INTERNSHIP REPORT

AI BASED HEADWAY OPTIMIZATION FOR METRO LINE 4, Mumbai

A Report Submitted in Partial Fulfillment of the Requirements for The Degree Of

Bachelor Of Technology

By

NAME: Amrit Bhardwaj **Roll no: 23EC008**

to the

School of Technology

GATI SHAKTI VISHWAVIDYALAYA

Central University, Under Ministry of Railways

Vadodara, India-390004

Date: 18/06/2025

SUMMER INTERNSHIP

CERTIFICATE OF COMPLETION

This Is to Certify That the Work Contained in The Project Titled **"Ai Based Headway Optimization for Line 4**" Has Successfully Completed by the Student Under My Supervision During 5 Week Summer Internship. During The Internship Period the Students Found to Be Hard Working, Punctual and Discipline and He Met All the Criteria Satisfactorily.

STUDENT NAME	ROLL NUMBER	SIGNATURE
Amrit Bhardwaj	**23EC008**	

MENTOR DETAILS:

MR RASHMI RANJAN PATTNAIK

SIGNATURE

Acknowledgement

I take this opportunity to express my sincere thanks to my mentor Mr Rashmi Ranjan Patnaik for their guidance and support. I would also like to express my gratitude towards them for showing confidence in me. It was a privilege to have a great experience working under him in a cordial environment.

I am very much thankful to the MMRDA, for providing me the opportunity of pursuing summer internship in Mumbai metro line 4 General consultant I am very much thankful to MMRDA for providing funds to do summer internship.

I am thankful to office staffs and other relevant persons who helped me to figure out and give a friendly environment.

Name

Amrit Bhardwaj

Abstract

This project focuses on headway optimization for Metro Line 4 using artificial intelligence. The core issue which is faced in the metro is cascading delays experienced by the metro trains, which is generally caused by increasing dwell time. Dwell time refers to the amount of time a train is at rest to allow a passenger to get on board and alight. Now, experiencing this dwell time in crowded cities like Mumbai during peak hours, the dwell time increases, and it leads to serial delays arising to the trains moving before it. This is because the current CBTC (Communication-Based Train Control) system, which is being employed in most metro networks—including Line 4—has a feature of using a fixed headway model. Headway is defined as the minimum time interval between two successive trains running on the same track, say upwards or downwards. It is calculated using the formula:

Headway = Run Time + Dwell Time + Crossover Time,
followed by division with the number of trains (rolling stock) available on the line.

The formula is not dependent on any real factor which is varying in operational measures and is revised every year based on the actual headway achieved. Fixed headway limits the responsiveness of the system to this load, especially during peak hours.

This leads to using some method or technology so that we can optimize the headway such that cascading delay could be reduced, and here comes artificial intelligence into the picture.

So, the metro needs a method where they could minimize the headway in such a manner that the safest headway is also not breached, and we could optimize the headway that is calculated from the formulas but also takes into account the new factors that cause headway issues.

To illustrate this, consider a train departing from **Gaimukh** at 8:00 AM and reaching **Gowniwada** at 8:05 AM, with a scheduled headway of 10 minutes. If the train is delayed by 2 minutes at Gowniwada and incurs a further delay of 2 minutes to the next train—originally expected to arrive at 8:15 AM—may now arrive at 8:17 AM. This clearly shows how the **traditional CBTC model stalls under dynamic conditions** and highlights the need for an adaptive AI-based headway optimization system.

To overcome the given problem here we used some of the machine learning models to predict the dwell time at each station during the time of the day so that accordingly we could speed up the cascading trains in such a manner that it doesn't lead braking of the signalling headway that is issued by signalling department but the running headway can be reduced we used open cv models to process the real time data at station and then getting dwell time at different interval of time .after getting the data I the training of those points were done leading to the prediction of dwell time and then accordingly the train operation was done on real time so that the headway may be optimized .the detailed explanation is written in the report below of how the technologies were used and what were the variables that were taken into consideration making this project make eligible for real life application .

Contents

Introduction

This project is undertaken in consultation with Mr. Rashmi Ranjan Patnaik under the General Consultant office, which is responsible for establishing Metro Line 4 and 4A, stretching over 35 km. The station locations are covered from Gaimukh to Bhakti Park, but this project focuses on a specific section between Gaimukh and Cadbury Junction, which spans nearly 10 km.

With increasing urbanization and population density in cities like Mumbai, metro systems face rising demand. Efficient train operations, especially during peak hours, are vital for reducing congestion, improving commuter experience, and maintaining schedule adherence. Delays not only inconvenience passengers but also strain the system's capacity. Hence, intelligent headway management becomes a critical focus area especially for cities like Mumbai .

The Mumbai Metro Line 4 development project is being carried out by multiple companies in partnership with each other under the leadership of MMRDA (Mumbai Metropolitan Region Development Authority). Companies like DB Engineering, Hill International, and Louis Berger are cooperating among themselves for the development of this project.

Metro Line 4 aims to address the major issue of connectivity from the Thane side to Wadala, as currently, there is no efficient transport system operating in this corridor. Many daily commuters reside near Thane and beyond, so connecting to Bhakti Park will benefit these commuters and help share the load on the Mumbai suburban local trains, which currently serve as the primary mode of connectivity between various parts of Mumbai.

- This project focuses on:
- **Prediction of dwell time** at stations and **identification of peak hours** using real-world data.
- Incorporation of **civil speed limits** and other operational constraints into the simulation.
- Adjustment of **follow-up speeds** to enable **headway optimization** between successive trains.
- Monitoring and managing **acceleration profiles** to ensure they stay within acceptable limits for metro systems.
- Visualization of outcomes using a **control chart** to evaluate the impact of headway optimization on service flow and delay patterns.
- To identify the causes and effects of cascading delay
- To analyse the limitation of existing fixed headway CBTC system

Technological Aspect of the Project

Computer Vision

A key factor in this project was the integration of computer vision with transport operations to accurately predict the dwell time at each station. If dwell time increases at a station, it can lead to cascading delays throughout the line. However, if we can trace the predicted

dwell time at each time of day, we can adjust the train's speed accordingly to optimize headway and reduce delays.

To achieve this, the YOLO model (You Only Look Once) was used for person detection in video frames. YOLO is a real-time object detection model that proved to be more effective compared to other models I tried—such as some internal OpenCV models. While the OpenCV models predicted reasonably, they didn't match the real-world data as accurately as YOLO. The YOLO model was used to extract crowd inflow at different times during the day, which provided useful data for analysing passenger load.

Machine Learning

Machine learning was applied to determine how dwell time changes with crowd inflow. After extracting inflow data from YOLO, I tested different ML models to figure out which one best matched the real data captured from the OpenCV-based crowd detection. This model helped predict dwell time dynamically at each station throughout the day. The final predicted dwell times were then fed into the simulation to optimize headway and maintain smooth train operations.

SIMULATION USING C++

Kinematics equations were used to figure out the movement of the train and the civil speed limit data so that the speed restriction is taken into place and all this is done using the data available like the breaking distance and the acceleration profile that is between the parameters that was provided by the mentor so that we could visualize our result and that can be used for the simulation of the train

This project therefore combines real world operational constraints with advanced technologies like computer vision and machine learning to create a realistic and adaptive metro simulation model. By focusing on dwell time predication and headway optimisation and delay control it offers a practical framework for enhancing the efficiency of urban rail system the following section of this report qdetails the methodology implementation and result derived from the work that was carried out

Methodology

In this section I have described how the technologies were used and what were their outputs

COMPUTER VISION:

The model used in this project is YOLOv5 (You Only Look Once, version 5) for real-time detection of crowd inflow at metro stations during different intervals of the day. Due to the current construction phase of Metro Line 4, actual crowd footage from this corridor was not available. Therefore, video surveillance feed from the operational Metro Line 1 (Ghatkopar to Versova) was used as a proxy dataset to simulate and prototype the computer vision-based crowd detection framework.

The YOLOv5 model was selected after comparative testing with other object detection models (basic OpenCV-based detectors). YOLOv5 outperformed others in terms of detection accuracy, real-time performance, and frame-level consistency, making it suitable for high-frequency detection needed in transport applications.

