A Methodology for Implementation of Fuzzy Rule-based and Fuzzy Clustering in Traffic Flow Reduction Estimation

1st Ho-chia Liu
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway
holiu1047@oslomet.no

2nd Alexander Paulsen
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway
s364573@oslomet.no

3rd Sigurd Sandlie
Department of Computer Science
Oslo Metropolitan University
Oslo, Norway
sisan5806@oslomet.no

***Abstract*—With a sharp increase in the speed of urbanization worldwide, there is a need for systematic estimation of traffic congestion; traffic congestion significantly impacts urban productivity, sustainability, and quality of life. In this project, a developed fuzzy rule-based system to output traffic conditions about the data-point Dalkrysset for its potential congestion level is used and examined in the uncertainty inherent in traffic dynamics. Real-world data of traffic checkpoint Dalkrysset, including variables pertaining to weather conditions, and traffic volume by car sizes, and total cars volume are collected and processed using exploratory data analysis and standard data preparation techniques to create a structured dataset for this analysis. Further implementation of the developed fuzzy system is explored in this paper in the second half with unsupervised machine learning techniques fuzzy clustering for realizing automation. The system leverages fuzzy inference methods to evaluate traffic flow and predict congestion with high interpretability, offering a robust tool for improving traffic management and safety. The results demonstrate the feasibility of fuzzy logic for traffic prediction and the finalization of fuzzy clustering paves the way for integrating advanced techniques like machine learning and real-time monitoring.**

***Index Terms*—Keywords: Traffic prediction, fuzzy logic, road capacity, traffic flow reduction, weather impact on traffic, membership functions, fuzzy rule evaluation, defuzzification, traffic data analysis, traffic volume modeling, intelligent transportation systems, machine learning in traffic, congestion prediction, traffic flow simulation, fuzzy clustering, unsupervised learning, machine learning**

I. Introduction

Traffic congestion presents a persistent challenge in urban environments, causing delays, economic losses, and increased emissions. These issues underscore the need for efficient traffic management systems to predict and mitigate congestion. Traditional traffic prediction methods, relying on deterministic models, often fail to address the inherent uncertainty and variability in real-world traffic conditions, particularly those influenced by weather, roadwork, and crashes. The main concept behind current traffic analysis are data-driven intelligence transportation systems (ITS) and exists three categories vision-driven ITS, multisource-driven ITS, and learning-driven ITS. [1] According to Zhang et al., video- and multisource-driven ITS obtain issues in image fusion, and mixing information when it comes to a performing model; and an universal issue all ITS face is the generalization of universal application of all models as different locations might require different sources for specific modeling. This project first proposes a fuzzy rule-based approach to output traffic flow reduction. More specific, this project focus on an output in short-term traffic congestion estimation. [2] In this, we combine the geo-location specific variable "weather" that affects traffic in the area around Dal as fuzzy sets input. Fuzzy logic offers a flexible framework for handling imprecise data and modeling complex systems, making it well-suited for traffic analysis. The study focuses on a single data-point north from Gardermoen, Norway, a critical corridor known for its dynamic traffic towards Hamar and Gardermoen airport.

The system integrates diverse data sources, including weather parameters (e.g., precipitation intensity and weather codes), traffic volume categorized by vehicle size. Automated data collection and preprocessing ensure that the dataset is accurate and well-structured for modeling. A fuzzy inference system is then applied to evaluate traffic conditions and provide interpretable predictions, balancing accuracy and usability.

In basic EDA, traffic volume was found to vary significantly throughout the day, with pronounced peaks during rush hours that contribute to congestion. Weather conditions, particularly precipitation, emerged as critical predictors of traffic flow reductions, demonstrating a strong correlation with adverse weather events. Additionally, vehicle composition and road capacity dynamics were shown to influence overall traffic behavior, with the presence of heavy vehicles and lane configurations playing a significant role in determining congestion levels. These findings validate the choice of variables and highlight the potential of fuzzy logic to model and predict traffic disruptions. The robust preprocessing, feature engineering, and analysis ensure that the dataset is well-prepared for accurate and interpretable predictions, supporting real-world applications in traffic management and planning.

The report outlines the methodology, system design, data

analysis, and fuzzy logic implementation. It concludes with an evaluation of the model's effectiveness and a discussion of future directions, including the potential integration of fuzzy networks or fuzzy clustering for enhanced scalability and real-time traffic monitoring.

II. Methodology

The methodology for this project focuses on developing a fuzzy rule-based system to predict traffic flow disruptions. We incorporate data collection, preprocessing, feature engineering, and fuzzy inference system design to address the inherent uncertainty and complexity of traffic dynamics. Selected variables influencing traffic patterns are altered into measurable inputs for the fuzzy rules incorporation. The core value of this project is to combine multiple fuzzy systems into a integrated fuzzy algorithm system. There are a few obstacles that we faced in the selection of data and was resolved by means of fuzzy system integration. In total, the training process is a repetitive procedure from data preparation, feature engineering, fuzzy rules, and fuzzy systems. Afterwards, the fuzzy systems are combined and output desired variables back into the datasets. Following, the new datasets outputs are then put into a training of fuzzy clustering.

Our layered approach utilised a standard fuzzy sets, rules, and systems knowledge and inference with clustering techniques. Throughout the training, we made sure that the data can capture the uncertainties that occur around Dal Krysset(E6) as closely as possible. The result of robust prediction fuzzy systems in a weather variable and a traffic variable is then tranfsormed into dynamic fuzzy clustering to adapt flexibility and broad interpretability.

A. Data Collection and Data Processing

In this project, the data collection integrates real-world traffic and weather data from public sources. For traffic historical recordings, all is retrieved from Statens Vegvesen (the Norwegian Public Roads Administration) via scraping and stored in csv files. The traffic data is a systematic check-point recording along the EV6 motorway. [3] Traffic data is capturing metrics such as vehicle density, traffic directions, and size distributions. We in total used 1 check-point from the 15 datasets we gathered between Helsfyr and Dal Krysset. For the training of fuzzy systems, only the dataset Dal Krysset was in use. The rest of the 15 check-point datasets is/can be used for fuzzy clustering, testing and training. For weather data, we obtain the dataset via API of Open-Metro service of historical weather API. The original data sources of Open-Metro of location is based on multiple combined weather station observations in which includes European Centre for Medium-Range Weather Forecasts (ECMWF) and Copernicus Regional Reanalysis for Europe (CERRA). [4] [5] Weather data includes key attributes such as precipitation intensity and the weather codes which are crucial for understanding environmental impacts on traffic flow. These datasets are compiled into a unified structure to support further processing and analysis. Each dataset are transformed and altered.

Data preprocessing is conducted to ensure consistency and usability. The traffic dataset is filtered to include relevant time intervals and directions, and missing or invalid entries are addressed through interpolation and imputation. The weather data is organized into hourly summaries to align with the temporal granularity of traffic data. Standardization of timestamps ensures that all data points are synchronized to the same time zone, while derived features such as "Rush hour" and "Is Weekday" are added to capture temporal patterns. These steps result in a clean, well-structured dataset suitable for fuzzy logic modeling. The corresponding datasets contributed to this project in following ways:

The *TrafficDataDownloader.py* script automates the process of downloading traffic data from online sources, enhancing efficiency and consistency in data acquisition for the project. By programmatically accessing traffic data websites and inputting required parameters—such as road IDs, date ranges, and specific locations—it initiates the download of traffic data files without manual intervention. This automation reduces the potential for human error and ensures that the datasets collected are accurate and up-to-date.

The script contributes significantly to the project by providing comprehensive traffic datasets along the EV6 motorway. These datasets record hourly traffic volumes from October 31, 2023, to October 31, 2024, categorized by vehicle sizes ranging from "<5.6m"to ">=24m". Each dataset is split into two directions: towards Oslo and towards Dal. For training the fuzzy logic system, the datasets in the direction towards Oslo are prioritized, as they are most relevant to the project's objectives.

By supplying detailed traffic information—including total vehicle counts and vehicle size distributions—the *TrafficData-Downloader* script enables the project to train the fuzzy system effectively. The ultimate goal is to determine congestion levels at each checkpoint along the EV6 motorway based on car volumes, thereby enhancing traffic management and forecasting.

- The *cleancsv.py* script removes irrelevant columns, filters rows based on specific conditions, and saves the cleaned data into CSV files. The *readcsv.py* script further processes the traffic data by filtering it for specified date ranges and traffic directions (e.g., "Totalt i retning Alnabru" and "Totalt i retning Sentrum"). It groups the data by date and time ranges, summarizing metrics such as total vehicle counts and vehicle size distributions. Each of traffic datasets along the EV6 motorway contains a column of 11 variables contributing to our project excluding date, starting hour and ending hour. The datasets records the passing traffic in car volumes by vehicle sizes from "<5.6m" to ">=24m" by quantity. The recording is by date per each hour from 31-10-2023 to 31-10-2024. Each traffic datasets is split into two way 1. direction towards Oslo and 2. direction towards Dal. In our training for fuzzy system, datasets with direction towards Oslo are prioritized and used the most. The desired setting is to train each datasets along EV6 motorway and outputs each check-point congestion level based on cars volume.

