An Expert-Level Report on the Development of an AI-Based System for Public Transport Optimization

Part I: Project Foundation and Context

This report provides a comprehensive framework for a research project aimed at optimizing public transport systems through the application of artificial intelligence. It begins by establishing the project's objectives and situates the proposed work within the existing landscape of academic research and open-source development. Subsequently, it presents a detailed, phased roadmap for the project's execution, from data acquisition to final validation, designed to guide the researcher through the technical and theoretical challenges involved.

Section 1: Abstract of the Proposed Research Project

The foundation of this investigation is encapsulated in the following abstract, which outlines the problem, proposed methodology, and intended contributions of the project.[1]

Title: AI for Optimising Public Transport Routes and Schedules

Abstract: Efficient public transportation is crucial for sustainable urban mobility. However, most city transit systems rely on static routes and fixed schedules, often resulting in underutilization, overcrowding, or increased waiting times. This project presents an AI-based solution to optimize public transport routes and schedules using open-source data and machine learning techniques. The approach involves identifying high-demand areas using clustering algorithms (e.g., K-Means), forecasting passenger flow through time-series models (e.g., ARIMA), and generating improved route suggestions using graph-based optimization. Open datasets such as General Transit Feed Specification (GTFS) and city traffic APIs are used for analysis and testing. The system is designed to adapt to temporal and spatial demand variations, aiming to reduce wait times and improve vehicle allocation.

A prototype simulation demonstrates how dynamic routing and scheduling can enhance operational efficiency without requiring infrastructure changes. This project leverages only open-source tools (Python, Pandas, scikit-learn, NetworkX), making it accessible and cost-effective for academic development. Results highlight the potential of AI in improving public transit planning, especially for cities exploring smart transportation initiatives.

Keywords: Public transport, route optimization, machine learning, GTFS, time-series forecasting, urban mobility, smart cities.

Section 2: A Review of the Academic and Open-Source Landscape

To properly contextualize the proposed research, it is essential to survey the body of existing work in this domain. The application of AI to transportation logistics is a mature and rapidly evolving field. This review examines key academic precedents that validate the project's methodological choices and surveys relevant open-source projects that provide practical implementation benchmarks.

2.1. Academic Precedents in AI-Powered Transit Optimization

The academic literature confirms a significant and sustained effort to leverage artificial intelligence and machine learning for enhancing the efficiency, cost-effectiveness, and service quality of transportation systems.[2] A central theme in this body of work is the transition from static, schedule-based transit operations to dynamic, data-responsive systems capable of adapting to real-time or predicted conditions.[2] The proposed project's three-stage methodology—demand analysis, flow forecasting, and route optimization—is firmly grounded in established research paradigms.

Demand Analysis via Clustering: The use of clustering algorithms to identify patterns in passenger demand is a foundational technique in modern transit planning. Research demonstrates that methods such as K-Means, K-Medoids, and DBSCAN are instrumental in discovering "transport hotspots," delineating demand zones, and understanding the underlying structure of passenger flows.[5] K-Means, as proposed in the abstract, is frequently cited for its effectiveness in this area.[5] Some studies employ hybrid approaches, such as combining Long Short-Term Memory (LSTM) networks with K-Means clustering, to achieve a more sophisticated model of passenger mobility patterns over time.[8]

This body of work validates the initial step of the project, which is to segment the urban environment into zones of distinct demand characteristics.

Passenger Flow Forecasting: The application of time-series models to predict passenger volume is a cornerstone of dynamic transit management. The Autoregressive Integrated Moving Average (ARIMA) model, specified in the project abstract, is a classical and well-documented method for this task.[9] Its strength lies in its ability to model trends, periodicity, and seasonality inherent in historical ridership data, making it a robust tool for short-term forecasting.[9] However, it is also important to acknowledge its limitations. The literature indicates that while ARIMA is effective for capturing linear trends, it often struggles with the complex non-linear variations present in real-world passenger flow, especially over longer prediction horizons.[12] In these scenarios, deep learning models such as LSTMs or Transformers have demonstrated superior performance.[12] For the scope of this project, ARIMA serves as an excellent and scientifically valid baseline model, while more advanced neural network-based models represent a clear and logical direction for future research extensions.