A screenshot of the detection output is attached below, demonstrating how the YOLOv5 model detects and counts individuals in each frame. This count is then logged with a timestamp and later used as an input feature for machine learning-based dwell time prediction.

YOLO v -5 code used to process the video is attached here

Python code snippet used for yolo v-5 model

```
import torch
import cv2
import csv
from datetime import datetime
import time
import pyautogui

# Load YOLOv5s model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', force_reload=False)
model.classes = [0]  # Only detect persons
model.conf = 0.25i

video_path = r"C:\Users\bhard\Downloads\videoplayback (1).webm"
cap = cv2.VideoCapture(video_path)

csv_file = open('inflow_data.csv', mode='w', newline='')
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['Timestamp', 'Person_Count'])

screen_width, screen_height = pyautogui.size()
window_name = 'YOLOv5 Person Detection'

frame_skip = 10
frame_count = 0
total_persons = 0

while cap.isOpened():
    ret, frame = cap.read()
```

```
    if not ret:
        break

    frame_count += 1
    if frame_count % frame_skip != 0:
        cv2.imshow(window_name, frame)
        if frame_count == 1:
            win_x = (screen_width - frame.shape[1]) // 2
            win_y = (screen_height - frame.shape[0]) // 2
            cv2.moveWindow(window_name, win_x, win_y)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        continue

    frame = cv2.resize(frame, (640, 360))

    results = model(frame)
    detections = results.pandas().xyxy[0]
    person_detections = detections[detections['name'] == 'person']
    person_count = len(person_detections)
    total_persons += person_count

    # Draw green circles around detected persons
    for _, row in person_detections.iterrows():
        x1, y1, x2, y2 = int(row['xmin']), int(row['ymin']), int(row['xmax']), int(row['ymax'])
        center_x = (x1 + x2) // 2
        center_y = (y1 + y2) // 2
        radius = int(0.5 * max(x2 - x1, y2 - y1))
        cv2.circle(frame, (center_x, center_y), radius, (0, 255, 0), 3)  # Green circle, thickness=3

    timestamp = datetime.now().strftime("%H:%M:%S")
    csv_writer.writerow([timestamp, person_count])
    print(f"{timestamp}: {person_count} persons")

    cv2.putText(frame, f'Person Count: {person_count}', (20, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow(window_name, frame)
    if frame_count == 1:
        win_x = (screen_width - frame.shape[1]) // 2
        win_y = (screen_height - frame.shape[0]) // 2
        cv2.moveWindow(window_name, win_x, win_y)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    time.sleep(0.05)  # Slow down video playback

cap.release()
cv2.destroyAllWindows()
csv_file.close()

print(f"Total persons detected (sum across frames): {total_persons}")
```

The detailed work flow of the code is like

- Detection of people in a video
- Count them frame by frame
- Log the count with timestamp into a csv files
- Display detection with green circles

Now the question comes what does this code operate like the basic working principle

The yolo v-5 is a deep learning model for real time object detection

It is based on the CNN convolution neural network and is trained across COCO dataset which has 80 object data classes that includes the person. now if we focus that how does it work it is like each frame of video is processed through convolutional layer, further the yolo v-5 layer divides the frame into a grid e.g.(20*20) each grid cell where each grid cell predicts bounding boxes and probabilities in the class for each grid cell the model outputs box coordinates (x,y,w,h) and an objectness score like is something there and then class score like is it a person or a cat …

Now further filters are applied to it for non-maximum suppression and we get the output as a list of [x1, y1, x2, y2, confidence score, class_name].

Now a major doubt arises is that what is CNN and how does it work so the CNN is like a convolutional layer that applies filter to an input to extract important visual feature like edges corners patterns and shapes like say an edge detection filter and texture detection filter these are just filter that are moves over the image multiplies with each point value and then summed up the normal that we studied digital signal processing .

The whole process is just mentioned here like how a convolution is done

A convolutional layer works by applying a small matrix called a **filter** or **kernel** over portions of the input image. For example, imagine a 5×5 section of an image. A filter, typically 3×3 in size, slides over this image section in steps known as **strides**. At each position, the filter performs an operation where it multiplies its values with the corresponding pixel values in the image patch it overlaps. These multiplied values are then summed to produce a single number. This number represents the presence of a specific feature, like an edge or texture, at that location. As the filter continues to slide across the entire image, it generates a new output called a **feature map**, which is essentially a smaller image highlighting the regions where certain visual features (like edges or patterns) exist.

After this we got a file which was having real crowd data of the station during different interval of time entering the station near the opening gates and accordingly the dwell time was forecasted by seeing the data of crowd. Now we got two three parameters the crowd the entry time and the dwell time actual

The traditional CBTC operates on fixed dwell time.

Now we can understand that is real dwell time exceeds the fixed dwell time the cascading effects come into place.

How does machine learning comes into places??

Machine learning

Now if we get the option to predict the expected dwell time in the system arising at any particular station during any hour, we would be able to optimize our headway accordingly and this let us dive into our project

Code snippet for using multiple techniques to figure out which one gives the correct predicted dwell time

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR

# Generate timestamps (1-min intervals)
times = pd.date_range("05:00", "23:59", freq="1min")

# Simulate passengers (Poisson distribution)
passengers = np.where(
    times.hour.isin([7, 9, 17, 18]),  # Rush hours
    np.random.poisson(50, len(times)),
    np.random.poisson(20, len(times))
)

# Calculate dwell time (with noise)
dwell = 30 + (passengers * 0.5) + np.random.randint(0, 15, len(times))

# Extrac
# t only the time part as string (HH:MM)
time_only = times.strftime('%H:%M')

# Create DataFrame for KASARVADVALI only
df = pd.DataFrame({
    "Time": time_only,
    "Dwell_Time": dwell
})

# Convert time to minutes since midnight for regression
df["Minutes"] = times.hour * 60 + times.minute

# Plot Dwell Time vs Time (scatter)
plt.figure(figsize=(12, 5))
plt.scatter(df["Minutes"], df["Dwell_Time"], label="Dwell Time (points)")
plt.xlabel("Time (minutes since midnight)")
plt.ylabel("Dwell Time (seconds)")
plt.title("Dwell Time at KASARVADVALI Throughout the Day")

# Linear Regression
X = df[["Minutes"]]
y = df["Dwell_Time"]
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
plt.plot(df["Minutes"], y_pred, color="red", label="Linear Regression")
# Linear Regression Equation
```

```
print("Linear Regression: y = {:.2f} * Minutes + {:.2f}".format(model.coef_[0],
model.intercept_))

# Polynomial Regression (degree 5)
poly = PolynomialFeatures(degree=5)
X_poly = poly.fit_transform(X)
poly_model = LinearRegression()
poly_model.fit(X_poly, y)
y_poly_pred = poly_model.predict(X_poly)
plt.plot(df["Minutes"], y_poly_pred, color="green", label="Polynomial Regression
(deg=5)")
print("Polynomial Regression Coefficients (degree=5):")
print(poly_model.coef_)
print("Intercept:", poly_model.intercept_)
# Polynomial Regression Equation
coefs = poly_model.coef_
equation = " + ".join([f"{coefs[i]:.2e}*Minutes^{i}" for i in range(len(coefs))])
print("Polynomial Regression Equation: y =", equation)

# Random Forest Regression
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)
y_rf_pred = rf.predict(X)
plt.plot(df["Minutes"], y_rf_pred, color="orange", label="Random Forest")

# Support Vector Regression
svr = SVR(kernel='rbf')
svr.fit(X, y)
y_svr_pred = svr.predict(X)
plt.plot(df["Minutes"], y_svr_pred, color="purple", label="SVR (RBF kernel)")

# Example predictions
minutes_val = 600  # Example: 10:00 AM (600 minutes since midnight)
print("Random Forest Prediction at 10:00:", rf.predict([[minutes_val]])[0])
print("SVR Prediction at 10:00:", svr.predict([[minutes_val]])[0])

plt.legend()
plt.tight_layout()
plt.show()

# --- Interactive Input for Prediction using Random Forest ---
while True:
    user_input = input("Enter time in HH:MM format (or 'exit' to quit): ").strip()
    if user_input.lower() == 'exit':
        break
    try:
        # Convert HH:MM to minutes since midnight
        hour, minute = map(int, user_input.split(":"))
        if hour < 0 or hour > 23 or minute < 0 or minute > 59:
            raise ValueError
```