- The *weatherapi.py* script processes raw weather data, converting it into structured summaries. Precipitation levels, visibility, and wind speeds are categorized and aggregated, ensuring they are ready for feature engineering and correlation analyses. Preprocessing also involves ensuring temporal alignment. For instance, timestamps in traffic data are standardized to the Europe/Oslo timezone using Helsfyr_EDA.ipynb, facilitating accurate time-series analyses. Missing values are handled using interpolation or imputation, while invalid entries are identified and corrected. Weather datasets records region of Oslo in which contains a column of 10 variables contributing to our project excluding date and time range. The datasets contains variables in temperature, precipitation, rain, showers, snowfall, and wind speed by different meters. In the formation of fuzzy system for weather data, we desire to output a weather condition fuzzy level using obtained variables.
- The *dataprocesser.py* processes traffic data by cleaning and filtering the datasets. It first removes irrelevant columns and handles missing or invalid entries, converting them to consistent numeric values. It filters the data for specified date ranges and selects rows where the traffic direction starts with "Totalt i retning". The script groups the data by date and time intervals, summarizing total vehicle counts and distributions across various vehicle sizes ranging from "<5.6m" to ">=24m". The traffic datasets record hourly car volumes from October 31, 2023, to October 31, 2024, and are split into two directions: towards Oslo and towards Dal. In training the fuzzy logic system, the datasets in the direction towards Oslo are prioritized to determine congestion levels based on car volumes at each checkpoint along the EV6 motorway.
- The *weatherhistoryapi.py* automates the retrieval of weather data from the Open-Meteo API for a specified latitude, longitude, and date range. It requests hourly weather variables—such as temperature, precipitation, and other atmospheric conditions—and saves the collected data into a CSV file. This ensures that the project has consistent and detailed weather information corresponding to the traffic data's time frames. By providing a comprehensive environmental data, the script contributes significantly to the project's ability to analyze the impact of weather on traffic patterns and congestion levels along the EV6 motorway. Incorporating weather variables into the fuzzy logic system enhances the accuracy of congestion predictions, as it allows the model to account for external factors influencing driving behavior and traffic flow.

These are essential development for our fuzzy clustering and we focus on creating an automated procedure when designing our project. Thus each of the python file can be extracted and used to extract different choice of timeframe, traffic, or weather data for training obtaining dataset.

III. Scripts

TrafficDataDownloader Class: 1. Purpose: Automates the downloading of traffic data from online sources to enhance efficiency and consistency in data acquisition. 2. Contribution to the Project: Streamlines the data collection process by automating the download of traffic datasets, ensuring that the data is accurate, up-to-date, and consistently formatted. This automation is crucial for training and validating the fuzzy logic system in congestion prediction. 3. Functionality:

1) Initialization: Sets up the download directory and initializes the Selenium WebDriver with options for automated downloads.
2) Driver Setup setup_driver: Configures the WebDriver preferences, including download directory, disabling download prompts, and enabling automatic file overwriting.
3) URL Generation (generate _url): Dynamically creates URLs based on parameters like road ID, date range, latitude, and longitude to access specific traffic data.
4) File Downloading (download _file): Automates navigation to the generated URL, waits for the download button to become clickable, initiates the download, and ensures files are saved with meaningful names.
5) Batch Downloading (download _multi _files): Iterates over multiple road IDs to download traffic data files for different locations and date ranges.
6) Download Monitoring (wait _for _download): Monitors the download directory to confirm that files have fully downloaded before proceeding, handling timeouts to prevent indefinite waiting.
7) Cleanup (quit): Closes the WebDriver instance to free up system resources.

Fetch_weather_data python Script Class: 1. Purpose: Retrieves historical weather data from the Open-Meteo API for specified locations and date ranges, saving it for further analysis. 2. Contribution to the Project: Provides accurate and relevant weather information, which is crucial for analyzing environmental impacts on traffic flow. Incorporating weather data enhances the predictive capabilities of the congestion prediction model by accounting for weather-related variabilities. 3. Functionality:

1) API Client Setup: Initializes an API client with caching and retry mechanisms to handle network issues and reduce redundant requests.
2) Parameter Configuration: Sets up the parameters required for the API call, including latitude, longitude, start and end dates, hourly variables to fetch (e.g., temperature, precipitation), and timezone.
3) Data Retrieval: Makes the API request to fetch weather data, handling responses and potential errors.
4) Data Processing: Extracts hourly data from the API response and constructs a pandas DataFrame containing timestamps and corresponding weather variables.
5) Data Saving: Writes the DataFrame to a CSV file, making the weather data readily available for merging

with traffic data.

Process_traffic_data python Function Class: 1. Purpose: Processes raw traffic data files to extract relevant information for analysis. 2. Contribution to the Project: Cleans and structures the traffic data, making it usable for further analysis and modeling. By organizing the data into a consistent format, it facilitates integration with weather data and its use in the congestion prediction model. 3. Functionality:

1) Data Loading: Reads the raw CSV traffic data into a pandas DataFrame.
2) Date Filtering: Converts date strings to datetime objects and filters the DataFrame to include only records within the specified date range.
3) Direction Filtering: Keeps only the rows where the "Felt" (lane or direction) starts with "Totalt i retning," indicating total counts in a particular direction.
4) Grouping and Aggregation: Groups the data by date and time range to aggregate traffic counts and vehicle size distributions.
5) Data Extraction: Extracts total car counts and counts of different vehicle size categories for each group.
6) Data Structuring: Compiles the extracted information into a structured format, creating a new DataFrame suitable for analysis or merging with other datasets.

clean_data python Function Class: 1. Purpose: Cleans and structures the traffic data, making it usable for further analysis and modeling. By organizing the data into a consistent format, it facilitates integration with weather data and its use in the congestion prediction model. 2. Contribution to the Project: Ensures that the traffic datasets are accurate and free of inconsistencies that could affect the model's performance. Clean data is essential for reliable analyses and effective training of the fuzzy logic system. 3. Functionality:

1) Encoding Detection: Determines the encoding of the CSV file to correctly read it, accommodating different file encodings that might be present in datasets.
2) Data Loading: Reads the CSV file using the detected encoding and appropriate delimiter.
3) Column Inspection and Dropping: Identifies and removes columns that are not needed for analysis, reducing data clutter.
4) Data Cleaning: Replaces placeholder values like '-' with zeros to standardize numerical data.
5) Type Conversion: Converts columns to appropriate data types (e.g., numeric types) to facilitate calculations and analyses.
6) Direction Filtering: Filters the data to include only relevant rows, specifically those starting with "Totalt i retning."
7) Data Saving: Saves the cleaned data to a new CSV file with a modified name, indicating that it has been processed.

CongestionPrediction python Script Class: 1. Purpose: Employs fuzzy logic to predict traffic congestion levels based on various input factors, including total car volume, heavy vehicle percentage, rush hour status, and weather conditions. 2. Contribution to the Project: Serves as the core tool for predicting congestion levels on the E6 motorway. By modeling uncertainties and nonlinear relationships inherent in traffic systems using fuzzy logic, it provides a sophisticated method for forecasting congestion. The integration of traffic and weather data allows for more accurate and reliable predictions, essential for traffic management and planning. 3. Functionality:

1) Data Loading and Preprocessing: Loads and prepares merged traffic and weather data for analysis.
2) Fuzzy Variable Definition: Defines fuzzy variables (antecedents and consequent) for inputs and outputs, assigning appropriate ranges and membership functions. Inputs include "Weather Category," "Total Cars," "Rush Hour," and "Heavy Vehicle Percentage"; the output is "Congestion Level."
3) Membership Functions: Uses predefined or automatic membership functions to represent linguistic terms like "Low," "Medium," "High," "Clear," and "Rainy."
4) Rule Generation: Generates fuzzy rules dynamically based on data patterns or a predefined rule set. These rules determine the output congestion level based on combinations of input variable levels.
5) Control System Creation: Constructs a fuzzy control system using the defined variables and rules, and initializes a simulation environment to evaluate the system.
6) Test Case Simulation: Runs multiple test cases by assigning input values, computing the congestion level using the fuzzy inference system, and logging the results.
7) Result Documentation: Writes the outcomes of the test cases to a text file and saves visualizations of the output membership functions, aiding in analysis and reporting.

Script with Fuzzy C-Means Clustering python: 1. Purpose: Enhances the congestion prediction model by incorporating fuzzy c-means clustering to identify patterns in the data, leading to dynamic rule generation. 2. Contribution to the Project: By incorporating clustering, the script allows for data-driven generation of fuzzy rules, which can capture complex patterns and relationships in the data that might not be evident through manual rule creation. This enhances the model's adaptability and potentially improves prediction accuracy, providing deeper insights into the factors affecting congestion. 3. Functionality:

1) Data Loading and Preprocessing: 1. Loading Data: Uses the load data function to read and merge traffic and weather datasets, including mapping weather codes using a provided mapping. 2. Preprocessing Data: Prepares the merged data using the preprocess data function, which may include normalization and feature selection.
2) Fuzzy C-Means Clustering: 1. Applying Clustering: Uses the fuzzy _cmeans _clustering function to apply fuzzy c-means clustering to the preprocessed data, specifying the number of clusters. 2. Obtaining Centroids and Labels: Retrieves the cluster centers (centroids) and labels, which represent patterns in the data.
3) Fuzzy Variable Definition: 1. Defining Antecedents and

Consequent: Defines fuzzy variables for inputs (weather category, total cars, rush hour, heavy vehicle percentage) and the output (congestion level). 3. Automatic Membership Functions: Uses *automf* to automatically generate membership functions with specified names (e.g., 'Low', 'Medium', 'High').

4) Dynamic Rule Generation: 1. Iterating Over Centroids: For each cluster centroid, calculates the degree of membership for each input variable's terms. 2. Determining Levels: Identifies the term (e.g., 'Low', 'High') with the highest degree of membership for each variable. 3. Defining Congestion Level: Similarly determines the congestion level based on the centroid. 4. Creating Rules: Constructs fuzzy rules dynamically using the identified levels for inputs and outputs, and adds them to the rule set.
5) Control System Creation: 1. Building the Control System: Creates a ControlSystem with the dynamically generated rules. 2. Initializing Simulation: Sets up a *ControlSystemSimulation* for running simulations.
6) Test Case Simulation: 1. Running Simulations: Iterates over predefined test cases, sets input values, and computes the predicted congestion level. 2. Mapping Outputs: Maps the predicted congestion level to a congestion status (e.g., 'Free Flow', 'Moderate') based on thresholds. 3. Logging Results: Writes the results to a file and prints them to the console for analysis. 4. Visualizing Outputs: Generates and saves plots of the output membership function for each test case.

Overall Workflow includes 1. Data Collection with using *TrafficDataDownloader* to automate the download of traffic data and using fetch _weather _data python to retrieve weather data corresponding to the traffic data time frames. 2. Data Processing with clean and process the traffic data using clean _data and process_traffic_data functions. Also ensure the data is structured and synchronized for integration. 3. Model Development where we Load and pre-process the merged data then apply fuzzy c-means clustering to identify patterns. Define fuzzy variables and membership functions. Generate fuzzy rules dynamically based on clustering results. Create the fuzzy control system. 4. Model Evaluation: Run simulations using test cases and evaluate and document the model's performance then visualize outputs for analysis. 5. Application: Use the developed model to predict congestion levels under various scenarios and provide insights for traffic management and planning along the E6 motorway.

