# Task 1: Pattern Classification

## Task Description

The dataset consists of flight trajectories, where each trajectory can consist of more than 1 pattern.

Map

Description automatically generated

The image shows a trajectory containing more than 1 pattern. The first pattern is from Singapore to Jeddah and the second pattern is using the flight pattern Singapore to Christchurch. In this dataset, a trajectory can contain a maximum of 2 patterns. Each trajectory consists of many points, where a point represents a signal.

The task is to use a model to accurately predict the label for each point in every trajectory.

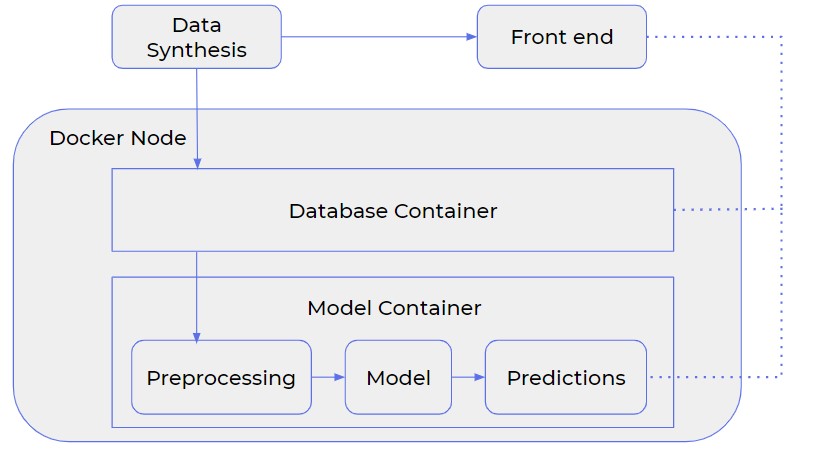
## Model

The model used is an encoder-decoder model.

(describe model architecture)

The model must be able to correctly predict the transition point between points as well as the label for each point.

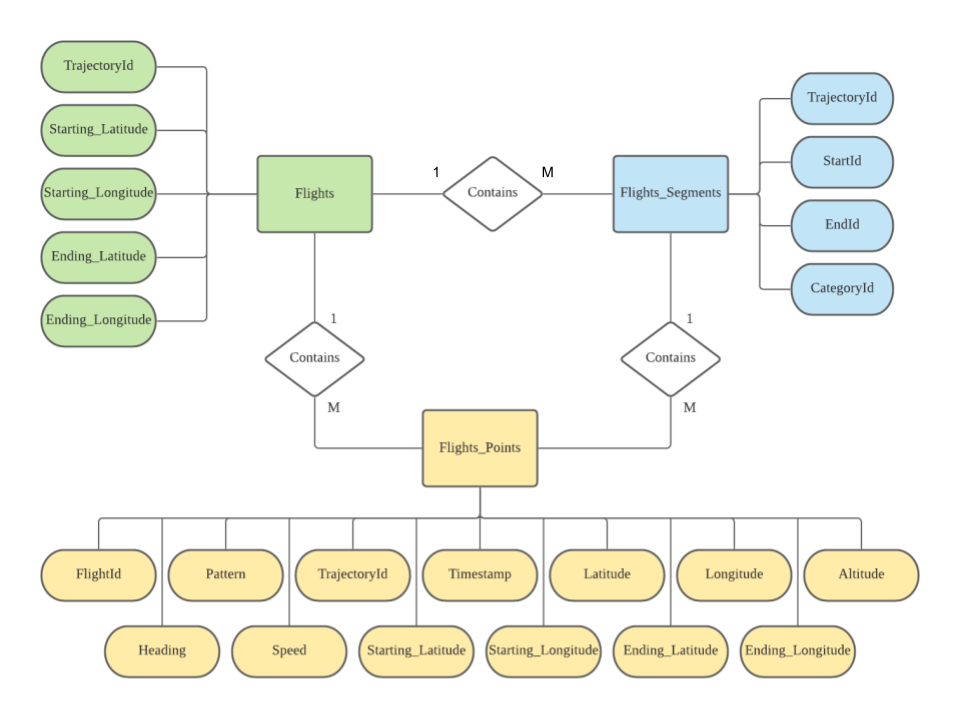
## Workflow



With reference to the image representing the workflow above, docker will be used to contain both the database and model containers. The synthesised data will be passed into the database container. The model will then extract data from the database container, preprocess the data for training, and then feed the predictions into the front end where the predictions can be visualised and for stakeholders to visualise the data as well.

## Input / Output

The inputs will be sql tables. The Entity Relationship (ER) diagram below shows how the sql tables are connected to one another and the tables following it contain a brief description of each column.



|  |  |  |  |
| --- | --- | --- | --- |
| Flights | | | |
| Column name | Type of data | Brief description | Example |
| TrajectoryId | Integer | A unique number given to each trajectory | 0 |
| Starting\_Latitude | Float | Latitude value of the first point of the trajectory | 1.36 |
| Starting\_Longitude | Float | Longitude value of the first point of the trajectory | 172.4 |
| Ending\_Latitude | Float | Latitude value of the last point of the trajectory | 3.54 |
| Ending\_Longitude | Float | Longitude value of the last point of the trajectory | 150.6 |

|  |  |  |  |
| --- | --- | --- | --- |
| Flights\_Segments | | | |
| Column name | Type of data | Brief description | Example |
| TrajectoryId | Integer | A unique number given to each trajectory | 0 |
| StartId | Integer | A unique number given to each point | 1 |
| EndId | Integer | A unique number given to each point | 2 |
| CategoryId | Integer | A unique number given to each segment | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Flights\_Points | | | |
| Column name | Type of data | Brief description | Example |
| FlightId | Integer | A unique number given to each flight | 0 |
| Pattern | String | The name of the class (segment) the point belongs to | 1.36-120.4-CC-1 |
| TrajectoryId | Integer | A unique number given to each trajectory | 2 |
| Timestamp | Float | A float representing the time at which the point is recorded | 106.5 |
| Latitude | Float | Latitude value of the point | 14.6 |
| Longitude | Float | Longitude value of the point | 134.7 |
| Altitude | Float | Altitude value of the point | 187.6 |
| Heading | Float | Heading value of the point | 190.7 |
| Speed | Float | Speed value of the point | 25.6 |
| Starting\_Latitude | Float | Latitude value of the first point of the trajectory the current point is in | 13.8 |
| Starting\_Longitude | Float | Longitude value of the first point of the trajectory the current point is in | 106.4 |
| Ending\_Latitude | Float | Latitude value of the last point of the trajectory the current point is in | 19.5 |
| Ending\_Longitude | Float | Longitude value of the last point of the trajectory the current point is in | 120.7 |

The output will be the updated sql tables with predicted labels and they will be passed into the database container for storage, as well as the front-end for visualisation.

## How to use

Step 1: Load the data (in sql format) in the db folder.

Step 2: Change the name of the sql file in the dockerfile in the db folder.

Step 3: docker-compose up. (sudo docker-compose up)

Step 4: Run the db image. (sudo docker run -d task1\_db)

Step 5: Run the model image.

(sudo docker run --link id\_of\_db\_container\_from\_step\_4:db --runtime nvidia -v absolute\_path\_in\_local\_machine:/src task1\_model). An example of the absolute\_path\_in\_local\_machine is /home/dh/Documents/flights/'task 1'/model/src.

Step 6: Run the annotator image to visualise on the front-end.

(sudo docker run --link id\_of\_db\_container\_from\_step\_4:db -p 8050:8050 task1\_annotator)