```
        minutes_input = hour * 60 + minute

        # Predict dwell time
        predicted_dwell = rf.predict([[minutes_input]])[0]
        print(f"Predicted dwell time at {user_input} is: {predicted_dwell:.2f} seconds")
    except:

        print("Invalid input. Please enter time in HH:MM format.")
```

This is for kasarvadavli station during overall days which is used to detect the crowd during the different interval of time to figure out the movement of the crowd

Specially stations like Shreyas cinema responded with different zones in which they are busy as morning time business was less on this station as planned and this is a case where there is R City mall near the station so the peak happens to be during the night and after afternoon so here the dwell time behaviour changes according all the data are generated

So, we got to generate each and every value and finally we achieved to a conclusion that dwell time is dependent on the station locality

The model finally which gave the best score like data results from support vector regression which gives the fitted data

The first type of model which was used here is

1) Linear regression
   So linear regression finds a straight line that figures out the best possible straight line possible to fit in the data provided this is used by finding the correct way or the values that better satisfies the equation

$$Y=MX+C$$

And the minimization of squared error is done by error=∑ (yactual-ypredicted) ^2 to be much less and then its value is minimized as much as possible to decrease the value for finding the correct y;

2) Polynomial regression
   This model works on the same principle as that of the linear only the change that happens is that a linear equation gets varied to a polynomial with varying degree so that we could identify the possible the equation using this

$$y = a * x^n + b * x^{n-1} + c * x^{n-2} + \ldots\ldots$$

And accordingly, the coefficients are identified and the data is set accordingly to find it on different x and coefficients we get a y and then accordingly the mean squared error is calculated this is done until we get a value which suggest us the exact nature these max points are predicted

3) **Support Vector Regression (SVR)** is a regression technique based on the principles of Support Vector Machines (SVM). Instead of trying to minimize the overall error like traditional regression methods, SVR tries to fit a curve such that most of the

predicted values lie within a certain margin of tolerance (called epsilon) around the actual data. Points that fall outside this margin are known as support vectors and play a crucial role in shaping the final regression function. SVR can model non-linear relationships effectively by using kernel functions (like the radial basis function or RBF kernel), which transform the input space into a higher-dimensional space where a linear separation is easier. This makes SVR particularly useful for capturing complex, smooth trends in data while maintaining a good balance between bias and variance.

4) **Random Forest Regression** is an ensemble machine learning method that uses a collection of decision trees to make predictions. Each tree in the forest is trained on a random subset of the data (both in terms of rows and features), which introduces diversity among the trees. During prediction, all trees make their own independent predictions, and the final result is obtained by averaging these outputs. This approach reduces overfitting, increases robustness, and handles both linear and non-linear relationships well. Random Forests are especially good at capturing complex patterns in noisy data and require little parameter tuning, though they are less interpretable compared to simpler models like linear regression.

So we can conclude that in the analysis of dwell time patterns at kasarvadavali station we explored various regression model like the linear ,polynomial ,support vector and random forest while linear regression provided a basic trend it failed to capture non linear variations such as rush hour spikes . Polynomial regression was good to go but as we know it is much sensitive to noise and svr was the only one which provided the smooth data just by using the or most valuable datapoints to figure out it . thus we used that predictor finally in our code to figure out the predicted dwell time

Simulation In C++

Here we have focused on the development of the correct logic for monitoring the correct headway optimization of the speed now the overall scenarios comes what were the factors

```
#include<iostream>
#include<vector>
#include<string>
#include<functional>
#include<cmath>
#include<algorithm>
#include<fstream>
#include <iomanip>

using namespace std;
struct station {
    string station_name;
    string line_name;
    string station_type;
    double distance;
    int run_time;
      int dwell_time;
     int civil_speed;
```

```
};

struct TimetableEntry {
    string train_id;
    string arrival_time;
    string departure_time;
    string station;
    string direction;
    double top_speed_that_canbe_achieved;
};
const vector<string> START_STATIONS = {"BHAKTI PARK METRO",
"GANDHINAGAR"};
const int TRAINS_PER_STATION = 2;
const int HEADWAY = 10*60; // 10 minutes
const int TERMINAL_TURNAROUND =6*60; // 15 minutes
//const int INTERMEDIATE_TURNAROUND = 1; // 1 minute
vector<station> stations = {
    {"GAIMUKH", "LINE4", "METRO_6CAR", 0.0, 180, 180, 35},
    {"GOWNIWADA", "LINE4", "METRO_6CAR", 1502.229, 180, 30, 45},
    {"KASARVADVALI", "LINE4", "METRO_6CAR", 1385.394, 180, 30, 45},
    {"VIJAYGARDEN", "LINE4", "METRO_6CAR", 1024.036, 180, 30, 45},
    {"DONGARI PADA", "LINE4", "METRO_6CAR", 1198.778, 180, 30, 45},
    {"TIKUJI NI WADI", "LINE4", "METRO_6CAR", 1226.694, 180, 30, 45},
    {"MANPADA", "LINE4", "METRO_6CAR", 758.992, 180, 30, 45},
    {"KAPURBAWDI", "LINE4", "METRO_6CAR", 815.824, 180, 30,45},
    {"MAJIWADA", "LINE4", "METRO_6CAR", 1453.707, 180, 30, 45},
    {"CADBUARY JUNCTION", "LINE4", "METRO_6CAR", 824.707, 180, 180, 45}
};

// amde this function for caclulating runtime

double calculate_run_time(double distance, int civil_speed) {
    const double brake_distance = 150.0;
    const double buffer_distance = 50.0; // meters for buffer
    double vmax_kmph = (distance > 800) ? 35.0 : 30.0;
    vmax_kmph = std::min(vmax_kmph, (double)civil_speed);
    double vmax = vmax_kmph * 1000.0 / 3600.0;

    // as mentioned to take the accelerating phase of the train as 1/8th of the distance
    double accelerating_distance = distance / 8.0;

    // 2. Acceleration needed to reach vmax in accelerating_distance: v^2 = 2*a*s => a =
v^2/(2*s)
    double accel = vmax * vmax / (2.0 * accelerating_distance);

    // 3. Time to reach vmax: v = u + at => t = v/a
    double t_accel = vmax / accel;
```

```
    // 4. Deceleration phase: vmax to 0 over brake_distance (assume deceleration = accel for
simplicity)
    double t_decel = sqrt(2 * (brake_distance+buffer_distance) / accel);

    // 5. Cruise distance and time
    double d_cruise = distance - accelerating_distance - brake_distance;
    double t_cruise = (d_cruise > 0) ? d_cruise / vmax : 0;

    // 6. If not enough distance to reach vmax, adjust calculation
    if (d_cruise < 0) {
        // Only accelerate and decelerate, never reach vmax
        double d_half = (distance - brake_distance > 0) ? (distance - brake_distance) :
(distance / 2.0);
        double v_peak = sqrt(2 * accel * d_half);
        t_accel = v_peak / accel;
        t_decel = sqrt(2 * brake_distance / accel);
        t_cruise = 0;
    }

    return t_accel + t_cruise + t_decel;
}
double safestheadwaysbetweeneachstation(double average_speed_in_thatsection, double
distance_between_stations) {
    double safestheadway = 0;
    double buffer_distance = 50.0; // meters for buffer
    double brake_distance = 150.0; // meters for brake distance
    double top_speed = 0; // m/s
    if (distance_between_stations > 800) {
        top_speed = 35.0 * 1000.0 / 3600.0; // m/s
    } else {
        top_speed = 30.0 * 1000.0 / 3600.0; // m/s
    }
    safestheadway = (brake_distance + buffer_distance) / top_speed;
    return safestheadway;
}

double headwayofthissection(double runtime){
    double no_of_trains = 7;
    double headway = runtime / no_of_trains;
    return headway;
}

void generate_timetable() {
    std::ofstream timetable("timetable_5to6.csv");
    timetable << "Station,Arrival,Departure\n";

    // Start at 5:00:00
    int current_time_sec = 5 * 3600; // 5:00 AM in seconds
    std::string arrival, departure;
```