A. Features

Our project focuses on predicting traffic congestion levels by leveraging both traffic and weather data. The integration of these datasets is crucial for capturing the multifaceted nature of traffic dynamics influenced by environmental conditions.

1) Missing values: using line plots and time-series analyses reveal pronounced peaks during morning and evening rush hours, reflecting commuter patterns. These temporal dependencies underscore the importance of understanding that we are dealing with time-based variables in the fuzzy logic model. One key ingredients of our data both traffic and weather are time-based data. This indicates that any missing or null values needs to be handled with extra care. Initially, the missing values were replaced with mean of total dataset. However, we discover high bias would be introduced into further application. At last, the missing values were replaced with a. k nearest neighbor with 10 closest neighbors or b. mean values by dates of corresponding variables with dataframe groupby function.
2) Traffic: understanding road capacity is essential for modeling traffic flow disruptions. Scripts such as *caclcapacityroads.py* and *calculateroadcapacity.py* calculate road capacity by incorporating several key parameters. Vehicle length and speed are factored into lane capacity calculations, while safety gaps are considered to create realistic traffic scenarios. Additionally, the number of lanes is used to estimate the capacity for each direction of traffic, providing a comprehensive framework for analyzing congestion potential. These calculations provide essential insights into the maximum load the road network can handle under varying conditions, forming a basis for assessing congestion levels. In so, the consideration of road condition may vary, we introduced existing simplified methodology for motorway capacity calculation. The U.S. Department of Transportation provides a general framework for motorway capacity and/or flow speed with number of motorway lanes. [6] In this case, we simplified it into two features: vehicle density per Lane and the proportion of heavy vehicles (% HV) (see Equation 1 and 2), are derived in ensuring the enhancement of the model's ability to capture complex relationships between inputs and possibility of traffic occurrence based on these two significant factors.

$$Density = \frac{Total\,Cars\,Volume}{Number\,of\,Lanes} \tag{1}$$

$$HV\,Percentage = \frac{Large\,Vehicles\,Volume}{Total\,Cars\,Volume} \tag{2}$$

3) Weather: Historical weather data was fetched using the Open-Meteo API. Key weather variables include temperature, precipitation intensity, and weather codes, which categorize weather conditions into 'Clear', 'Cloudy', 'Rainy', and 'Snowy'. The weather code mapping dictionary translates these weather codes into categorical labels, enabling seamless integration with traffic data. Another important feature we included was using the date and time. For time variable: according to geolocation technology specialist TomTom, Oslo estimated traffic congestion occurs during 7 to 9 a.m. and 14 to 17 p.m. [7] The time variable is thus transformed into

$$\begin{cases} 1, & \text{if x = } 7-9\ a.m.\ or\ x = 14-17\ p.m. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

For date variable, it is shows that weekday often bring about higher traffic volume than weekend and is thus taken into account by transforming into binary variables. [8]

$$\begin{cases} 1, & \text{if x = } weekdays \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

B. Fuzzy Rules and Membership Function

As abovementioned, this project is formed by two fuzzy systems: traffic and weather. The first traffic fuzzy system ought to output the desired variable: traffic volume level with [low], [medium], and [high]. The second weather fuzzy system ought to output the desired variable: weather condition level [none], [mild], [cautious], and [severe]. Our standard procedure for defining the fuzzy systems includes defining the membership functions, fuzzy rules, and wrap up with fuzzy inference system (FIS) for each fuzzy systems' formation.

We fuzzified our input variables using membership functions. This is for the purpose of mapping crisp values into categories for later procedure of fuzzy rules. There are in total of two membership function in application for all fuzzy systems in this project via triangular fuzz.trimf and trapezoidal fuzz.trapmf(see equation 5 and 6). This is cater to capture different varaibles' nature in either continuous or binary.

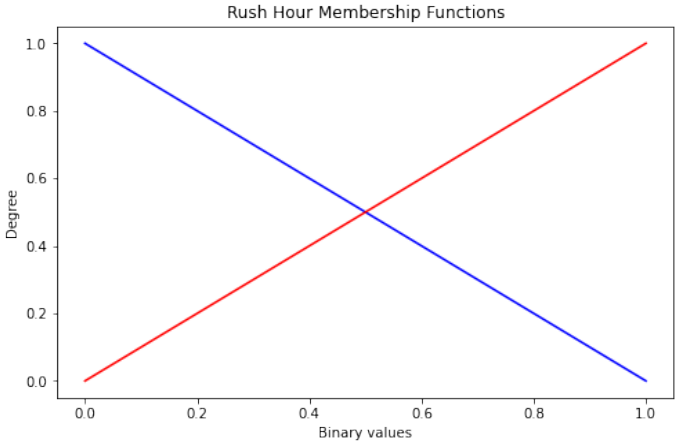
$$\mu(x)_{triangular} = \begin{cases} 0, & x \ \leq \ or\ x \geq c, \\ \frac{x-a}{b-a}, & x \prec a \leq b, \\ \frac{c-x}{c-b}, & b \prec x \leq c. \end{cases} \tag{5}$$

$$\mu(x)_{trapezoidal} = \begin{cases} 0, & x \ \leq a\ or\ x \geq d, \\ \frac{x-a}{b-a}, & a \prec x \leq b, \\ 1, & b \prec x \leq c, \\ \frac{d-x}{d-c}, & c \prec x \prec d. \end{cases} \tag{6}$$

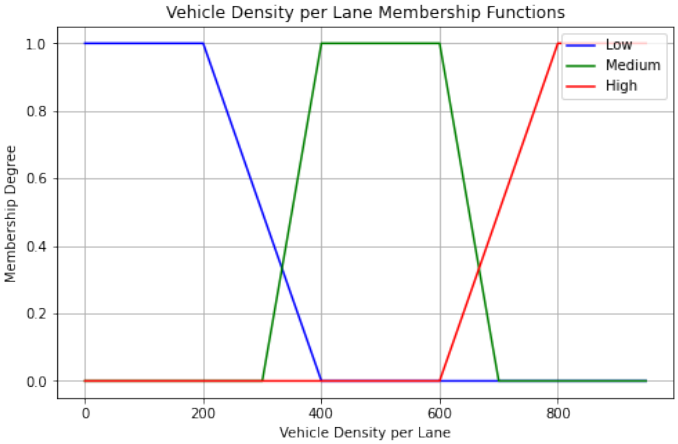
The two graphs illustrates the two utilized membership function that are the formed-base for fuzzy rules preparation. The "Vehicle Density per Lane," for instance, is divided into 3 level using trapezoidal membership function where we identify the distribution of range first [0 to 950] and manually selected each range for the three levels. For the binary variables like "Is Weekday" and "Rush Hour," an example of membership function of triangular membership function is shown. Each variables involved in any of fuzzy systems in this project is fuzzified with its according membership function based on the variables data types.

C. Fuzzy Inference Systems

Fuzzy rules are the core of each fuzzy systems. These each lines forms a backbone connecting our input to a controlled system output. [9] The importance of setting the fuzzy rules in this case comes with its significant challenges. However, we



(a) Triangular



(b) Trapezoidal

Fig. 1: Membership Functions for Fuzzy Systems

group the rules by stages. The fuzzy system for traffic data takes "rush hour," "is weekend," "vehicle density per lane," and "heavy vehicle percentage" into fuzzy rule formation. The fuzzy rule for traffic fuzzy system is using possibility of permutation and combination selecting of the most impacting and suitable ones. Next, for weather inputs, we consider the sequence in the following: "rain" in severity level, and "snow" in severity level, and in combination of both as the highest order. In following, we introduced "wind speed 10m." The weather inputs serves as a guidelines as to how visible the condition of the road is as the data lacks visibility input, we deem it of significant importance to have certain degree of threshold to add visibility into our fuzzy system. Here it is crucial to identify the guidance of weather fuzzy rules. It has been tested and estimated that Rain and Snow are two key indicator when it comes to traffic flow reduction especially in weather to traffic design. [10] The U.S. Department of Transportation provides the estimated percentage of impacts on mobility and is key acknowledgment of our threshold setting for weather fuzzy rules. Here, we alternated the column of average speed in accordance to the percentage shown. As

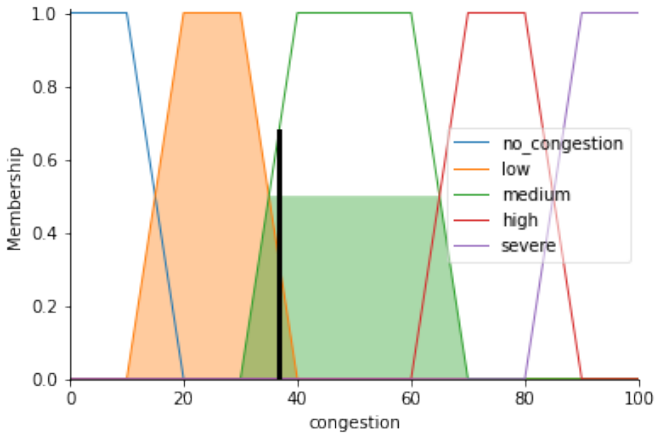


Fig. 2: Fuzzy System of Congestion Level Simulation Output

previously discussed, we prioritized rain and snow as our fuzzy rule defining for this matter.

TABLE I: Weather to Traffic Flow Reductions.[10]

Weather Conditions	Traffic Reudction by %		
	Average Speed	*Volume*	*Capacity*
Light Rain/Snow	3%-13%	5%-10%	4%-11%
Heavy Rain	3%-16%	14%	10%-30%
Heavy Snow	5%-40%	12%-27%	10%-30%
Low Visibility	10%-12%	N.A.	12%

Entering the last stage of a formation of fuzzy system, we activate via python package skfuzzy and alternate our crisp value into antecedent to determine its degree of memberships in the above created fuzzy sets. We do so with also our output membership function to complete our fuzzy systems for both weather and traffic separately. Here we defuzzify by the centroid method from antecedent and aggregated fuzzy sets which is also for fuzzy C means clustering (FCM).