Graph-Based Route Optimization: The representation of a transportation network as a mathematical graph is a standard and powerful abstraction in transportation science.[13] Stops are modeled as nodes (vertices), and the connections between them (transit segments or walking paths) are modeled as edges. Foundational algorithms like Dijkstra's are commonly used to compute the shortest path between an origin and a destination based on a given cost metric, such as travel time or distance.[13] More advanced research builds upon this foundation by developing complex graph models that can simultaneously optimize for multiple criteria, such as minimizing travel time, the number of transfers, and monetary cost.[14] Some research has even developed novel, non-Dijkstra-based algorithms like RAPTOR (Round-bAsed Public Transit Optimized Router) specifically for multi-objective journey planning in dynamic public transit networks.[13] The project's proposed use of the NetworkX library to implement a weighted, directed graph and apply shortest-path algorithms is a practical and effective application of these well-established principles.

The Role of General Transit Feed Specification (GTFS) Data: The GTFS data standard has become the lingua franca for public transit agencies worldwide, enabling unprecedented levels of analysis and application development.[16] GTFS provides a structured format for publishing schedule, route, and stop information, which has powered a generation of trip-planning applications. Researchers and planners are increasingly using this data for more advanced analyses, including network performance evaluation, accessibility studies, and service gap identification.[16] A critical distinction exists within the GTFS ecosystem between static GTFS feeds, which describe the *scheduled* service, and GTFS-Realtime feeds, which provide live updates on vehicle positions, trip updates, and service alerts.[19] The project's reliance on static GTFS data is a key defining characteristic.

It positions the system as a *predictive* optimization tool, where dynamism is achieved by adapting schedules based on *forecasted* demand rather than reacting to live operational data. This distinction is fundamental to understanding the system's capabilities and limitations.

The convergence of these distinct research areas—clustering for demand, time-series for forecasting, and graph theory for optimization—forms a coherent and validated methodological pipeline. The proposed project aligns with this established workflow, indicating that its novelty will arise not from the invention of a new paradigm, but from the specific implementation, the integration of components, the dataset used, and the quantitative evaluation of its results.

2.2. Survey of Relevant Open-Source Projects

The open-source software landscape provides valuable resources, from individual libraries to complete, operational systems, that can inform the project's design and implementation. An analysis of public repositories on platforms like GitHub reveals several projects with goals and architectures that are highly relevant to this work.

Several projects directly mirror the objectives of the proposed system. The sakshi170920/Public-Transport-System-Optimization repository, for instance, outlines a solution for route optimization and dynamic timetable generation, structured into the same core components of route analysis, timetable creation, and a passenger-facing application.[21] Another project, SergeyFilipov/GornaOryahovitsa-TSP-ACO, demonstrates the complete clustering-to-optimization pipeline, using K-Means to identify demand clusters and then applying metaheuristic algorithms like Ant Colony Optimization (ACO) to plan routes, complete with interactive map visualizations.[22]

At a more advanced level, the work of the simovilab research group represents a production-grade, institutional effort to build a comprehensive transit data ecosystem.[24] Their system architecture, comprising Databús for data collection and Infobús for information dissemination, is highly sophisticated. It handles both static GTFS and GTFS-Realtime data, employs a modular microservices approach, and includes a "Stop Times Estimator" service that uses ARIMA and LSTM models for arrival prediction. This project serves as an architectural gold standard, illustrating how the components of the proposed student project could be scaled into a robust, real-world system.Other relevant projects focus on different aspects of the problem. For example, san089/Optimizing-Public-Transportation centers on building a real-time data engineering pipeline using Apache Kafka to process and display live transit status, which points toward a potential future direction for the project involving GTFS-Realtime data.[25]

The broader field of the Vehicle Routing Problem (VRP) also offers powerful libraries like Timefold Solver, which provide general-purpose AI constraint solvers that could be adapted for more complex, multi-objective transit optimization tasks in the future.[26]

To systematically evaluate these precedents, the following table provides a comparative analysis of the most relevant open-source projects. This analysis helps to position the proposed work, learn from existing implementations, and identify unique contributions.

