National College of Ireland

CONFIGURATION MANUAL FOR IMPROVING HYBRID MODELS FOR FORECASTING DUBLIN BUS DELAY DUE TO WEATHER CONDITIONS

MSc Research Project
MSc Data Analytics

Loheswari Solisamay Alagarsamy Nagarajan
x23266449

School of Computing
National College of Ireland

Supervisor: Singh J

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Loheswari Solaisamy Alagarsamy Nagarajan
Student ID:	X23266449
Programme:	Msc Data Analytics **Year:** 2024-2025
Module:	Msc research Project
Lecturer:	Singh J
Submission Due Date:	11/8/2025
Project Title:	HYBRID MODELS FOR FORECASTING DUBLIN BUS DELAY DUE TO WEATHER CONDITIONS
Word Count:	1310 **Page Count: 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.

Signature: Loheswari Solaisamy Alagarsamy Nagarajan

Date: 11/8/2025

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 FOR IMPROVING HYBRID MODELS FOR FORECASTING DUBLIN BUS DELAY DUE TO WEATHER CONDITIONS

Loheswari Solisamay Alagarsamy Nagarajan
x23266449

1. Introduction

This configuration manual provides step-by-step instructions for setting up the environment and configurations required to replicate and run the research study hybrid models for forecasting Dublin bus delay due to weather conditions. The model used in this research include standalone Prophet, GRU ad Random Forest , along with the proposed hybrid GRU-Prophet model.The research utilizes the real time Dublin bus data combined with weather data to enhance the prediction accuracy. The data was collected via public APIs, stored in postgreSQL, which was automated using dagster orchestration framework for every 15 minutes.This hybrid model is proposed to improve the performance of the model in real time data.

2. System Hardware Requirements

Before starting the setup, the system should meet he following hardware requirements:

Component	Specification
Operating System	Windows 10/11
RAM	Minimum 8GB but 16 GB recommended
GPU	NVIDIA RTX 3050 Ti, 4 GB VRAM (CUDA 12.2)
Storage	Minimum 10 GB
Processor	Intel Core i7-12700H or Intel Core i5 14 cores

Table 1. Hardware Specifications

3. Software Requirements:

Component	**Specification**
Operating System	**Windows 11 Pro, 64-bit**
Programming Language	**Python 3.8 or more**
IDEs	**VS Code (v1.91.0) for API data extraction and automation, Jupyter Notebook (Anaconda v6.5.4) for model development and evaluation**
Virtual Environment	**dagster_env (conda)**
Workflow Orchestration	**Dagster (v1.6.6) for automated data fetching and storage in PostgreSQL**
Database	**PostgreSQL**

Table 2. Software Requirements

Required Python Packages:

Library / Package	Version	Description
numpy	1.26.4	Efficient numerical computations, array handling, and mathematical functions.
pandas	2.2.1	Data manipulation, cleaning, and handling of time-series datasets.
scikit-learn	1.4.2	Machine learning models, preprocessing, and evaluation metrics.
matplotlib	3.8.3	Visualization library for creating plots, charts, and graphs.
seaborn	0.13.2	Statistical data visualization with enhanced styling for plots.
prophet	1.1.5	Time series forecasting with trend and seasonality decomposition.
tensorflow	2.15.0	Deep learning framework for building and training neural networks.
keras	2.15.0	High-level API for building deep learning models on top of TensorFlow.
dagster	1.6.10	Data pipeline orchestration and workflow management.
SQLAlchemy	2.0.29	Database ORM for structured database connections and queries.
psycopg2-binary	2.9.9	PostgreSQL database adapter for Python.
requests	2.31.0	HTTP library for making API calls and retrieving data from web sources.
statsmodels	0.14.1	Statistical modeling and implementation of ARIMA and other econometric models.