In total of the first-stage in this project, we tested out three fuzzy systems: one solely using traffic dataset to output simulation of traffic vehicle level, one solely using weather dataset to output simulation of weather severity level, and third using combined variables of both dataset as a testing for fuzzy systems to fuzzy clustering methods.

The first two fuzzy systems are further used into the forming of fuzzy clustering as the output for both fuzzy systems serves as a third layer of variables of our initial dataset and in combination are a possible thorough fuzzy logic clustering application. Shown in Fig. 2, a single sample input of congestion data-point was tested and the fuzzy system presented a congestion traffic level of 36.78 on the scale of 0 to 100. This locates the congestion level at medium level. The reason for further investigate the possibility of fuzzy clustering method instead of simply using the third fuzzy system is based on the few shortcomings we encountered during the project. Despite of the vast applicable situation and easy-constructed nature of fuzzy rules, we are observing a few limitations that cannot overcome to be implemented into application. We discovering a few overlapping situations and the lack of scalability is hard to apply for all other datasets in traffic data we aim to test. The discussion of limitation of fuzzy systems will be redirected in the discussion section.

IV. Implementation with Fuzzy Clustering

To classify and structure our data for rule implementation based on identified clusters, we employed the c-means clustering method. This technique is particularly effective for unlabeled data and excels at uncovering patterns within unstructured datasets. While models like ANFIS (Adaptive Neuro-Fuzzy Inference System) could offer advanced capabilities, their requirement for training data with target outputs posed a significant challenge for us, especially given our lack of prior experience with such models. C-means clustering integrates seamlessly with fuzzy logic systems because each data point possesses a degree of membership across all clusters. This characteristic is invaluable for establishing membership functions within fuzzy systems. Each cluster is defined by its centroid, the weighted average of all data points calculated based on their membership values to the clusters.

Our parameters for C-means clustering includes features such as weather code, total number of cars, heavy vehicle count, and rush hour indicators. While one cluster is designated per ten features or labels. We adjusted the cluster amount accordingly for our dataset. In the model, the fuzziness parameters (m) is the key hyper-parameter that determines the degree of overlap between clusters and convergence criterion (ε) is the threshold of which we deem achieving the convergence of cluster centroids.

As similar to fuzzy system procedure, the FCM follows the steps of membership martix set-up, cluster centroids computation, mambership values tuning, and convergence check. For membership matrix, instead of assigning each data point to a single cluster, we assign membership scores to each data point for all clusters. For example, a data point might have 60% membership in Cluster A, 10% in Cluster B, and 30% in Cluster C. In the calculation of the centroid for each cluster, they are computed by the weighted average of all data points, using their membership scores as weights. Membership values and convergence check are both finalized via the program as the iterative process either terminates of continues until stabilization.

V. Results and Discussion

The results were as expected for the fuzzy system. Based on our data, we had no real way to determine if the roads were congested or not, but we used some mathematical formulas and common sense to make the fuzzy rules. The test results scored 100% for the fuzzy system, largely due to the dynamic way of generating the rules. We were shocked that this actually worked, given that we faced similar problems with the fuzzy system as we did with the clustering script earlier in the process. That said, the accuracy could certainly be improved. Using only “low,” “medium,” and “high” labels may have been too simplistic. According to the graphs, some

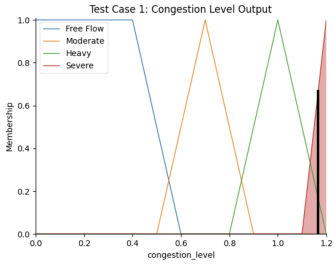


Fig. 3: Test Case 1: High total cars, rush hour, low heavy

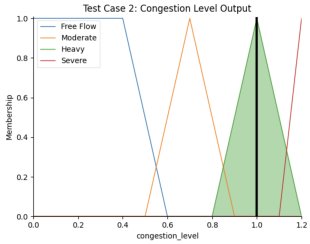


Fig. 4: Test Case 2: Medium total cars, non-rush hour, medium heavy vehicles, rainy weather

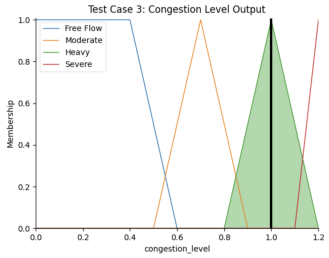


Fig. 5: Test Case 3: Low total cars, rush hour, high heavy vehicles, snowy weather

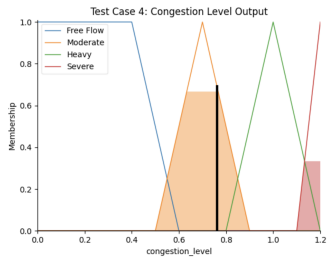


Fig. 6: Test Case 4: Medium total cars, rush hour, low heavy vehicles, cloudy weather

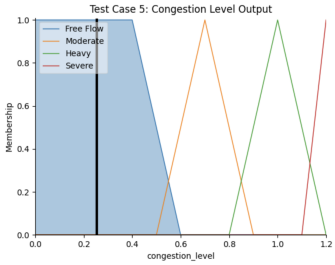


Fig. 7: Test Case 5: Low total cars, non-rush hour, low heavy vehicles, clear weather

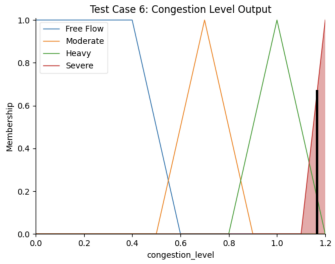


Fig. 8: Test Case 6: High total cars, rush hour, high heavy vehicles, rainy weather

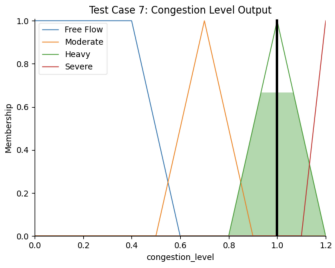


Fig. 9: Test Case 7: Medium total cars, non-rush hour, medium heavy vehicles, snowy weather

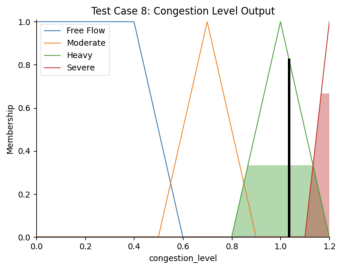


Fig. 10: Test Case 8: High total cars, rush hour, medium heavy vehicles, cloudy weather

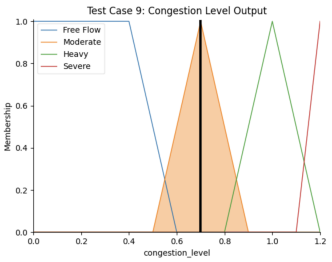


Fig. 11: Test Case 9: Low total cars, non-rush hour, high heavy vehicles, rainy weather

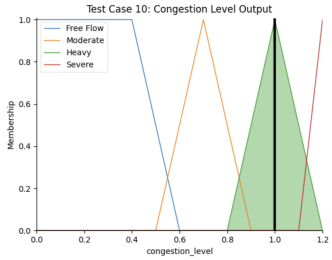


Fig. 12: Test Case 10: High total cars, non-rush hour, low heavy vehicles, clear weather

results were very similar, especially when the system predicted heavy congestion based on the heavy vehicle (H/V) score. By introducing an additional label, such as “severe congestion,” the results could potentially be more distinct and accurate. However, adding more labels and variables would require redesigning the entire dynamic rule-setting system. Given our time constraints, we chose not to pursue this approach and instead focused on clustering. Unfortunately, when we applied clustering, we struggled significantly. The membership functions initially generated poor rules, leading to outputs that were unusable. While we attempted to create rules based on the clustering centroids, this approach failed to produce meaningful results. As a result, we were unable to make clustering work effectively within this project.

A. Challenges

One of the core challenges with fuzzy clustering was the inability to adapt it effectively to our data. Although fuzzy clustering introduces ”freedom” to the model by grouping data dynamically, it relies heavily on the quality of initial parameters, such as the number of clusters (c) and initial cluster centroids. For our dataset, which focused on Dal krysset as the only data point, these challenges became pronounced. Without sufficient diversity in the dataset, clustering struggled to converge meaningfully, and the rules generated were poor and unreliable. Additionally, while fuzzy clustering offers flexibility, it lacks the transparency of rule-based systems. The unsupervised nature of clustering makes it harder to understand and control the thresholds influencing the outputs. In our case, this lack of interpretability made it difficult to debug and fine-tune the clustering process.

TABLE II: Test Cases Overview

Congestion	Inputs		
	Total Cars	***Rush Hour***	***Heavy Vehicle %***
Heavy 1.17	4500	Yes	5%
Heavy 1.00	2500	No	15%
Heavy 1.00	800	Yes	25
Moderate 0.76	3000	Yes	5%
Free Flow 0.25	500	No	5%
Heavy 1.17 0.76	4000	Yes	25%
Heavy 1.00 0.76	2000	No	15%
Heavy 1.04	3500	Yes	15%
Moderate 0.70	1000	No	30%
Heavy 1.00	4000	Yes	5%

B. Strengths and Weaknesses

The fuzzy system, on the other hand, performed surprisingly well given its static nature. By relying on clear mathematical relationships and simple rules, the fuzzy system produced interpretable results, even if they were limited by the constraints of the dataset. For example, we noticed that heavy congestion predictions were sometimes overly influenced by the H/V score, even when other factors, like total cars, suggested otherwise. This highlights the strength of fuzzy systems in providing transparent outputs, but also their weakness in adapting to complex, dynamic datasets. The results indicated that while the fuzzy system excelled at handling predefined scenarios, it struggled with generalization. Since our dataset was based entirely on Dal krysset, the system was heavily biased toward this specific traffic checkpoint. Expanding the dataset to include other locations would likely reduce this bias, but doing so would require significant reconfiguration of the rule-based system.