```
// First station: departure only
arrival = "--";
int dwell_time = stations[0].dwell_time;
departure = "05:00:00";
timetable << stations[0].station_name << "," << arrival << "," << departure << "\n";

// For each next station
for (size_t i = 1; i < stations.size(); ++i) {
    double distance = stations[i].distance;
    int civil_speed = std::min(stations[i-1].civil_speed, stations[i].civil_speed);
    double runtime = calculate_run_time(distance, civil_speed);

    // Arrival time at this station
    current_time_sec += static_cast<int>(runtime);
    int arr_h = current_time_sec / 3600;
    int arr_m = (current_time_sec % 3600) / 60;
    int arr_s = current_time_sec % 60;
    std::ostringstream arr_ss;
    arr_ss << std::setw(2) << std::setfill('0') << arr_h << ":"
           << std::setw(2) << std::setfill('0') << arr_m << ":"
           << std::setw(2) << std::setfill('0') << arr_s;
    arrival = arr_ss.str();

    // Dwell time at this station
    dwell_time = stations[i].dwell_time;
    current_time_sec += dwell_time;

    // Departure time from this station
    int dep_h = current_time_sec / 3600;
    int dep_m = (current_time_sec % 3600) / 60;
    int dep_s = current_time_sec % 60;
    std::ostringstream dep_ss;
    dep_ss << std::setw(2) << std::setfill('0') << dep_h << ":"
           << std::setw(2) << std::setfill('0') << dep_m << ":"
           << std::setw(2) << std::setfill('0') << dep_s;
    departure = dep_ss.str();

    // Only write if arrival is before 6:00 AM
    if (current_time_sec <= 6 * 3600) {
        timetable << stations[i].station_name << "," << arrival << "," << departure << "\n";
    } else {
        // If arrival is before 6:00 but departure is after, still write
        if ((current_time_sec - dwell_time) < 6 * 3600) {
            timetable << stations[i].station_name << "," << arrival << "," << "--\n";
        }
        break;
    }
}
timetable.close();
std::cout << "Timetable written to timetable_5to6.csv\n";
```

```
}

int main() {
    double total_runtime = 0.0;
    double total_distance = 0.0;
    double total_dwell_time = 0.0;

    std::ofstream file("section_3.csv");
    file <<
"From,To,Distance(m),RunTime(s),DwellTime(s),TotalSectionTime(s),AverageSpeed(km/
h),SafeHeadway(s)\n";
    for (size_t i = 0; i < stations.size() - 1; ++i) {
        double distance = stations[i+1].distance;
        int civil_speed = std::min(stations[i].civil_speed, stations[i+1].civil_speed);
        double runtime = calculate_run_time(distance, civil_speed);

        // Use terminal dwell for first row, else arrival station dwell
        int dwell_time = (i == 0) ? stations[0].dwell_time : stations[i+1].dwell_time;

        double section_time = runtime + dwell_time;
        double avg_speed = (runtime > 0) ? (distance / runtime) * 3.6 : 0; // km/h
        double safe_headway = safestheadwaysbetweeneachstation(avg_speed, distance);

        file << stations[i].station_name << ","
            << stations[i+1].station_name << ","
            << distance << ","
            << runtime << ","
            << dwell_time << ","
            << section_time << ","
            << avg_speed << ","
            << safe_headway << "\n";

        total_runtime += runtime;
        total_distance += distance;
        total_dwell_time += dwell_time;
    }

    // Add dwell at the starting terminal (if not already included)
    total_dwell_time += stations[0].dwell_time;

    double overall_headway_min = ((total_runtime + total_dwell_time)*2) / 7.0 / 60.0;

    file << "#Total running distance:," << total_distance << "\n";
    file << "#Total run time (s):," << total_runtime << "\n";
    file << "#Total dwell time (s):," << total_dwell_time << "\n";
    file << "#Overall headway for 7 trains (min):," << overall_headway_min << "\n";
    file.close();

    std::cout << "Overall headway for 7 trains (min): " << overall_headway_min <<
std::endl;
```

```
    std::cout << "CSV file 'section_3.csv' created. Open it in Excel." << std::endl;

    generate_timetable();