Table 3. Required Python Packages

4. Dataset Details

4.1 Dataset Source and Access

Dataset	Source	Link	Preprocessing
Dublin Bus Data	NTA GTFS Realtime API	https://developer.nationaltransport.ie/apis	Extracted trip updates & vehicle positions, merged with weather data
Weather Data	OpenWeatherMap API	https://openweathermap.org/api	Selected variables: temperature, humidity, wind speed, precipitation

Table 4. Dataset Sources and Preprocessing

The dagster pipeline integreates real time trip updates, vechile locations and weather data, for every 15 minutes. Then the data is stored in PosterSQL for data analysis and modeling .After the data collection , the dataset was downloaded as transport csv file. The data preprocessing , modeling and evaluation was done in Jupyter Notebook.

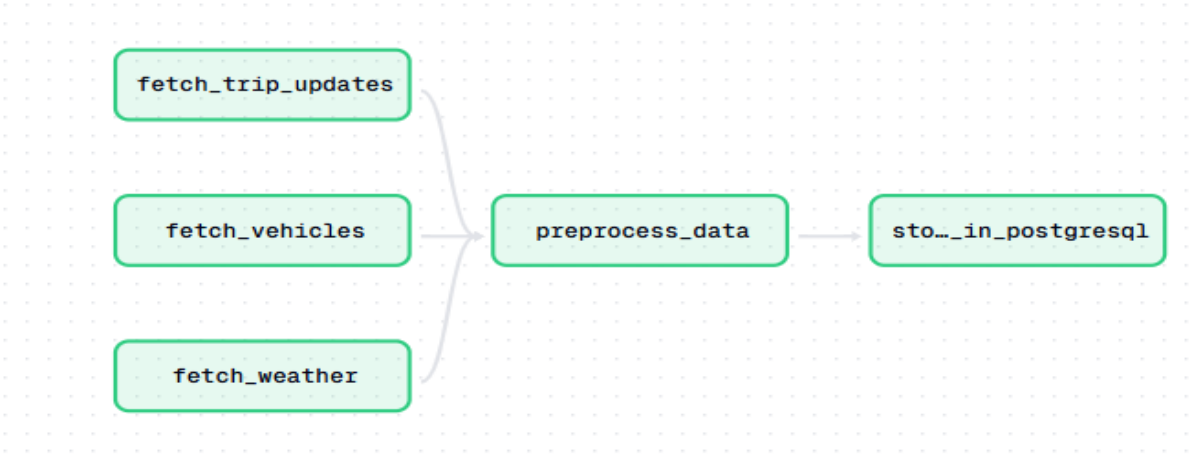

Figure 1. Dagster Orchestration Workflow for Data Collection

4.2 Dataset Features

The dataset includes the following key variables:

	trip_id	start_time	route_id	stop_sequence	arrival_delay	departure_delay	stop_id	latitude	longitude	vehicle_id	...	pressure	weather_main
0	4650_7559	2025-05-26 07:52:00	4630_96257	69	108	108	8240DB007161	53.3913	-6.434465	987	...	1011	Rain
1	4650_7559	2025-05-26 07:52:00	4630_96257	53	337	337	8240DB001899	53.3913	-6.434465	987	...	1011	Rain
2	4650_7559	2025-05-26 07:52:00	4630_96257	52	271	271	8240DB001879	53.3913	-6.434465	987	...	1011	Rain
3	4650_7559	2025-05-26 07:52:00	4630_96257	51	298	298	8240DB001878	53.3913	-6.434465	987	...	1011	Rain
4	4650_7559	2025-05-26 07:52:00	4630_96257	57	302	302	8240DB001890	53.3913	-6.434465	987	...	1011	Rain