C. Future Directions

Looking back at this project, there are several steps we could have taken to enhance the application and broaden its impact. A key improvement would involve incorporating more datapoints along the EV6 motorway and ensuring a connection between them. By integrating data from multiple checkpoints, the system could have a more comprehensive understanding of traffic conditions and generalize better to different areas. For instance, traffic patterns from a dataset at Dal krysset could have been compared and integrated with patterns observed at Helsfyr, providing a more dynamic and interconnected prediction system. Another critical improvement lies in the use of weather data. Currently, our weather data includes basic conditions such as precipitation, visibility, and wind speed. However, adding more granular weather features, such as temperature and its interactions with precipitation, could enable the system to determine whether roads are icy. For example, conditions like heavy snowfall followed by slight warming could make roads particularly hazardous, and incorporating this into the fuzzy logic system would improve its predictive capability.

Fuzzy clustering presented significant challenges in this project, and one area of improvement would have been ex-

ploring alternative clustering algorithms. While we primarily focused on one clustering method, we had considered implementing C-means clustering as a second contender. However, our ability to fully resolve the first clustering algorithm gives us large consuming fixated solving puzzle to test this alternative. Future iterations of the project could compare the performance of multiple clustering algorithms to determine which is most suitable for this type of data and problem.

The fuzzy clustering method we used also faced a key limitation: it failed to generate rules for every possible membership scenario. This lack of comprehensive rule generation resulted in unreliable outputs for certain test cases. A potential solution could involve experimenting with supervised or hybrid clustering techniques, where clustering is guided by predefined conditions or rules. Another approach could include revisiting and refining the initialization parameters, such as the number of clusters and centroids, to better align with the dataset's structure.

1) Integration of More Traffic Checkpoints: Expanding the dataset to include multiple checkpoints along the EV6 motorway would provide a more robust foundation for both fuzzy systems and clustering. These additional data-points could capture diverse traffic patterns, making the model more adaptable.
2) Enhanced Weather Features: Including advanced weather features, such as a calculated "icy road probability" based on temperature and precipitation data, could improve the model's ability to predict hazardous conditions.
3) Alternative Clustering Algorithms: Exploring different clustering methods, such as C-means clustering or density-based clustering (DBSCAN), could provide insights into better handling the dataset's structure. These methods might also offer more flexibility in generating comprehensive membership rules.
4) Dynamic Rule Generation: Refining the rule-generation process to ensure that every possible membership scenario is accounted for would address one of the key weaknesses in the current implementation.
5) Real-Time Monitoring: In the long term, integrating the fuzzy system and clustering models into a real-time monitoring application could provide dynamic traffic predictions and live updates, benefiting traffic management and planning.
6) Supervised Learning Integration: Combining fuzzy logic with supervised machine learning models, such as neural networks, could enhance the system's adaptability while retaining the interpretability of fuzzy systems.

VI. Conclusion

One of the most significant barriers to achieving the project's full potential was time management. If more time had been allocated to the project, we could have connected additional data-points, extracted more data, and performed deeper analyses. Expanding the dataset would not only improve the fuzzy system but also potentially address some of the clustering algorithm's limitations. With more diverse data, the clustering process might converge more effectively, providing meaningful groupings and improving rule generation.

The lessons learned from this project provide a clear path for future enhancements. While the fuzzy system demonstrated strong interpret-ability and accuracy within its constraints, the clustering method faced challenges that need to be addressed in subsequent iterations. By incorporating more data-points, improving weather data integration, exploring alternative clustering methods, and focusing on real-time applications, the project could evolve into a comprehensive and adaptable traffic prediction system. Moving forward, time management and strategic planning will be critical to overcoming these challenges and realizing the full potential of fuzzy logic and clustering in intelligent transportation systems.

References

[1] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-Driven Intelligent Transportation Systems: A survey," IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pp. 1624–1639, Jul. 2011, doi: 10.1109/tits.2011.2158001.

[2] Y. Hou, Z. Deng, and H. Cui, "Short-Term Traffic Flow Prediction with Weather Conditions: Based on Deep Learning Algorithms and Data Fusion," Complexity, vol. 2021, no. 1, Jan. 2021, doi: 10.1155/2021/6662959.

[3] "Vegvesen trafikk: Trafikkmeldinger, webkamera og ruteplanlegger," Statens Vegvesen. https://www.vegvesen.no/trafikk.

[4] "Integrated Forecasting System," ECMWF. https://www.ecmwf.int/en/forecasts/documentation-and-support/changes-ecmwf-model

[5] "CERRA sub-daily regional reanalysis data for Europe on single levels from 1984 to present," Aug. 02, 2022. https://cds.climate.copernicus.eu/datasets/reanalysis-cerra-single-levels?tab=overview

[6] "Simplified Highway Capacity Calculation Method - Chapter 2 - Policy — Federal Highway Administration." https://www.fhwa.dot.gov/policyinformation/pubs/pl18003/chap02.cfm.

[7] "Oslo traffic report — TomTom Traffic Index," Oslo Traffic Report — TomTom Traffic Index. https://www.tomtom.com/traffic-index/oslo-traffic/

[8] H.-D. He, J.-L. Wang, H.-R. Wei, C. Ye, and Y. Ding, "Fractal behavior of traffic volume on urban expressway through adaptive fractal analysis," Physica a Statistical Mechanics and Its Applications, vol. 443, pp. 518–525, Oct. 2015, doi: 10.1016/j.physa.2015.10.004.

[9] J. Bohidar, "2 Fundamentals of fuzzy logic control – fuzzy sets, fuzzy rules and defuzzifications," Apr. 2016, [Online]. Available: $https://www.academia.edu/24675683/2_Fundamentals_of_Fuzzy_Logic_Control_Fuzz$

[10] "How do weather events impact roads? - FHWA Road Weather Management." $https://ops.fhwa.dot.gov/weather/q1_roadimpact.html$

Appendix: ReadME.

README

Traffic Congestion Prediction Using Fuzzy Logic

This project involves predicting traffic congestion levels using fuzzy logic systems. To demonstrate the fuzzy logic model, a test script is provided that can be run without the need for extensive data extraction.

Running the Fuzzy Logic Test Script

Script: fuzzysystemtest.py

Purpose: The fuzzysystemtest.py script contains the implementation of the fuzzy logic system for predicting traffic congestion. It uses predefined test cases to simulate and display the output of the model.

Running the Test Script

The fuzzysytemtest.py script demonstrates the fuzzy logic system's capabilities using predefined test cases. Follow the steps below to execute the script:

1. **Ensure All Necessary Files Are Present:**
    - fuzzysytemtest.py
    - constants.py
2. Both files should reside in the same directory to ensure proper import and execution.
    - **Install Required Libraries:**
      If you haven't installed the necessary Python libraries yet, do so using the following command:
      bash
      Copy code
      pip install numpy scikit-fuzzy matplotlib
3.
    - **Execute the Script:**
      Open a terminal or command prompt in the project directory and run the following command:
      bash

Copy code
python fuzzysytemtest.py

4. **View the Results:**
    - **Console Output:** The script will display the predicted congestion levels for each test case directly in the terminal.
    - **Result File:** A text file named test_case_result.txt will be generated, containing detailed results of each test case.
    - **Visualizations:** For each test case, a corresponding image file (e.g., test_case_1_output.png) will be saved in the current directory, illustrating the congestion level output.

Understanding the Output

Upon running the fuzzysystemtest.py script, you will receive both textual and visual outputs that detail the predicted traffic congestion levels based on the predefined test cases.

1. **Console Output:**
   Each test case will display:
    - **Test Case Number and Description:** Provides context about the scenario being tested.
    - **Inputs:** Shows the values of total_cars, rush_hour, heavy_vehicle_percentage, and weather_category used for the prediction.
    - **Predicted Congestion Level (V/C Ratio):** Numerical value representing the congestion level.
    - **Congestion Status:** Categorical interpretation of the V/C Ratio (e.g., Free Flow, Moderate, Heavy, Severe).
    - **Example:**
      yaml
      Copy code

    - Test Case 1: High total cars, rush hour, low heavy vehicles, clear weather
    - Predicted Congestion Level (V/C Ratio): 0.65
    - Congestion Status: Moderate
2.
3. **Result File (test_case_result.txt):**
   Contains detailed logs of each test case, mirroring the console output for record-keeping and further analysis.
4. **Visualizations (test_case_X_output.png):**
   Graphical representations of the fuzzy logic system's interpretation of congestion levels for each test case. These plots illustrate how input variables are mapped to congestion levels based on the defined fuzzy rules and membership functions.
    -

Notes

- **Data Extraction:**
  - The other scripts in the project are designed to download, process, and analyze large datasets of traffic and weather information.
  - This data extraction process can be time-consuming and may require significant computational resources.
  - To simplify the demonstration, the fuzzonday.py script uses predefined test cases and does not require external data.
- **Dependencies:**
  - Ensure all required Python packages are installed before running the script.
  - If you encounter any issues with missing packages, you can install them individually using pip.
- **Mistakes:**
  - We are aware upon checking the code that we have left some codeparts with norwegian comments.