    return 0;
}
```

This C++ code models the movement of a metro train along Mumbai Metro Line 4 by simulating realistic train dynamics, including acceleration, cruising, and deceleration phases between stations. The core of the simulation lies in the calculate_run_time function, which estimates how long a train takes to cover the distance between two stations based on civil speed limits and the physical behavior of the train.

The simulation assumes that the train begins each section from rest, accelerates uniformly over one-eighth of the section's distance to reach a top speed (vmax), which is either 30 or 35 km/h depending on the section length and civil speed. After reaching this speed, if enough distance remains, the train enters a cruising phase, where it maintains this top speed. The cruise distance is calculated as the remaining segment after accounting for acceleration and deceleration zones. The deceleration phase begins approximately 200 meters (braking + buffer) before the next station, ensuring the train slows down smoothly to a stop. If the total section is too short for cruising, the train accelerates and decelerates directly without reaching top speed.

The overall movement is a realistic simulation of metro operations, where each section's dynamics are adapted to civil constraints and physics-based limits, generating a detailed run-time and speed profile.

In this project, several critical factors were taken into account to accurately calculate the train runtime between stations. One of the primary considerations was the average operating speed of the metro, which is maintained at 35 km/h. This speed is not increased unless the headway — the time interval between consecutive trains — is reduced, as doing so ensures both operational efficiency and passenger safety. The simulation also adheres to the standard service hours of metro operations, running from 05:00 in the morning until 23:59 at night, covering the full span of daily operations.

Another important factor was the distance between consecutive stations. If the distance between two incoming stations is less than or equal to 800 meters, the maximum allowable speed is limited to 30 km/h. This is due to the shorter time available for acceleration and braking. However, if the distance exceeds 800 meters, the train is permitted to reach up to 35 km/h, allowing for a longer cruising phase at higher speed. To model realistic acceleration, typical values ranging from 1 m/s^2 to 3 m/s^2 were considered, aligning with common metro train performance characteristics.

All these parameters were applied based on detailed station-level data provided by the mentor. This included information such as station names, distances, and civil speed limits, enabling the calculation of realistic and dynamic run times between stations. These

calculated runtimes form the backbone of the train movement simulation, ensuring it reflects actual operating conditions as closely as possible.

Exploratory Data analysis

All the data used in this project was sourced from the official tender documents published by MMRDA (Mumbai Metropolitan Region Development Authority). These documents provided detailed information about station locations, distances, dwell times, and civil speed limits, which were critical in building a realistic and rule-driven metro simulation model.

3.1 Station-Level Dataset Overview

The selected stations for simulation are listed in the table below. The dataset includes the cumulative distance from the start station (Gaimukh), dwell time in seconds, and the civil speed limit in km/h for each segment.

The station data is shown below in tabular form

