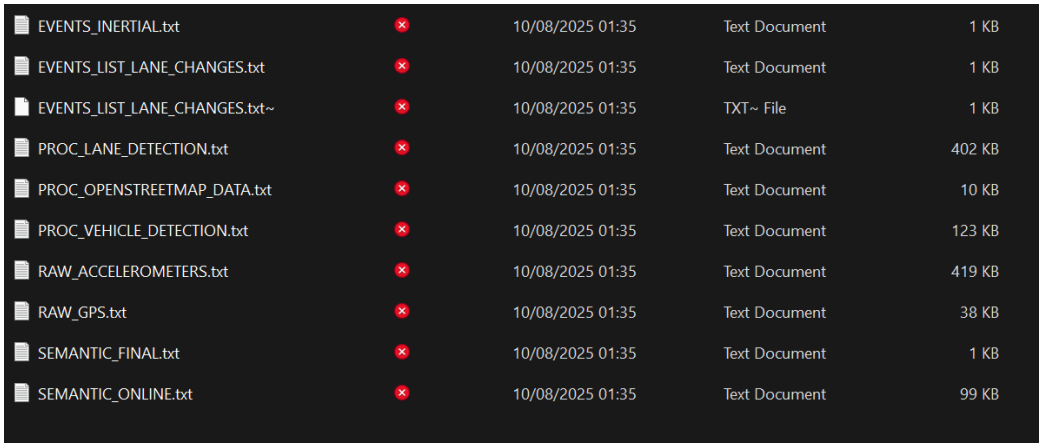

Figure 6. Files for a Sample Driving Records File

Figure 4. Main driver folders in the data set. Each folder contains data from a different driver.

Figure 5. Different driving records for a sample driver. Folder names contain information about the date, distance, type (Normal, Aggressive, Drowsy) and road type of the drive.

Figure 6. Data files contained in a selected driving record. Here, raw sensor data (RAW_ACCELEROMETERS.txt, RAW_GPS.txt) and processed data (PROC_* files) are found. Additionally, event lists related to the driving scenario and road information are included.

The UAH-DriveSet data set was used in this project. The data set contains real sensor data collected from the driving of different drivers in various scenarios. Each driver folder contains multiple driving records. In each driving record, information such as accelerometer, GPS, lane change, and vehicle detection is stored in separate files. Within the scope of the project, these files were merged into a single dataset, and this combined dataset was used in the analyses. The dataset is publicly available and can be downloaded from http://www.robesafe.uah.es/personal/eduardo.romera/uah-driveset/

4 Project Development

4.1 Libraries

```
import glob
import os
import random
from collections import defaultdict
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from geopy.distance import geodesic
from scipy.stats import randint
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, average_precision_score, classification_report, \
    confusion_matrix, f1_score, precision_score, recall_score, roc_curve, auc, precision_recall_curve
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, label_binarize
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, Conv1D, MaxPooling1D
from tensorflow.keras.models import Sequential, load_model
```

Figure 7. Shows the libraries that are used in the project

Figure 7. Python libraries used in the project. The necessary packages for data processing, visualisation, machine learning, and deep learning are included.

4.2 Data Merging

Figure 8 & Figure 9. This code scans all driver folders and driving sessions in the data set and reads the accelerometer (RAW_ACCELEROMETERS.txt) and GPS (RAW_GPS.txt) data. The data is matched based on the time column and combined into a single table. For each record, the driver ID, session name, road type, and total distance information are added. All sessions are combined and stored in the all_drive_data.csv file.

```
# Root directory of the extracted dataset
root_dir = "./UAH-DRIVESET-v1"

# Store merged data for all sessions
all_sessions = []

# Loop through each driver folder (D1 to D6)
for driver_id in range(1, 7):
    driver_folder = os.path.join(root_dir, f"D{driver_id}")
    if not os.path.exists(driver_folder):
        continue

    # Loop through each driving session
    for session_path in glob.glob(os.path.join(driver_folder, "*")):
        accel_path = os.path.join(session_path, "RAW_ACCELEROMETERS.txt")
        gps_path = os.path.join(session_path, "RAW_GPS.txt")

        print(f"Processing session: {session_path}")

        if os.path.exists(accel_path) and os.path.exists(gps_path):
            try:
                # Read accelerometer data
                accel_df = pd.read_csv(accel_path, sep="\s+", header=None, engine='python')
                gps_df = pd.read_csv(gps_path, sep="\s+", header=None, engine='python')

                if accel_df.shape[1] < 4:
                    raise ValueError("RAW_ACCELEROMETERS.txt has insufficient columns")
                if gps_df.shape[1] < 5:
                    raise ValueError("RAW_GPS.txt has insufficient columns")

                # Format accelerometer columns
                accel_df = pd.concat([accel_df.iloc[:, 0], accel_df.iloc[:, -3:]], axis=1)
                accel_df.columns = ["time", "accel_x", "accel_y", "accel_z"]

                # Use correct GPS columns (time, speed, latitude, longitude)
                gps_df = gps_df.iloc[:, [0, 1, 2, 3]]
                gps_df.columns = ["time", "speed", "latitude", "longitude"]

                # Round time for merging
                accel_df["time"] = accel_df["time"].round(2)
                gps_df["time"] = gps_df["time"].round(2)

                # Merge on time
                merged_df = pd.merge_asof(accel_df.sort_values("time"),
                                          gps_df.sort_values("time"),
                                          on="time", direction="nearest", tolerance=0.05)
                merged_df.dropna(inplace=True)
```

```
                if not merged_df.empty:
                    merged_df['driver'] = f"D{driver_id}"
                    merged_df['session'] = os.path.basename(session_path)

                    # Extract road_type from session name
                    session_name = merged_df['session'].iloc[0].lower()
                    if "motorway" in session_name:
                        merged_df['road_type'] = "MOTORWAY"
                    elif "secondary" in session_name:
                        merged_df['road_type'] = "SECONDARY"
                    else:
                        merged_df['road_type'] = "UNKNOWN"

                    # Calculate distance in km
                    coords = list(zip(merged_df['latitude'], merged_df['longitude']))
                    total_distance = sum(geodesic(coords[i], coords[i + 1]).km for i in range(len(coords) - 1))
                    merged_df['distance_km'] = round(total_distance)

                    all_sessions.append(merged_df)

            except Exception as e:
                print(f"Error in session {session_path}: {e}")
                continue

print(f"Total valid sessions collected: {len(all_sessions)}")

# Save final dataset
if all_sessions:
    final_dataset = pd.concat(all_sessions, ignore_index=True)

    # Reorder columns
    final_dataset = final_dataset[[
        "time", "accel_x", "accel_y", "accel_z",
        "latitude", "longitude", "speed",
        "driver", "session", "road_type", "distance_km"
    ]]

    print("Final dataset shape:", final_dataset.shape)
    final_dataset.to_csv("all_drive_data.csv", index=False)
    print("Dataset saved as all_drive_data.csv")
else:
    print("No valid session data found.")
```