Appendix: Python files

constants.py

```
Roadids = {
    "Dalkrysset": {
        "roadid": '87610V1811579',
        "northbound": "Totalt i retning HAMAR",
        "southbound": "Totalt i retning OSLO"
    }
}


weather_code_mapping = {
    # Rain
    51: 0, 53: 0, 55: 0, 56: 0, 57: 0, 61: 0, 63: 0, 65: 0, 66: 0, 67:
0,
    80: 0, 81: 0, 82: 0, 96: 0, 99: 0,

    # Snow
    71: 1, 73: 1, 75: 1, 77: 1, 85: 1, 86: 1,

    # Fog
    45: 2, 48: 2,

    # Clear
    0: 3, 1: 3, 2: 3, 3: 3,

    # Other
    95: 4,
}

# List of test cases
test_cases = [
    {
        'total_cars': 4500,                  # High
        'rush_hour': 1,                      # Yes
        'heavy_vehicle_percentage': 5,       # Low
        'weather_category': 0,               # Clear
```

```
        'description': 'High total cars, rush hour, low heavy vehicles, clear weather'
    },
    {
        'total_cars': 2500,                  # Medium
        'rush_hour': 0,                      # No
        'heavy_vehicle_percentage': 15,      # Medium
        'weather_category': 2,               # Rainy
        'description': 'Medium total cars, non-rush hour, medium heavy vehicles, rainy weather'
    },
    {
        'total_cars': 800,                   # Low
        'rush_hour': 1,                      # Yes
        'heavy_vehicle_percentage': 25,      # High
        'weather_category': 3,               # Snowy
        'description': 'Low total cars, rush hour, high heavy vehicles, snowy weather'
    },
    {
        'total_cars': 3000,                  # Medium
        'rush_hour': 1,                      # Yes
        'heavy_vehicle_percentage': 5,       # Low
        'weather_category': 1,               # Cloudy
        'description': 'Medium total cars, rush hour, low heavy vehicles, cloudy weather'
    },
    {
        'total_cars': 500,                   # Low
        'rush_hour': 0,                      # No
        'heavy_vehicle_percentage': 5,       # Low
        'weather_category': 0,               # Clear
        'description': 'Low total cars, non-rush hour, low heavy vehicles, clear weather'
    },
    {
        'total_cars': 4000,                  # High
        'rush_hour': 1,                      # Yes
        'heavy_vehicle_percentage': 25,      # High
        'weather_category': 2,               # Rainy
        'description': 'High total cars, rush hour, high heavy vehicles, rainy weather'
    },
```

```
    {
        'total_cars': 2000,                    # Medium
        'rush_hour': 0,                        # No
        'heavy_vehicle_percentage': 15,        # Medium
        'weather_category': 3,                 # Snowy
        'description': 'Medium total cars, non-rush hour, medium heavy
vehicles, snowy weather'
    },
    {
        'total_cars': 3500,                    # High
        'rush_hour': 1,                        # Yes
        'heavy_vehicle_percentage': 15,        # Medium
        'weather_category': 1,                 # Cloudy
        'description': 'High total cars, rush hour, medium heavy
vehicles, cloudy weather'
    },
    {
        'total_cars': 1000,                    # Low
        'rush_hour': 0,                        # No
        'heavy_vehicle_percentage': 30,        # High
        'weather_category': 2,                 # Rainy
        'description': 'Low total cars, non-rush hour, high heavy
vehicles, rainy weather'
    },
    {
        'total_cars': 4000,                    # High
        'rush_hour': 0,                        # No
        'heavy_vehicle_percentage': 5,         # Low
        'weather_category': 0,                 # Clear
        'description': 'High total cars, non-rush hour, low heavy
vehicles, clear weather'
    },
]
```

dataprocesser.py

```
import os
import pandas as pd
import chardet
import csv
#for dataset without ; as seperator
```

```
def process_traffic_data(file_path, start_date, end_date):
    # Load the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Ensure the "Dato" column is a datetime format for filtering
    df['Dato'] = pd.to_datetime(df['Dato'], format='%Y-%m-%d')

    # Filter the rows based on the date range
    df = df[(df['Dato'] >= pd.to_datetime(start_date)) & (df['Dato'] <=
pd.to_datetime(end_date))]

    # Filter rows to keep only those where "Felt" starts with "Totalt i
retning "
    df = df[df["Felt"].str.startswith("Totalt i retning ")]

    # List to store output data
    output_data = []

    # Loop through each unique hour and collect results
    grouped = df.groupby(["Dato", "Fra tidspunkt"])
    for (date, time_range), group in grouped:
        # Get unique directions for this group (e.g., different cities)
        directions = group["Felt"].unique()

        for direction in directions:
            direction_data = group[group["Felt"] == direction]
            if not direction_data.empty:
                total_cars = direction_data["Trafikkmengde"].values[0]
                car_sizes = {
                    "< 5,6m": direction_data["< 5,6m"].values[0],
                    ">= 5,6m": direction_data[">= 5,6m"].values[0],
                    "5,6m - 7,6m": direction_data["5,6m -
7,6m"].values[0],
                    "7,6m - 12,5m": direction_data["7,6m -
12,5m"].values[0],
                    "12,5m - 16,0m": direction_data["12,5m -
16,0m"].values[0],
                    ">= 16,0m": direction_data[">= 16,0m"].values[0],
                    "16,0m - 24,0m": direction_data["16,0m -
24,0m"].values[0],
                    ">= 24,0m": direction_data[">= 24,0m"].values[0],
                }
```

```
                # Append a row of data to the list
                output_data.append({
                    "Date": date.strftime('%Y-%m-%d'),
                    "Time Range": time_range,
                    "Direction": direction,
                    "Total Cars": total_cars,
                    "< 5.6m": car_sizes["< 5,6m"],
                    ">= 5.6m": car_sizes[">= 5,6m"],
                    "5.6m - 7.6m": car_sizes["5,6m - 7,6m"],
                    "7.6m - 12.5m": car_sizes["7,6m - 12,5m"],
                    "12.5m - 16.0m": car_sizes["12,5m - 16,0m"],
                    ">= 16.0m": car_sizes[">= 16,0m"],
                    "16.0m - 24.0m": car_sizes["16,0m - 24,0m"],
                    ">= 24.0m": car_sizes[">= 24,0m"]
                })

    # Convert the list of dictionaries to a DataFrame
    output_df = pd.DataFrame(output_data)

    # Return the DataFrame for further use
    return output_df


def clean_data(file_path):
    # Detect the file encoding
    with open(file_path, 'rb') as f:
        result = chardet.detect(f.read())
    encoding = result['encoding']

    try:
        # Load the CSV file with the detected encoding
        data = pd.read_csv(file_path, delimiter=';', encoding=encoding)

        # Inspect column names for debugging
        print("Columns in the file:", data.columns.tolist())

        # Drop unnecessary columns
        columns_to_drop = [
            'Fra', 'Til', 'Vegreferanse', 'Dekningsgrad (%)',
            'Antall timer total', 'Antall timer inkludert',
            'Antall timer ugyldig', 'Ikke gyldig lengde',
            'Lengdekvalitetsgrad (%)'
        ]
```

```
        existing_columns_to_drop = [col for col in columns_to_drop if
col in data.columns]
        data = data.drop(columns=existing_columns_to_drop, axis=1)

        # Replace all `-` values with `0` to ensure consistent numeric
data
        data.replace('-', 0, inplace=True)

        # Convert numeric columns to proper types
        data = data.apply(pd.to_numeric, errors='ignore')

        # Filter rows to keep only those where "Felt" starts with
"Totalt i retning"
        if 'Felt' in data.columns:
            data = data[data['Felt'].str.startswith('Totalt')]
        else:
            print("Error: 'Felt' column not found. Ensure the CSV file
has the correct format.")
            return None

        # Extract everything before ".csv"
        base_name = file_path.split(".csv")[0]
        # Save the cleaned data to a new CSV file
        output_file = f'{base_name}_cleaned.csv'
        data.to_csv(output_file, index=False, encoding='utf-8')
        os.remove(file_path)
        print(f"Processing complete. Filtered data saved to
'{output_file}'.")
        return output_file

    except Exception as e:
        print(f"Error processing file {file_path}: {e}")
        return None
```

weatherhistoryapi.py

```
import openmeteo_requests
import requests_cache
import pandas as pd
from retry_requests import retry

def fetch_weather_data(latitude, longitude, start_date, end_date,
hourly_variables, timezone, output_file):
    """
```

```
    Fetch weather data from the Open-Meteo API and save it to a CSV
file.

    Parameters:
        latitude (float): Latitude of the location.
        longitude (float): Longitude of the location.
        start_date (str): Start date for the data in YYYY-MM-DD format.
        end_date (str): End date for the data in YYYY-MM-DD format.
        hourly_variables (list): List of hourly weather variables to
request.
        timezone (str): Timezone for the data.
        output_file (str): Path to save the weather data CSV file.

    Returns:
        pandas.DataFrame: DataFrame containing the weather data.
    """
    # Setup Open-Meteo API client with cache and retry mechanism
    cache_session = requests_cache.CachedSession('.cache',
expire_after=-1)
    retry_session = retry(cache_session, retries=5, backoff_factor=0.2)
    openmeteo = openmeteo_requests.Client(session=retry_session)

    # Request parameters
    url = "https://archive-api.open-meteo.com/v1/archive"
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "start_date": start_date,
        "end_date": end_date,
        "hourly": hourly_variables,
        "timezone": timezone
    }

    # Fetch weather data
    responses = openmeteo.weather_api(url, params=params)

    # Process the first response
    response = responses[0]
    print(f"Coordinates {response.Latitude()}°N
{response.Longitude()}°E")
    print(f"Elevation {response.Elevation()} m asl")
    print(f"Timezone {response.Timezone()}
{response.TimezoneAbbreviation()}")
```

```
    print(f"Timezone difference to GMT+0 {response.UtcOffsetSeconds()}
s")

    # Extract hourly data
    hourly = response.Hourly()
    hourly_data = {
        "date": pd.date_range(
            start=pd.to_datetime(hourly.Time(), unit="s", utc=True),
            end=pd.to_datetime(hourly.TimeEnd(), unit="s", utc=True),
            freq=pd.Timedelta(seconds=hourly.Interval()),
            inclusive="left"
        )
    }

    for i, variable in enumerate(hourly_variables):
        hourly_data[variable] = hourly.Variables(i).ValuesAsNumpy()

    # Create DataFrame
    hourly_dataframe = pd.DataFrame(data=hourly_data)

    # Save DataFrame to CSV
    hourly_dataframe.to_csv(output_file, index=False)
    print(f"Weather data saved to {output_file}")

    return hourly_dataframe
```

trafficdownload.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager
import os
import time
import pandas as pd
import pandas as pd
import chardet
import csv


class TrafficDataDownloader:
    def __init__(self, download_dir):
        """Initialize the TrafficDataDownloader class."""
```