STATION	DISTANCE	DWELL TIME	CIVIL SPEED LIMIT
GAIMUKH	0.0	180	35
GOWNIWADA	1502.229	30	45
KASARVADAVLI	1385.394	30	45
VIJAYGARDEN	1024.036	30	45
D0ONGARI PADA	1198.778	30	45
TIKU JI NI WADI	1226.294	30	45
MANPADA	758.992	30	45
KAPURBAWDI	815.824	30	45
MAJIWADA	1453.707	30	45
CADBUARY JN	824.707	180	45

3.2 Speed Analysis and Runtime Estimation

A core aspect of the data analysis was to ensure that the maximum simulated train speed never exceeds the civil speed limits defined for each segment. Based on the inter-station

distances and acceleration profiles (1–3 m/s²), theoretical speeds were calculated and cross-checked with these civil limits. Any attempt to exceed the civil speed was clipped during simulation.

Additional runtime was also calculated using distance, dwell time, acceleration, and deceleration phases to simulate realistic run segments. The final speed profiles and run durations are validated and shown in the outcome section through graphs and tables.

3.3 Passenger Flow Analysis (Factual Check using opencv)

Another layer of EDA involved validating the passenger gate counts using opencv-based computer vision. The camera setup captured people entering and exiting through the station gates. A factual check was required to ensure that a person was not counted multiple times, especially in high-traffic zones.

To tackle this, we tuned the frame rate and motion detection thresholds, attempting to filter duplicate entries due to slow walking or crowd movement. Though the results were not perfect, the frame-wise optimization provided a close approximation. Future work can include more accurate person re-identification or pose estimation for improved accuracy.

3.4 Train Allocation Strategy

Based on the analysis of demand and operational efficiency, a 7+1 train movement strategy was adopted:

- One train was kept in reserve for contingency or maintenance.
- The remaining seven trains were allocated such that:
    - Two trains were stationed at each terminal station (Gaimukh and Cadbury Junction),
    - Three trains were placed at Kapurbawdi, identified as the busiest junction due to its central location and interchange potential.

This allocation ensured that operational headways could be maintained with minimal delay and efficient turnover.

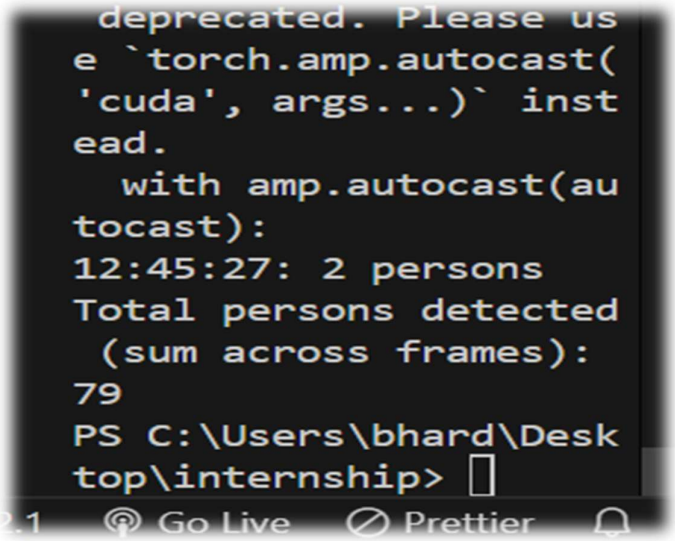
3.5 Turnover and Headway Considerations

Turnover duration — the time a train spends at the terminal before reversing — was fixed at 180 seconds, in line with standard operational practices. This duration was included in all scheduling and simulation logic.

Although no point speed restrictions (temporary or local speed limits) were found in the available data, a note was made that introducing such restrictions in future data collection could improve the realism and accuracy of the model.

OUTCOMES AND RESULTS:

The computer vision results are shown below

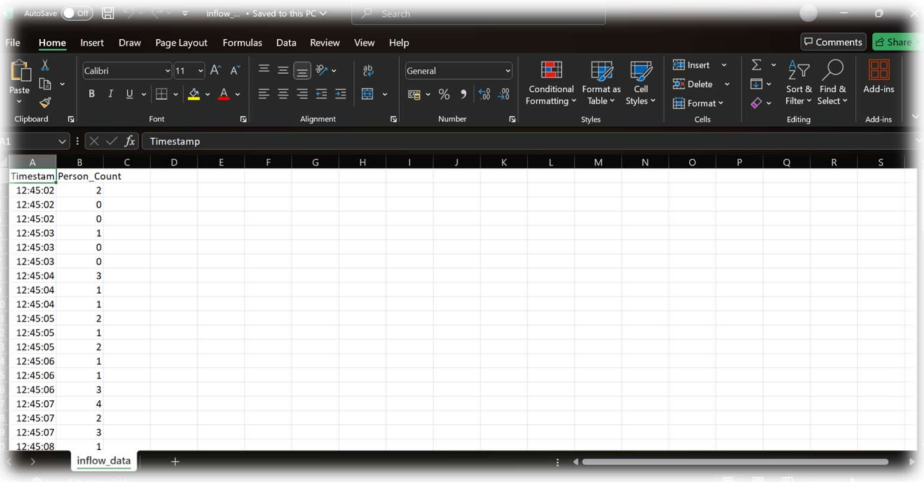


This section presents the final results obtained using our opencv-based person detection system. The objective was to accurately count the number of individuals passing through the metro gate area, which was essential for validating the passenger flow and ensuring the station occupancy remained within permissible limits. For this purpose, we implemented a custom solution using the yolov5 Convolutional Neural Network (CNN) model integrated with opencv.

After comparing several models and methods, yolov5 was selected for its high accuracy and real-time performance, proving to be significantly more reliable than traditional rule-based or built-in human detection methods available in opencv. The model was capable of identifying and tracking individuals across frames, and with appropriate frame-rate tuning