Figure 8, 9. Data Merging Process

4.3 Machine Learning Models

```python
# Train and evaluate Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Figure 10. Random Forest Train

```python
# Define feature columns
features = [
    'accel_x', 'accel_y', 'accel_z',
    'speed', 'speed_delta',
    'jerk_x', 'jerk_y', 'jerk_z', 'jerk_magnitude',
    'accel_magnitude', 'accel_magnitude_roll_var',
    'accel_x_roll', 'accel_y_roll', 'accel_z_roll',
    'speed_roll', 'speed_stability'
]

# Create feature matrix X and label vector y
X = df[features]
y = df['label']

# Normalization
# Scale features to range [0, 1]
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Test and Train
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)
```

```python
# Train and evaluate K-Nearest Neighbors model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

Figure 11. Derived Features and KNN training

Figure 10 & Figure 11. In this section, features consisting of acceleration, velocity, jerk (sudden acceleration) and their derivatives are defined. The data is scaled between 0 and 1 and divided into training and test sets. Then, two different machine learning models are applied: K-Nearest Neighbors (KNN) and Random Forest (RF). Both models are trained with the training data and predictions are made on the test data.

4.4 LSTM Model

Figure 12. The process of creating and training the LSTM-based model. First, balanced data sets are created and training-test separation is performed. The model starts with Conv1D and MaxPooling1D layers, followed by bidirectional LSTM layers. Dropout layers are used to prevent overfitting. Classification is performed in the final layer. Early stopping (EarlyStopping) is applied during training to preserve the best weights.

```python
# Balanced sequence generation and data separation
def create_sequences_balanced(data, labels, window_size):
    class_data = defaultdict(list)
    for i in range(len(data) - window_size):
        window = data[i:i+window_size]
        label_seq = labels[i:i+window_size]
        majority_label = np.bincount(label_seq).argmax()
        class_data[majority_label].append((window, majority_label))

    max_len = max(len(v) for v in class_data.values())

    balanced_data = []
    balanced_labels = []
    for label, samples in class_data.items():
        if len(samples) < max_len:
            reps = max_len // len(samples) + 1
            samples = (samples * reps)[:max_len]
        for window, lbl in samples:
            balanced_data.append(window)
            balanced_labels.append(lbl)

    return np.array(balanced_data), np.array(balanced_labels)

window_size = 75
X, y = create_sequences_balanced(df[features].values, df[target].values, window_size)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
```

```python
# Model definition and training
model = Sequential()
model.add(Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.4))
model.add(Bidirectional(LSTM(64, return_sequences=True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(3, activation='softmax'))

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

callbacks = [
    EarlyStopping(
    monitor='val_loss',
    patience=3,
    min_delta=0.015,
    restore_best_weights=True
)

]

history = model.fit(X_train, y_train, epochs=10, batch_size=32,
                    validation_data=(X_test, y_test),
                    callbacks=callbacks, verbose=1)
```

Figure 12. LSTM Model

5 References

Romera, E., Bergasa, L. M., Arroyo, R., & Barea, R. (2016). Need data for driver behavior analysis? Presenting the public UAH-DriveSet. IEEE Intelligent Vehicles Symposium (IV), 1201–1206.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.