```
        self.download_dir = os.path.join(os.getcwd(), download_dir)
        self.driver = None
        print(self.download_dir)
        print("Current Working Directory:", os.getcwd())  # Directory
the script is running in
        print("Resolved Trafficdata Directory:", self.download_dir)  #
Resolved download directory
        # Ensure the download directory exists
        if not os.path.exists(self.download_dir):
            os.makedirs(self.download_dir)



        self.setup_driver()

    def setup_driver(self):
        """Sets up the WebDriver with necessary options."""
        options = webdriver.ChromeOptions()
        prefs = {
            "download.default_directory": self.download_dir,  # Set
default download directory
            "download.prompt_for_download": False,           # Disable
download prompt
            "directory_upgrade": True,                       #
Automatically overwrite files
            "safebrowsing.enabled": True                     # Disable
Chrome's safe browsing feature
        }
        options.add_experimental_option("prefs", prefs)

        # Initialize the WebDriver
        self.driver = webdriver.Chrome(ChromeDriverManager().install(),
options=options)

    def generate_url(self, road_id, date_from, date_to,
lat=59.93623700438067, lon=10.874669090206872, datatype="HOUR",
zoom=12):
        """Generate the URL dynamically for downloading traffic
data."""
        return (
            f"https://trafikkdata.atlas.vegvesen.no/#/eksport?"
            f"datatype={datatype}&from={date_from}&lat={lat}&lon={lon}"
            f"&to={date_to}&trpids={road_id}&zoom={zoom}"
```

```
        )

    def download_file(self, name, road_id, date_from, date_to,
lat=60.2751, lon=11.179, datatype="HOUR", zoom=12):
        """Download the traffic data file for a given road ID and date
range."""
        url = self.generate_url(road_id, date_from, date_to, lat, lon,
datatype, zoom)
        self.driver.get(url)

        try:
            # Wait until the download button is clickable
            wait = WebDriverWait(self.driver, 10)
            button =
wait.until(EC.element_to_be_clickable((By.CLASS_NAME, "knapp-liten")))

            # List files before clicking the button
            existing_files = set(os.listdir(self.download_dir))

            # Click the download button
            button.click()
            print("Download started...")

            # Wait for the new file to appear
            time.sleep(5)  # Allow the download process to start
            new_files = set(os.listdir(self.download_dir)) -
existing_files
            print(new_files)

            if not new_files:
                raise Exception("No new file detected. Ensure the
download directory is correct.")

            # Detect the downloading file
            downloading_file = list(new_files)[0]
            base_file_name = downloading_file.split(".csv")[0] + ".csv"
            print(f"Detected base file name: {base_file_name}")

            # Wait for the file to complete downloading
            self.wait_for_download(base_file_name)

            # Rename the file with a custom name
            new_name = f"{name}_data_{date_from}_to_{date_to}.csv"
```

```
            os.rename(
                os.path.join(self.download_dir, base_file_name),
                os.path.join(self.download_dir, new_name)
            )
            print(f"File renamed to: {new_name}")

        except Exception as e:
            print(f"An error occurred during download: {e}")

    def download_multi_files(self, roadids, date_from, date_to):
        for location, road_info in roadids.items():
            road_id = road_info.get('roadid')  # Extract roadid
            print(f"Downloading data for {location} (Road ID:
{road_id})")

            # Download file using the roadid
            self.download_file(
                name=location,  # Use location name as the file name
                road_id=road_id,
                date_from=date_from,
                date_to=date_to
            )



    def wait_for_download(self, base_file_name, max_wait_time=300):
        """Waits for the file to complete downloading."""
        start_time = time.time()
        while True:
            files = os.listdir(self.download_dir)
            if base_file_name in files:
                print(f"Download complete: {base_file_name}")
                return
            if time.time() - start_time > max_wait_time:
                raise TimeoutError("Download did not complete within
the expected time.")
            #print("File is still downloading...")
            time.sleep(2)


    def quit(self):
        """Closes the WebDriver."""
        if self.driver:
```

```
            self.driver.quit()
```

fuzzy_clustering.py `import numpy as np`

```
import pandas as pd
import matplotlib.pyplot as plt
from skfuzzy.cluster import cmeans
from constants import weather_code_mapping
from skfuzzy import control as ctrl
import skfuzzy as fuzz
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler

# Step 0: Load Data
def load_data(road_data, weather_data, weather_code_mapping):
    """
    Load and merge road and weather data based on Date and Time Range.

    Parameters:
        road_data (str): Path to the road data CSV file.
        weather_data (str): Path to the weather data CSV file.
        weather_code_mapping (dict): Dictionary mapping weather codes
to categories.

    Returns:
        pandas.DataFrame: Merged and cleaned dataset.
    """
    import pandas as pd

    # Load the road and weather data
    road_data = pd.read_csv(road_data)
    weather_data = pd.read_csv(weather_data)

    # Format 'date' column in weather data
    weather_data['date'] = pd.to_datetime(weather_data['date'])  #
Ensure datetime format
    weather_data['Date'] = weather_data['date'].dt.strftime('%Y-%m-%d')
# Extract date
```

```
    weather_data['Time Range'] =
weather_data['date'].dt.strftime('%H:%M')  # Extract time
    weather_data['weather_category'] =
weather_data['weather_code'].map(weather_code_mapping)  # Map weather
categories

    # Drop unnecessary columns in weather data
    weather_data = weather_data.drop(columns=['date', 'weather_code'])

    # Ensure 'Time Range' in road_data is in HH:MM format
    road_data['Time Range'] = road_data['Time Range'].apply(lambda x:
x[:5])

    # Merge road data and weather data on 'Date' and 'Time Range'
    merged_data = pd.merge(road_data, weather_data, on=['Date', 'Time
Range'], how='inner')

    # Drop rows with missing values (optional)
    return merged_data.dropna()



# Step 1: Prepare Data
def preprocess_data(data):
    # Select relevant columns
    features = data[['weather_category', 'Total Cars','Heavy Vehicle
Percentage', 'Rush Hour']]
    # Normalize features
    normalized_features = (features - features.min()) / (features.max()
- features.min())
    return normalized_features.values

# Step 2: Apply Fuzzy C-Means
def fuzzy_cmeans_clustering(data, n_clusters):
    # Transpose data for FCM
    data_transposed = data.T
    # Apply Fuzzy C-Means
    cntr, u, u0, d, jm, p, fpc = cmeans(data_transposed, c=n_clusters,
m=2, error=0.005, maxiter=10000)
    # Assign clusters
    cluster_labels = np.argmax(u, axis=0)
    return cluster_labels, cntr, n_clusters
```

```python
# Step 3: Visualize Clusters
def visualize_fcm_clusters(data, cluster_labels, feature_names):
    plt.figure(figsize=(10, 6))
    for cluster in np.unique(cluster_labels):
        cluster_data = data[cluster_labels == cluster]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1],
label=f'Cluster {cluster + 1}')
    plt.title('Fuzzy C-Means Clustering')
    plt.xlabel(feature_names[0])
    plt.ylabel(feature_names[1])
    plt.legend()
    plt.show()


def visualize_fcm_clusters_pos_3D(data, labels, cntr):
    """
    Visualize Possibility C-Means Clusters in 3D.

    Parameters:
    - data: np.ndarray
        The dataset in 3D (N samples x 3 features).
    - labels: np.ndarray
        Cluster labels for each data point.
    - cntr: np.ndarray
        Cluster centers.
    """
    # Create a 3D plot
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot each cluster
    for cluster in range(len(cntr)):
        ax.scatter(
            data[labels == cluster, 0],  # X-axis
            data[labels == cluster, 1],  # Y-axis
            data[labels == cluster, 2],  # Z-axis
            label=f"Cluster {cluster}",
            alpha=0.6
        )
```

```
    # Plot cluster centers
    ax.scatter(
        cntr[:, 0],  # X-axis
        cntr[:, 1],  # Y-axis
        cntr[:, 2],  # Z-axis
        s=300, c='red', marker='X', label='Centroids'
    )