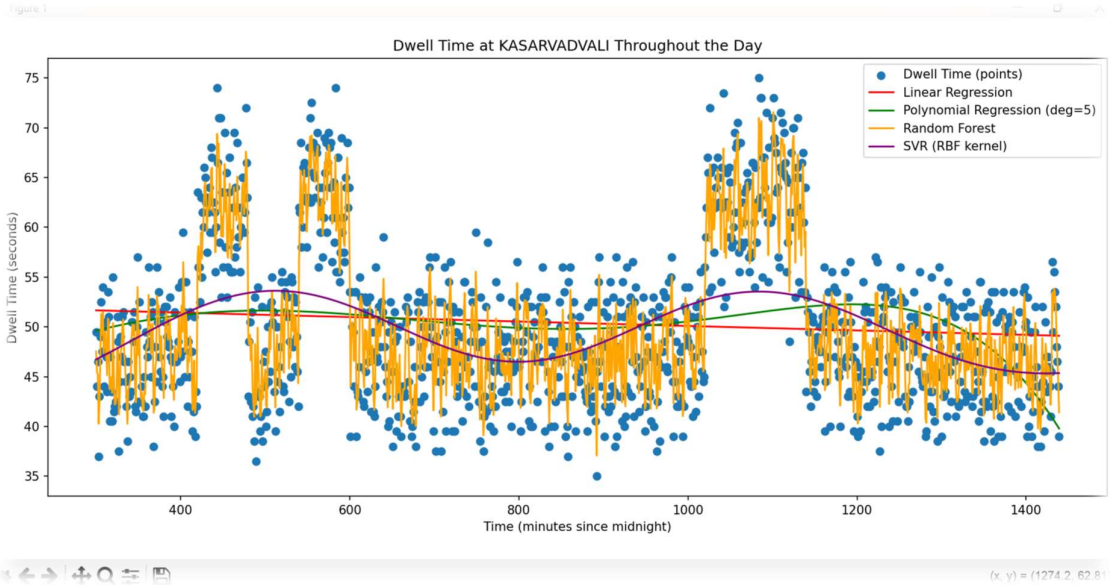
and duplicate filtering, we were able to minimize errors caused by multiple detections of the same person.

The final output displayed the total person count per frame, which was aggregated over time to provide a factual check against the expected station-wise passenger entry. These results were critical in verifying that the entry gates were not overloaded beyond capacity and provided useful feedback for further improving passenger management algorithms in metro station operations.



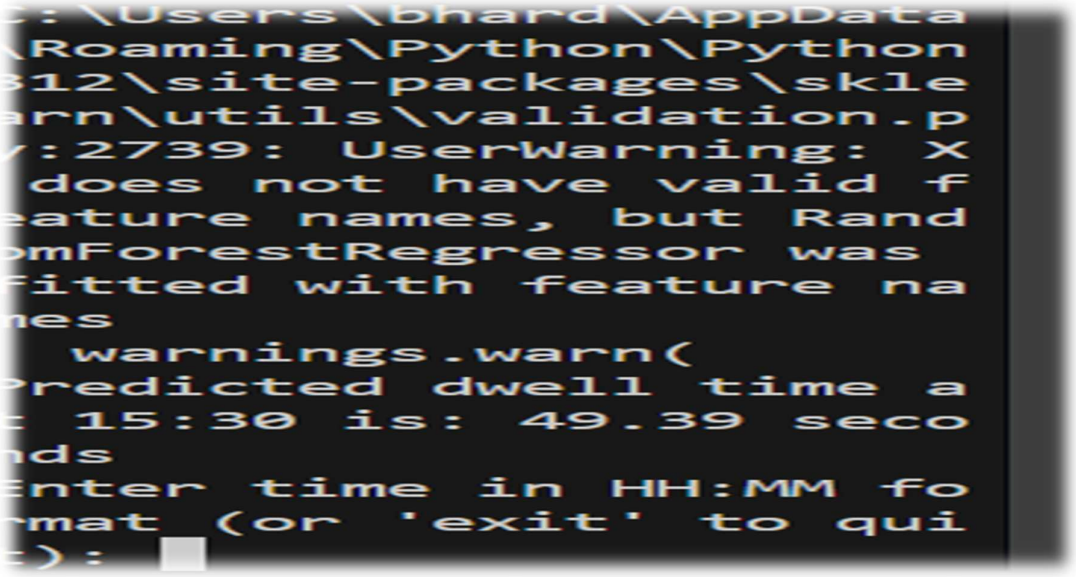
Timestamp	Person_Count
12:45:02	2
12:45:02	0
12:45:02	0
12:45:03	1
12:45:03	0
12:45:03	0
12:45:04	3
12:45:04	1
12:45:04	1
12:45:05	2
12:45:05	1
12:45:05	2
12:45:06	1
12:45:06	1
12:45:06	3
12:45:07	4
12:45:07	2
12:45:07	3
12:45:08	1

This is what we got the inflow of data and different interval of time at different minutes during the day a section of the result is shown here like in which time duration the result was there and then the dwell time was predicted and the results were forecasted finally



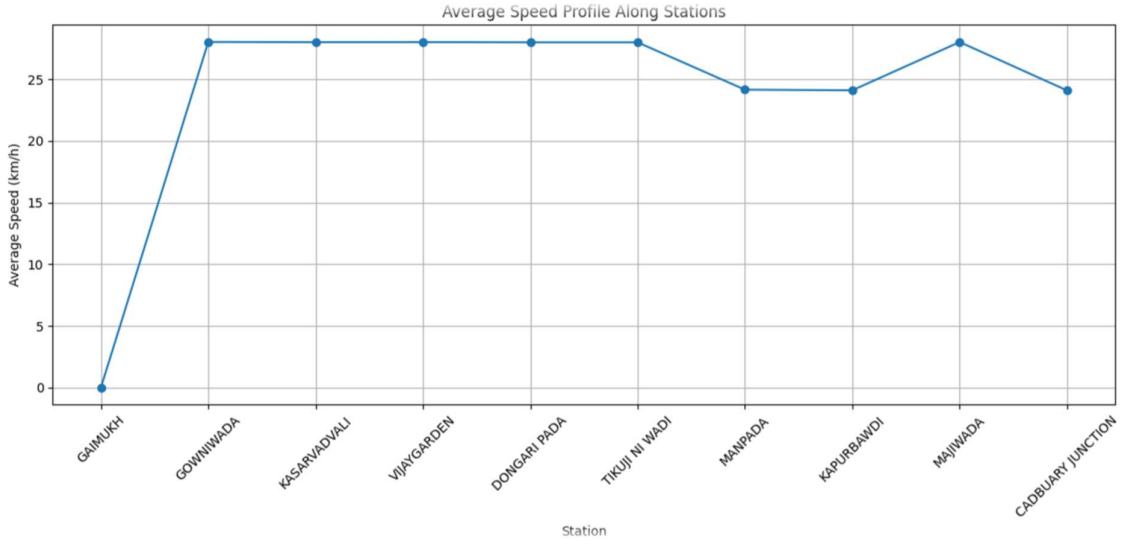
As mentioned the results were finally achieved most proper with the help of SVR MODEL

An example of the dwell time predictor is shown below here



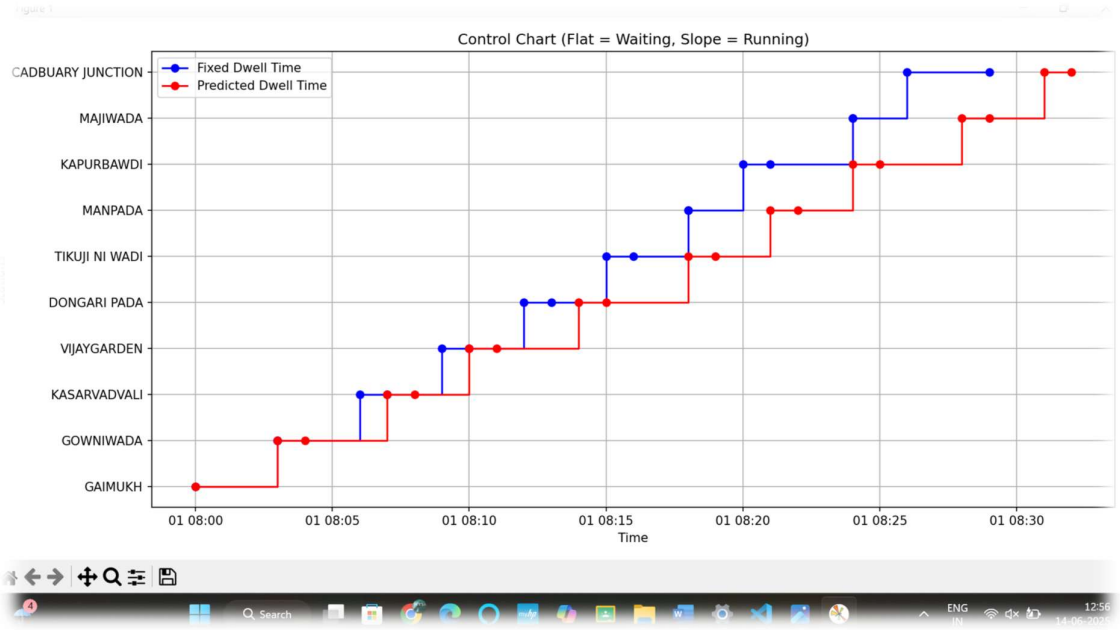
after this the focus was made on the average speed between each section so that our runtime could be calculated accordingly

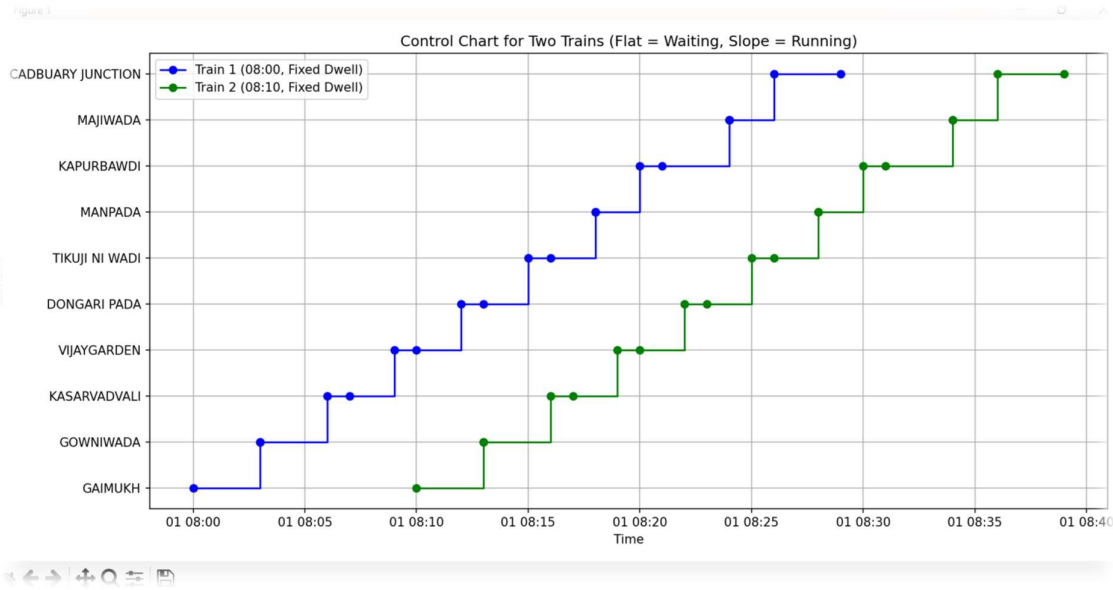
This was the average speed profile that we could achieve in this cadbuary section



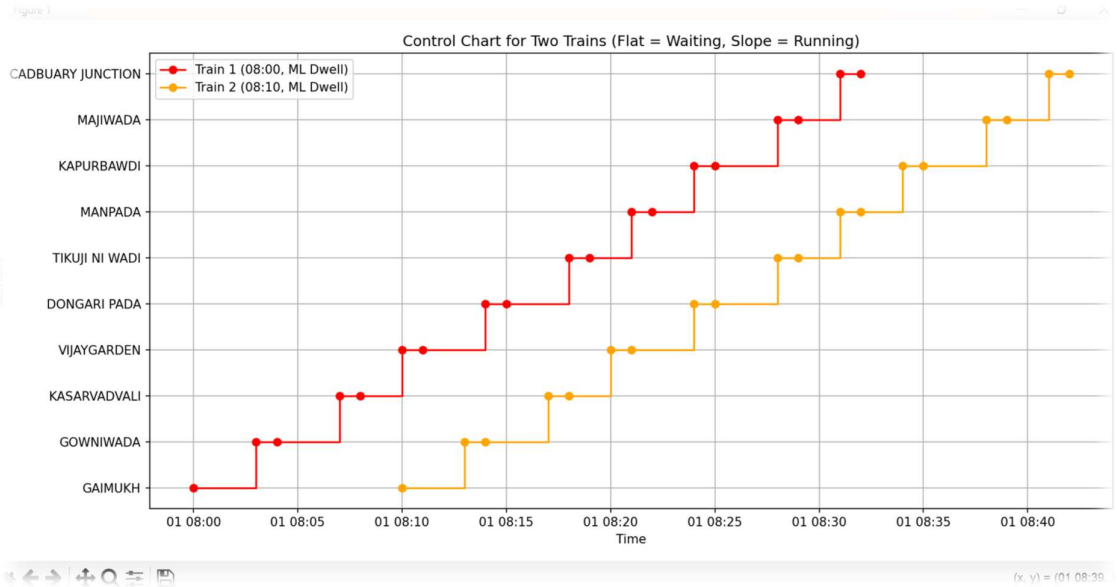
First the focus was made on the control chart preparation predict the train movement logic only using the results of the departure time but the correct results could only be visualized if and only if we focus on the departure time

And this is the one with departure time as well





Here we tried to visualize it the two trains normal running without any delay occurred in it



And this was the output that we got based on our dwell time model and then accordingly we applied the desired model .

The final output on the next page shows us like how the idea that we thought of optimizing the headway could give us the result that the violet line on which my train headway was cut down dynamically in such a way that the train operations were optimized ad this did finally happen our train .

THE OUTPUT :-

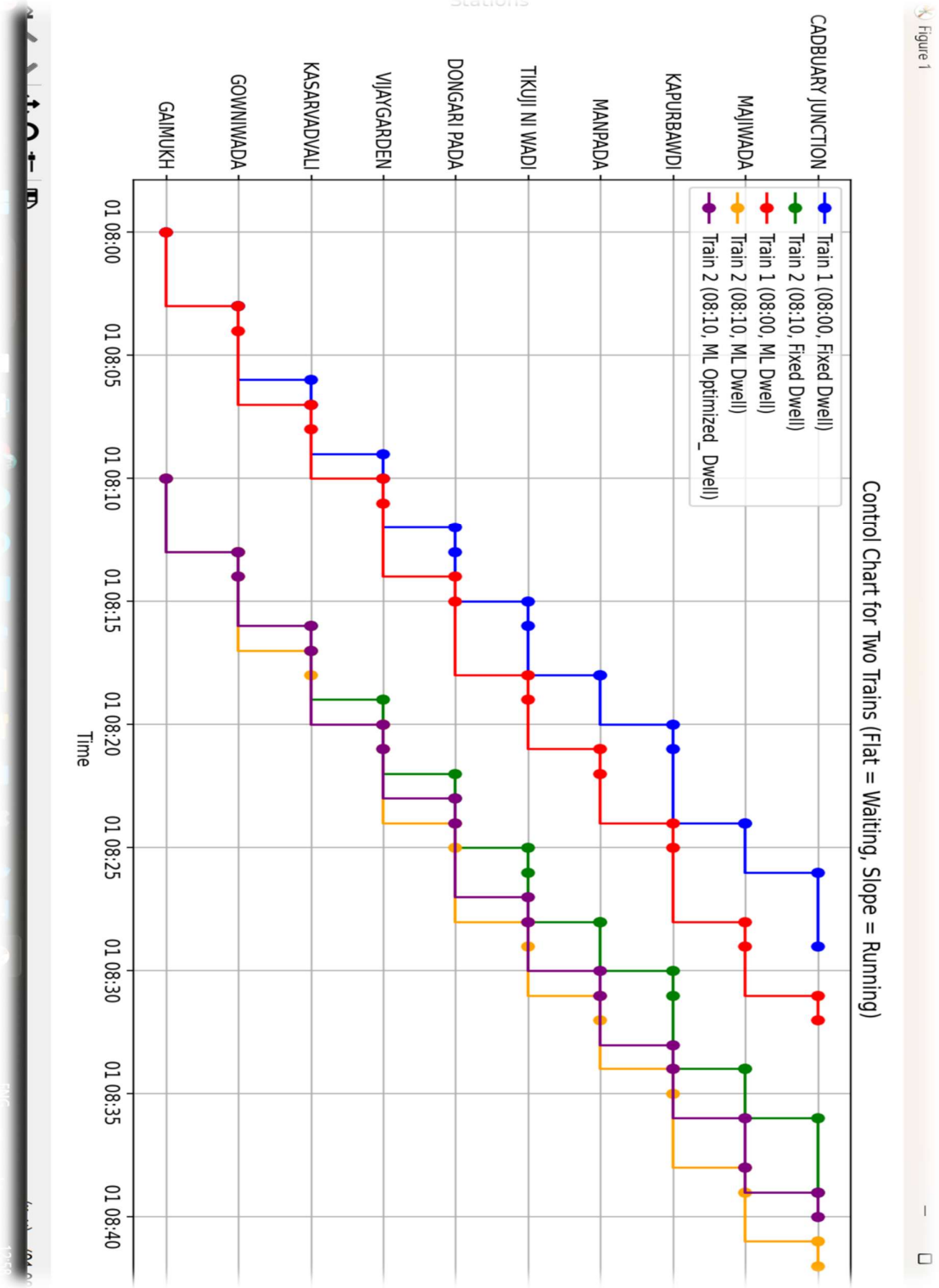


Figure 1
Control Chart for Two Trains (Flat = Waiting, Slope = Running)
Train 1 (08:00, Fixed Dwell)
Train 2 (08:10, Fixed Dwell)
Train 1 (08:00, ML Dwell)
Train 2 (08:10, ML Dwell)
Train 2 (08:10, ML Optimized_Dwell)
CADBUARY JUNCTION
MAJIWADA
KAPURBAWDI
MANPADA
TIKUJI NI WADI
DONGARI PADA
VIJAYGARDEN
KASARVADVALI
GOWNIWADA
GAIMUKH
Stations
01 08:00
01 08:05
01 08:10
01 08:15
01 08:20
01 08:25
01 08:30
01 08:35
01 08:40
Time

Finally understanding what are the results and how is the optimization done here

Like take two examples

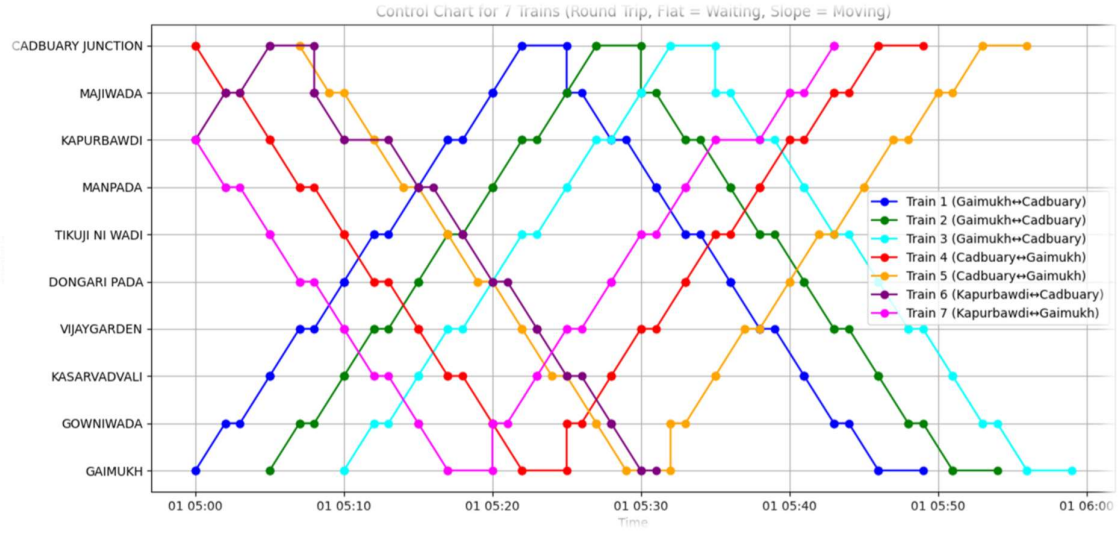
One for station say kapurbawdi

Station	Normal arrival	Ml dwell arrival	Optimized_headway
Kapurbawdi	08:40	08:43	08:42

This is what we were trying to achieve like saving minutes if the train gets delayed due to some unforeseen reason like weather do increase the dwell time and finally the violet line represents the same only for it

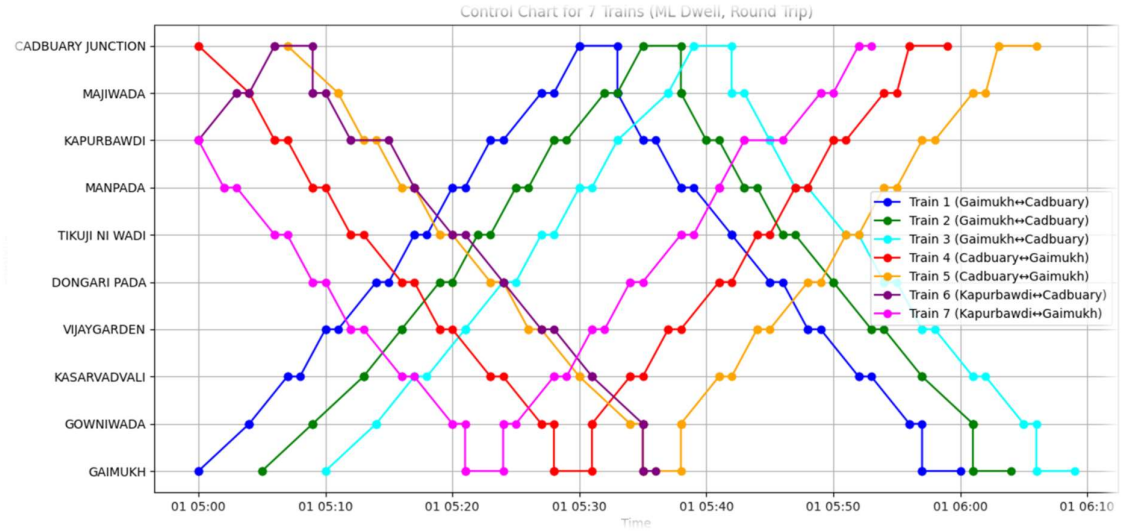
One more example is in the case of last stop where you can see it is the case of the graph how much we could optimize it