]:

stop_id	latitude	longitude	vehicle_id	...	pressure	weather_main	weather_description	wind_deg	cloudiness	visibility	rain_1h	sunrise	sunset	timestamp
B007161	53.3913	-6.434465	987	...	1011	Rain	light rain	230	100	10000	0.99	04:09:52	20:34:43	2025-05-26 08:32:31
B001899	53.3913	-6.434465	987	...	1011	Rain	light rain	230	100	10000	0.99	04:09:52	20:34:43	2025-05-26 08:32:31
B001879	53.3913	-6.434465	987	...	1011	Rain	light rain	230	100	10000	0.99	04:09:52	20:34:43	2025-05-26 08:32:31
B001878	53.3913	-6.434465	987	...	1011	Rain	light rain	230	100	10000	0.99	04:09:52	20:34:43	2025-05-26 08:32:31
B001890	53.3913	-6.434465	987	...	1011	Rain	light rain	230	100	10000	0.99	04:09:52	20:34:43	2025-05-26 08:32:31

Figure 2: Feature of the collected dataset

Feature	Type	Description
trip_id	Categorical	Unique identifier for each bus trip
start_time	Timestamp	Scheduled start time of the trip
route_id	Categorical	Bus route identifier

stop_sequence	Numerical	Order of the stop in the route
arrival_delay	Numerical	Arrival delay in seconds
departure_delay	Numerical	Departure delay in seconds
stop_id	Categorical	Unique identifier of the stop
GPS coordinates(latitude,longitude)	Numerical	Geographical position of the bus
timestamp	Timestamp	Time when the vehicle location was recorded
vehicle_id	Categorical	Unique ID of the bus vehicle
hour_of_day	Numerical	Hour (0–23) extracted from start time
day_of_week	Numerical	Day of the week (0 = Monday, 6 = Sunday)
time_of_day	Categorical	Time bucket: Morning, Afternoon, Evening, Night
is_weekend	Boolean	1 if Saturday or Sunday; 0 otherwise
temperature	Numerical	Measured and perceived temperature in °C
humidity	Numerical	Relative humidity (%)
pressure	Numerical	Atmospheric pressure (hPa)
Wind (speed, direction)	Numerical	Wind speed and direction
cloudiness	Numerical	Cloud cover percentage
Rainfall details	Numerical	Rainfall amount in the last hour
visibility	Numerical	Visibility in meters
Weather conditions	Categorical	General and Detailed weather condition description (e.g., light rain, drizzle)
Sunrise/sunset	Time	Sunrise and sunset time on the observation day

Table 5. Dataset Features and Descriptions

5. Model Configuration

This section highlights the configuration details for all the models used in the research.

5.1. Standalone Models

Random Forest(GridSearchCV Optimized):

Best parameter:

- max_depth: 15
- min_samples_leaf: 1
- min_samples_split: 5
- n_estimators: 100

```python
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df[features])
y = df[target].values
timestamps = df['timestamp'].values

X_train_rd, X_test_rd, y_train_rd, y_test_rd, time_train, time_test = train_test_split(
    X_scaled, y, timestamps, test_size=0.2, shuffle=False
)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_rd, y_train_rd)

best_model = grid_search.best_estimator_
y_pred_rd = best_model.predict(X_test_rd)

mae = mean_absolute_error(y_test_rd, y_pred_rd)
rmse = np.sqrt(mean_squared_error(y_test_rd, y_pred_rd))
r2 = r2_score(y_test_rd, y_pred_rd)

print("Random Forest (GridSearchCV Optimized):")
print(f"Best Parameters: {grid_search.best_params_}")
print(f"MAE: {mae:.2f} seconds")
print(f"RMSE: {rmse:.2f} seconds")
print(f"R² Score: {r2:.4f}")
```

```
Random Forest (GridSearchCV Optimized):
Best Parameters: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
MAE: 0.20 seconds
RMSE: 1.86 seconds
R² Score: 0.9963
```

Figure 3. Random forest Model Architecture for Bus Delay Forecasting

GRU MODEL:

Input shape: Sequence Length=36, Features = all feature engineered and encoded pre)

ARCHITECTURE:

- **Bidirectional GRU layer**: 128 units
- **Dropout:**0.3
- **GRU layer:** 64 units
- **Dropout:** 0.3
- **Dense layer:** 1 neuron

Optimizer: Adam(Learning rate=0.001)

```python
# Scale features
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(df[features])
scaled_target = scaler.fit_transform(df[[target]])

# Sequence function
def create_sequences(X, y, seq_length=36):
    Xs, ys = [], []
    for i in range(seq_length, len(X)):
        Xs.append(X[i-seq_length:i])
        ys.append(y[i])
    return np.array(Xs), np.array(ys)

# Create sequences
X, y = create_sequences(scaled_features, scaled_target, seq_length=36)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, shuffle=False)

# Build model
model = Sequential([
    Bidirectional(GRU(128, return_sequences=True), input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.3),
    GRU(64),
    Dropout(0.3),
    Dense(1)
])
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse')

# Train model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=20, batch_size=32, verbose=1)

# Predict
y_pred_scaled = model.predict(X_val)
y_pred_GRU = scaler.inverse_transform(y_pred_scaled)
y_true_GRU = scaler.inverse_transform(y_val)

# Evaluate
mae = mean_absolute_error(y_true_GRU, y_pred_GRU)
rmse = np.sqrt(mean_squared_error(y_true_GRU, y_pred_GRU))
r2 = r2_score(y_true_GRU, y_pred_GRU)

print(f"MAE: {mae:.2f} seconds")
print(f"RMSE: {rmse:.2f} seconds")
print(f"R² Score: {r2:.4f}")
```

Figure 4. GRU Model Architecture for Bus Delay Forecasting

Prophet (with Selected Regressors):

- **Seasonality:** Daily = True, Weekly = False
- **Regressors:** temperature, precipitation, arrival_delay_lag1, hour_cos
- **Feature Engineering:** Lag delay, 3-hour rolling mean, cyclical hour encoding, peak hour flag
- **Train-Test Split:** 80% training and 20% testing with 48-step gap to avoid leakage

```
'hour_cos', 'is_peak_hour']].rename(columns={'timestamp': 'ds', 'arrival_delay': 'y'})

# Step 4: Train-Test Split
buffer_size = 48  # 1 or 2 days gap
cutoff = int(len(df_prophet) * 0.8)
train_df = df_prophet.iloc[:cutoff]
test_df = df_prophet.iloc[cutoff + buffer_size:]  # Leave a gap

# Step 5: Prophet Model
model = Prophet(daily_seasonality=True, weekly_seasonality=False)
selected_regressors = ['temperature', 'precipitation', 'arrival_delay_lag1', 'hour_cos']
for reg in selected_regressors:
    model.add_regressor(reg)

model.fit(train_df)

future = test_df[['ds'] + selected_regressors]
forecast = model.predict(future)

# Step 7: Evaluation
y_true_ph = test_df['y'].values
y_pred_ph = forecast['yhat'].values

mae = mean_absolute_error(y_true_ph, y_pred_ph)
rmse = np.sqrt(mean_squared_error(y_true_ph, y_pred_ph))
r2 = r2_score(y_true_Ph, y_pred_ph)

# Step 8: Print Metrics
print("Prophet with Selected Regressors and Gap Validation:")
print("MAE: ", round(mae, 2), "seconds")
print("RMSE:", round(rmse, 2), "seconds")
print("R² Score:", round(r2, 4))
```

```
10:56:52 - cmdstanpy - INFO - Chain [1] start processing
10:56:53 - cmdstanpy - INFO - Chain [1] done processing
```

```
Prophet with Selected Regressors and Gap Validation:
MAE:  17.23 seconds
RMSE: 23.25 seconds
R² Score: 0.4308
```

Figure 5. Prophet Model Trend Decomposition (Daily Seasonality)

5.2 Hybrid GRU + Prophet Model:

Prophet Component:

- **Purpose:** Extract long-term smoothed trend from bus delay data.
- **Seasonality:** Auto-detected (daily + weekly if present)
- **Output:** prophet_trend feature added to GRU inputs.

```python
# Drop missing values
df.dropna(inplace=True)

# Selected features
features = [
    'prophet_trend', 'arrival_delay_lag1', 'rolling_mean_3hr',
    'hour_sin', 'hour_cos',
    'day_of_week', 'is_weekend', 'is_peak_hour', 'is_morning',
    'temperature', 'wind_speed', 'precipitation', 'humidity'
]

target = 'arrival_delay'

# Normalize
scaler_X = MinMaxScaler()
X_scaled = scaler_X.fit_transform(df[features])
scaler_y = MinMaxScaler()
y_scaled = scaler_y.fit_transform(df[[target]])

# Sequence creation
def create_sequences(X, y, seq_length=48):
    Xs, ys = [], []
    for i in range(seq_length, len(X)):
        Xs.append(X[i-seq_length:i])
        ys.append(y[i])
    return np.array(Xs), np.array(ys)

X, y = create_sequences(X_scaled, y_scaled, seq_length=48)

# Train/test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Learning rate schedule
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.001,
    decay_steps=1000,
    decay_rate=0.96,
    staircase=True
)
```

Figure 6. Hybrid GRU–Prophet Framework with Engineered Features

Feature Engineering:

- **Lag Features:** arrival_delay_lag1
- **Rolling Statistics:** 3-hour rolling mean of delays
- **Cyclical Encoding**: hour_sin, hour_cos for time-of-day
- **Flags:** is_peak_hour, is_morning, is_weekend
- **Weather Inputs:** Temperature, wind speed, precipitation, humidity

GRU Component:

Input Shape: (Sequence Length = 48, Features = Engineered + Prophet trend)

Architecture:

- **Bidirectional GRU** (128 units, L2 regularization, return sequences) + Layer Normalization + Dropout (0.3)
- **Bidirectional GRU** (64 units, L2 regularization) + Dropout (0.3)
- **Dense Layer:** 1 neuron (linear activation)

Optimizer: Adam with exponential learning rate decay (0.001 → decays over steps)
Early Stopping: Patience = 5 epochs, restores best weights

```
# GRU Model
model = Sequential([
    Bidirectional(GRU(128, return_sequences=True, kernel_regularizer=tf.keras.regularizers.l2(0.001)), input_shape=(X_train.shape
    LayerNormalization(),
    Dropout(0.3),
    Bidirectional(GRU(64, kernel_regularizer=tf.keras.regularizers.l2(0.001))),
    Dropout(0.3),
    Dense(1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule), loss=tf.keras.losses.Huber())

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
                    validation_data=(X_test, y_test), verbose=1,
                    callbacks=[early_stop])

# Predictions
y_pred_scaled = model.predict(X_test)
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_true = scaler_y.inverse_transform(y_test)

# Evaluation
mae = mean_absolute_error(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
r2 = r2_score(y_true, y_pred)

(mae, rmse, r2)
```

Figure 7. Hybrid GRU–Prophet Framework model building

6. Conclusion

This configuration manual provides a structured guide for the replication Dublin bus delay prediction research. It covers the steps for setting up like how to automate the dagster to collect and store the data in SQL. Download the dataset then do data cleaning, preprocessing , implementing the feature engineering, configuring both the standalone and hybrid models and evaluating their performance.By following the outlined procedures, can produce the results, compare baseline models such as random forest , GRU, Prophet with the hybrid model GRU+Prophet.

References

Python: https://www.python.org
TensorFlow: https://www.tensorflow.org
Keras: https://keras.io
Scikit-learn: https://scikit-learn.org
Pandas: https://pandas.pydata.org/
NumPy: https://numpy.org/
Matplotlib: https://matplotlib.org/
Seaborn: https://seaborn.pydata.org/
Statsmodels: https://www.statsmodels.org/stable/index.html