    # Set plot labels and title
    ax.set_title('Possibility C-Means Clustering (3D)', fontsize=14)
    ax.set_xlabel('Feature 1', fontsize=12)
    ax.set_ylabel('Feature 2', fontsize=12)
    ax.set_zlabel('Feature 3', fontsize=12)
    ax.legend()
    plt.show()
```

fuzzysystemwithcluster.py

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
from fuzzy_clustering import load_data, preprocess_data,
fuzzy_cmeans_clustering
from constants import weather_code_mapping,test_cases

# Load and preprocess data
merged_data = load_data(
    'V4/trafficdata/north/n_Dalkrysset_data_2020-10-31_to_2024-10-
31_cleaned.csv',
    'weatherdata.csv',
    weather_code_mapping=weather_code_mapping,
)
data = preprocess_data(merged_data)

# Apply Fuzzy C-Means clustering
clusters = 5
labels, cntr, n_clusters = fuzzy_cmeans_clustering(data, clusters)

# Define fuzzy variables
weather_category = ctrl.Antecedent(np.arange(0, 4, 1),
'weather_category')
total_cars = ctrl.Antecedent(np.arange(0, 5001, 1), 'total_cars')
```

```
rush_hour = ctrl.Antecedent(np.arange(0, 2, 1), 'rush_hour')
heavy_vehicle_percentage = ctrl.Antecedent(np.arange(0, 31, 1),
'heavy_vehicle_percentage')
congestion_level = ctrl.Consequent(np.arange(0, 1.4, 0.1),
'congestion_level')

# Membership Functions
weather_category.automf(names=['Clear', 'Cloudy', 'Rainy', 'Snowy'])
total_cars.automf(names=['Low', 'Medium', 'High'])
rush_hour.automf(names=['No', 'Yes'])
heavy_vehicle_percentage.automf(names=['Low', 'Medium', 'High'])
congestion_level.automf(names=['Free Flow', 'Moderate', 'Heavy',
'Severe'])

# Generate dynamic rules based on cluster centroids
rules = []
# Dynamically map centroids to fuzzy levels
for i, centroid in enumerate(cntr):
    # Use membership functions to determine the highest degree of
membership
    weather_degrees = {label:
fuzz.interp_membership(weather_category.universe,
weather_category[label].mf, centroid[0])
                       for label in weather_category.terms}
    weather_level = max(weather_degrees, key=weather_degrees.get)  #
Get the label with the highest degree

    total_cars_degrees = {label:
fuzz.interp_membership(total_cars.universe, total_cars[label].mf,
centroid[1])
                          for label in total_cars.terms}
    total_cars_level = max(total_cars_degrees,
key=total_cars_degrees.get)

    hv_degrees = {label:
fuzz.interp_membership(heavy_vehicle_percentage.universe,
heavy_vehicle_percentage[label].mf, centroid[2])
                  for label in heavy_vehicle_percentage.terms}
    hv_level = max(hv_degrees, key=hv_degrees.get)

    rush_hour_degrees = {label:
fuzz.interp_membership(rush_hour.universe, rush_hour[label].mf,
centroid[3])
```

```
                        for label in rush_hour.terms}
    rush_hour_level = max(rush_hour_degrees, key=rush_hour_degrees.get)

    # Define congestion based on a combination of centroid values
    congestion_degrees = {label:
fuzz.interp_membership(congestion_level.universe,
congestion_level[label].mf, centroid[3])
                          for label in congestion_level.terms}
    congestion = max(congestion_degrees, key=congestion_degrees.get)

    # Create rule dynamically
    rule = ctrl.Rule(
        weather_category[weather_level] &
        total_cars[total_cars_level] &
        heavy_vehicle_percentage[hv_level] &
        rush_hour[rush_hour_level],
        congestion_level[congestion]
    )
    rules.append(rule)
# Create fuzzy control system
congestion_ctrl = ctrl.ControlSystem(rules)
congestion_simulation = ctrl.ControlSystemSimulation(congestion_ctrl)


# Run and log test cases
with open('test_case_results.txt', 'w') as result_file:
    for i, test_case in enumerate(test_cases, start=1):
        # Set inputs for simulation
        congestion_simulation.input['total_cars'] =
test_case['total_cars']
        congestion_simulation.input['rush_hour'] =
test_case['rush_hour']
        congestion_simulation.input['heavy_vehicle_percentage'] =
test_case['heavy_vehicle_percentage']
        congestion_simulation.input['weather_category'] =
test_case['weather_category']

        # Compute the result
        congestion_simulation.compute()
        predicted_congestion =
congestion_simulation.output['congestion_level']

        # Map V/C ratio to status
```

```
        if predicted_congestion <= 0.6:
            congestion_status = 'Free Flow'
        elif predicted_congestion <= 0.9:
            congestion_status = 'Moderate'
        elif predicted_congestion <= 1.2:
            congestion_status = 'Heavy'
        else:
            congestion_status = 'Severe'

        # Write results to file and console
        result = f"Test Case {i}: {test_case['description']}\n"
        result += f"Inputs: {test_case}\n"
        result += f"Predicted Congestion Level:
{predicted_congestion:.2f} ({congestion_status})\n"
        result += "---\n"
        print(result)
        result_file.write(result)

        # Visualize and save output
        congestion_level.view(sim=congestion_simulation)
        plt.title(f"Test Case {i} Output")
        plt.savefig(f"test_case_{i}_output.png")
        plt.clf()
```

fuzzysystemtest.py

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
from constants import test_cases
# Define fuzzy variables
# Input Variables
weather_category = ctrl.Antecedent(np.arange(0, 4, 1),
'weather_category')
total_cars = ctrl.Antecedent(np.arange(0, 5001, 1), 'total_cars')
rush_hour = ctrl.Antecedent(np.arange(0, 2, 1), 'rush_hour')
heavy_vehicle_percentage = ctrl.Antecedent(np.arange(0, 31, 1),
'heavy_vehicle_percentage')

# Output Variable
congestion_level = ctrl.Consequent(np.arange(0, 1.3, 0.1),
'congestion_level')  # V/C Ratio
```

```
# Membership Functions
# Weather Category
weather_category['Clear'] = fuzz.trimf(weather_category.universe, [0,
0, 1])
weather_category['Cloudy'] = fuzz.trimf(weather_category.universe, [0,
1, 2])
weather_category['Rainy'] = fuzz.trimf(weather_category.universe, [1,
2, 3])
weather_category['Snowy'] = fuzz.trimf(weather_category.universe, [2,
3, 3])

# Total Cars
total_cars['Low'] = fuzz.trapmf(total_cars.universe, [0, 0, 1000,
1500])
total_cars['Medium'] = fuzz.trimf(total_cars.universe, [1000, 2500,
4000])
total_cars['High'] = fuzz.trapmf(total_cars.universe, [2500, 4000,
5000, 5000])

# Rush Hour
rush_hour['No'] = fuzz.trimf(rush_hour.universe, [0, 0, 1])
rush_hour['Yes'] = fuzz.trimf(rush_hour.universe, [0, 1, 1])

# Heavy Vehicle Percentage
heavy_vehicle_percentage['Low'] =
fuzz.trapmf(heavy_vehicle_percentage.universe, [0, 0, 5, 10])
heavy_vehicle_percentage['Medium'] =
fuzz.trimf(heavy_vehicle_percentage.universe, [5, 15, 25])
heavy_vehicle_percentage['High'] =
fuzz.trapmf(heavy_vehicle_percentage.universe, [15, 25, 30, 30])

# Congestion Level (V/C Ratio)
congestion_level['Free Flow'] = fuzz.trapmf(congestion_level.universe,
[0, 0, 0.4, 0.6])
congestion_level['Moderate'] = fuzz.trimf(congestion_level.universe,
[0.5, 0.7, 0.9])
congestion_level['Heavy'] = fuzz.trimf(congestion_level.universe, [0.8,
1.0, 1.2])
congestion_level['Severe'] = fuzz.trapmf(congestion_level.universe,
[1.1, 1.2, 1.3, 1.3])

# Define rules
```

```
rules = []

total_cars_levels = ['Low', 'Medium', 'High']
rush_hour_levels = ['No', 'Yes']
heavy_vehicle_levels = ['Low', 'Medium', 'High']
weather_levels = ['Clear', 'Cloudy', 'Rainy', 'Snowy']

def determine_congestion(total_cars_level, rush_hour_level, hv_level,
weather_level):
    # Base V/C ratio based on total cars
    if total_cars_level == 'Low':
        vc_ratio = 0.4
    elif total_cars_level == 'Medium':
        vc_ratio = 0.8
    else:  # High
        vc_ratio = 1.2

    # Adjust for heavy vehicles
    if hv_level == 'Medium':
        vc_ratio += 0.1
    elif hv_level == 'High':
        vc_ratio += 0.2

    # Adjust for rush hour
    if rush_hour_level == 'Yes':
        vc_ratio += 0.1

    # Adjust for weather
    if weather_level == 'Rainy':
        vc_ratio += 0.1
    elif weather_level == 'Snowy':
        vc_ratio += 0.2

    # Determine congestion level
    if vc_ratio <= 0.6:
        return 'Free Flow'
    elif vc_ratio <= 0.9:
        return 'Moderate'
    elif vc_ratio <= 1.2:
        return 'Heavy'
    else:
        return 'Severe'
```

```
# Map string labels to membership functions
total_cars_mf = total_cars.terms
rush_hour_mf = rush_hour.terms
heavy_vehicle_mf = heavy_vehicle_percentage.terms
weather_mf = weather_category.terms
congestion_mf = congestion_level.terms

# Generate rules
for total_cars_level in total_cars_levels:
    for rush_hour_level in rush_hour_levels:
        for hv_level in heavy_vehicle_levels:
            for weather_level in weather_levels:
                congestion = determine_congestion(total_cars_level,
rush_hour_level, hv_level, weather_level)
                rule = ctrl.Rule(
                    total_cars_mf[total_cars_level] &
                    rush_hour_mf[rush_hour_level] &
                    heavy_vehicle_mf[hv_level] &
                    weather_mf[weather_level],
                    congestion_mf[congestion]
                )
                rules.append(rule)

# Create control system
congestion_ctrl = ctrl.ControlSystem(rules)
congestion_simulation = ctrl.ControlSystemSimulation(congestion_ctrl)

# Define weather labels for mapping
weather_labels = {0: 'Clear', 1: 'Cloudy', 2: 'Rainy', 3: 'Snowy'}



# Open a text file for writing
with open('test_case_result.txt', 'w') as result_file:
    # Loop through the test cases
    for i, test_case in enumerate(test_cases, start=1):
        print(f"---\nTest Case {i}: {test_case['description']}")

        # Assign inputs
        congestion_simulation.input['total_cars'] =
test_case['total_cars']
        congestion_simulation.input['rush_hour'] =
test_case['rush_hour']
```

```
        congestion_simulation.input['heavy_vehicle_percentage'] =
test_case['heavy_vehicle_percentage']
        congestion_simulation.input['weather_category'] =
test_case['weather_category']

        # Compute the result
        congestion_simulation.compute()

        # Output the result
        predicted_congestion =
congestion_simulation.output['congestion_level']
        print(f"Predicted Congestion Level (V/C Ratio):
{predicted_congestion:.2f}")

        # Map V/C Ratio to Congestion Level Category
        if predicted_congestion <= 0.6:
            congestion_status = 'Free Flow'
        elif predicted_congestion <= 0.9:
            congestion_status = 'Moderate'
        elif predicted_congestion <= 1.2:
            congestion_status = 'Heavy'
        else:
            congestion_status = 'Severe'

        print(f"Congestion Status: {congestion_status}\n")

        # Write the results to the file
        result_file.write(f"---\nTest Case {i}:
{test_case['description']}\n")
        result_file.write(f"Inputs:\n")
        result_file.write(f"  Total Cars: {test_case['total_cars']}\n")
        result_file.write(f"  Rush Hour: {'Yes' if
test_case['rush_hour'] == 1 else 'No'}\n")
        result_file.write(f"  Heavy Vehicle Percentage:
{test_case['heavy_vehicle_percentage']}%\n")
        result_file.write(f"  Weather Category:
{weather_labels[test_case['weather_category']]}\n")
        result_file.write(f"Predicted Congestion Level (V/C Ratio):
{predicted_congestion:.2f}\n")
        result_file.write(f"Congestion Status:
{congestion_status}\n\n")

        # Visualize and save the output
```

```
        congestion_level.view(sim=congestion_simulation)
        plt.title(f"Test Case {i}: Congestion Level Output")
        plt.savefig(f"test_case_{i}_output.png")
        plt.clf()  # Clear the current figure for the next plot
```