Finally the trial was done on scaling it down in such a way that we could visualize it far better

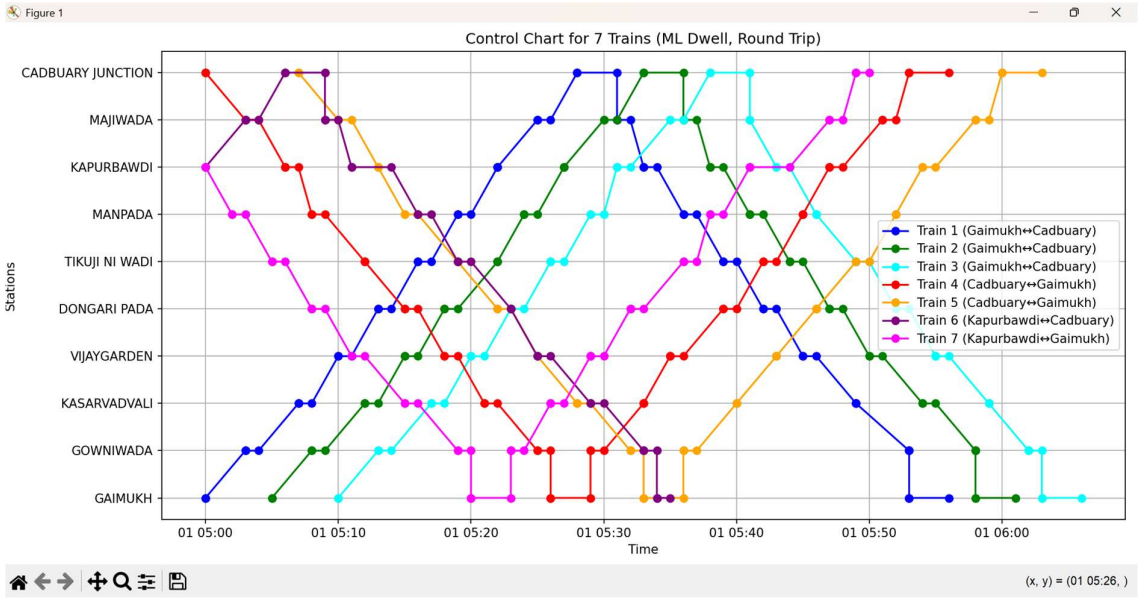


This is the control chart with fixed dwell system and without any predicted dwell

Now we tried like there is a change in fixed dwell time with ml predicted dwell time



Now the output to visualize it using optimized headway for it .



All three graphs help us to understand this easily how headway optimization can help us

Implications and policy recommendations

The outcomes of this simulation and data-driven analysis highlight several important implications for metro system planning, passenger safety, and operational efficiency:

- **Optimized Train Scheduling**:
  With accurate station-wise runtime and dwell time estimation, metro operations can be better scheduled to **minimize headways** and **maximize train frequency** during peak hours without exceeding infrastructure limitations.
- **Speed Profile Enforcement**:
  By ensuring that train speeds do not exceed civil speed limits and adjusting speed dynamically based on inter-station distances, we can maintain **safety compliance** while still ensuring punctuality.
- **Passenger Flow Monitoring**:
  The use of YOLOv5-based real-time passenger counting demonstrates the **feasibility of using computer vision** for crowd monitoring. This system can be scaled for real-time alerts at overcrowded stations or during emergencies.
- **Data-Driven Turnaround Planning**:
  Incorporating a fixed turnaround time (180 seconds) helps avoid cascading delays at terminal stations, ensuring a more **reliable service model** even under high passenger demand.
- **Strategic Train Deployment**:
  The adopted 7+1 deployment logic provides a **balanced train allocation strategy** that reduces idle time while meeting demand at key interchange stations like Kapurbawdi.

2 Policy Recommendations

Based on the findings and system performance, the following policy suggestions are proposed:

1. **Adopt AI-Powered Crowd Management Systems**:
   Implement computer vision models like YOLOv5 at key metro gates to monitor and regulate real-time passenger flow, replacing or supplementing manual counting systems.
2. **Dynamic Speed and Headway Regulation**:
   Allow train speeds to be adjusted in real-time based on actual headway

data and inter-station distances, improving service flexibility and responsiveness.

3. **Infrastructure-Aware Scheduling**:
   Integrate acceleration limits, dwell times, and civil speed constraints directly into the central train scheduling system to reflect real-world dynamics more accurately.

4. **Station-Specific Policy Adjustments**:
   For high-demand stations (e.g., Kapurbawdi), allocate additional rolling stock or implement adaptive dwell times based on real-time passenger volume.

5. **Future Integration of Point Speed Restrictions**:
   Consider incorporating **point-based speed restrictions** into official datasets and simulation logic. This would allow more fine-grained control in areas with curves, gradients, or construction zones.

6. **Centralized Data Access Policy**:
   Encourage open access to tender and infrastructure data (as used in this project) for research and simulation purposes to foster innovation in public transit optimization.

References

Documents provided by the mentor for documents

For generating a data set to train the model and identifying which one of them was best to be used took help from multiple ml models

Some of the class lecture notes of sem 4 were used to check the preparation of control chart as taught in the classes

Some online articles were used for figuring out the delays causes that were happening in the metros.