Table 1: Comparative Analysis of Key GitHub Repositories

Repository	**Primary Objective**	**Key Algorithms/Methods**	**Data Sources**	**Technology Stack**	**Relevance to Project**
sakshi170920/Public-Transport-System-Optimization [21]	End-to-end optimization of routes and timetables.	Genetic algorithms (mentioned in code).	Not specified (likely static traffic/population data).	Jupyter Notebook, Python, Dart.	**High.** Direct conceptual parallel. Demonstrates a project-based approach.
simovilab/ [24]	Comprehensive transit data management and analysis platform.	ARIMA, LSTM for arrival prediction; Heuristic search.	GTFS, GTFS-Realtime.	Python, Django, PostgreSQL/PostGIS, REST/GraphQL.	**Very High.** Represents a production-grade, modular architecture. The "Stop Times Estimator" is directly relevant.
SergeyFilipov/GornaOryahovitsa-TSP-ACO [22]	Bus route optimization for a specific city.	K-Means, Ant Colony Optimization (ACO), Greedy TSP.	OpenRouteService API.	Python, K-Means, Folium.	**High.** Excellent example of the clustering-to-optimization pipeline with visualization.
san089/Optimizing-Public-Transportation [25]	Real-time event pipeline for transit status display.	Kafka streaming, REST Proxy.	Chicago Transit Authority public data.	Python, Apache Kafka, Docker, PostgreSQL.	**Medium.** Focuses on real-time data engineering, which is a potential future direction for the user's project (using GTFS-RT).

This survey reveals a critical distinction that frames the entire project: the chasm between systems based on static, scheduled data and those that are truly dynamic and reactive. The project, as defined by its abstract and reliance on static GTFS, falls into the former category.

Its "dynamism" is predictive; it optimizes routes and schedules for a future time period based on a forecast of demand for that period. This is fundamentally different from a system that ingests a GTFS-Realtime feed and adjusts operations on a minute-by-minute basis in response to live vehicle locations and traffic conditions. This distinction must be clearly articulated in the project's scope and limitations. It defines the boundary of what the system can achieve and manages expectations about its real-world applicability. The proposed system is a strategic planning tool, not a real-time operational control system.

Part II: A Strategic Roadmap for Project Execution

This section presents a detailed, five-phase roadmap for the successful execution of the research project. Each phase is broken down into specific tasks, with recommendations for tools and methodologies grounded in the academic and open-source precedents reviewed above. This roadmap is designed to serve as a practical guide for research and development.

Section 3: Phase 1 - Data Acquisition, Preprocessing, and Exploratory Analysis

This foundational phase is the most critical component of the project. The validity and performance of all subsequent machine learning and optimization models are contingent upon the quality, cleanliness, and integrity of the data prepared here. Errors or logical inconsistencies introduced at this stage will propagate throughout the entire system, rendering the final results unreliable.

3.1. Sourcing and Ingesting Transit Data

The initial task is to acquire the necessary datasets that will fuel the analysis. The project requires two primary types of data: public transit schedule data and supplementary traffic information.

Primary Data Source (GTFS): The core of the analysis will be a static General Transit Feed Specification (GTFS) dataset. These are typically provided by municipal or regional transit authorities. Excellent sources for finding these feeds include centralized repositories like OpenMobilityData or direct downloads from an agency's developer portal.[27] A GTFS feed is distributed as a single.zip archive containing a collection of comma-separated value (.txt) files.

Key files for this project will include stops.txt (geographic locations of stops), routes.txt (descriptions of transit routes), trips.txt (individual journeys along a route), and stop_times.txt (the sequence of stops and scheduled arrival/departure times for each trip).[27]

Data Ingestion: The data should be ingested into the Python environment using the pandas library. It is efficient to read the.txt files directly from the.zip archive using Python's built-in zipfile module, which avoids the need for manual extraction.[30] During the data loading process, it is a critical best practice to explicitly define the data type for each column (e.g., string for IDs, float for coordinates). This practice prevents pandas from making incorrect type inferences, which can lead to silent errors during subsequent data merging and analysis operations.[30]

Supplementary Data (Traffic APIs): Static GTFS data provides scheduled travel times but contains no information about real-world traffic conditions. To create a more realistic model, especially for weighting the network graph in Phase 4, it is necessary to incorporate historical or typical traffic data. This can be obtained from a public API, such as the Google Maps Directions API or the OpenRouteService API.[22] The goal is to query these services for typical travel times between key points in the transit network during different times of the day (e.g., peak vs. off-peak hours).

3.2. Data Cleaning and Preprocessing

Raw data is rarely perfect. This sub-phase involves a rigorous process of validation, cleaning, and transformation to prepare the data for analysis.

GTFS Data Validation: Before proceeding, the integrity of the GTFS feed must be validated. Specialized open-source libraries such as gtfs-kit are designed for this purpose.[31] These tools perform a series of checks for common issues, including referential integrity errors (e.g., a trip_id in stop_times.txt that does not exist in trips.txt), logical impossibilities (e.g., a departure time that is earlier than an arrival time), and formatting violations. Identifying and correcting these issues early is essential.

Handling Missing Data: Missing values are a common problem. For instance, the stop_times.txt file may contain blank entries for arrival_time or departure_time for the first or last stop of a trip. These can often be filled through linear interpolation based on the travel times between other stops on the same trip.[32] Missing values in other critical fields, such as the geographic coordinates in stops.txt, may require either imputation based on nearby stops or the removal of the incomplete record.

Time and Date Conversion: The time and date information in GTFS files must be parsed and converted into standardized formats suitable for computation. Time fields, often stored as strings (e.g., "08:30:00"), should be converted into pandas timedelta objects to allow for arithmetic operations. The service patterns, which define on which days a particular trip operates, are determined by a combination of the calendar.txt file (for recurring weekly schedules) and the calendar_dates.txt file (for exceptions like holidays or special services).

A robust parsing logic must be implemented to correctly determine the set of active service_ids for any given date of analysis.[30]

Geospatial Processing: To perform spatial analysis, the latitude and longitude columns from stops.txt must be converted into geospatial point objects. The geopandas library is the standard tool for this in Python. This conversion enables powerful spatial operations, such as calculating distances between stops, performing spatial joins, and visualizing the network. Furthermore, the shapes.txt file, which contains a sequence of points that trace the physical path of a route, can be processed to create LineString geometries for each unique route shape. This can be accomplished by grouping the points by shape_id and applying a function that constructs a line from the ordered sequence of points.[29]

The relational complexity of the GTFS standard cannot be overstated. A single conceptual entity, like a bus journey, is described across multiple tables linked by IDs. For example, to fully understand a trip, one must join routes.txt (to get the route name, like "Route 55"), trips.txt (to get a specific instance of that route, like the 8:00 AM weekday trip), and stop_times.txt (to get the sequence of stops for that specific trip). The failure to correctly implement this relational logic will result in a fundamentally flawed representation of the transit network, invalidating all subsequent analysis. This data preparation stage, therefore, requires meticulous attention to detail and rigorous validation checks.

3.3. Exploratory Data Analysis (EDA)

Before applying complex algorithms, it is crucial to develop an intuitive understanding of the dataset. EDA involves summarizing the main characteristics of the data, often with visual methods.

System-Level Analysis: Begin by computing basic descriptive statistics for the entire transit system: the total number of agencies, routes, stops, and trips per day.[33] This provides a high-level sense of the network's scale and complexity.

Temporal Analysis: Analyze the temporal patterns of service. By aggregating the number of trips from stop_times.txt by hour of the day and day of the week, one can generate plots that clearly visualize the system's operational rhythm, highlighting peak and off-peak periods.

Spatial Analysis: Visualize the spatial distribution of the transit infrastructure. A simple scatter plot of all stop locations from stops.txt on a map provides an immediate overview of the network's geographic coverage.[33] To gain deeper insight, a kernel density estimation can be applied to these points to create a heatmap. This heatmap will reveal areas with a high concentration of transit stops, offering a preliminary, visual proxy for high-service corridors and hubs.[18]

Route Analysis: Examine the characteristics of individual routes. Plotting the geographic path of specific routes helps in understanding how they connect different parts of the city. Further analysis can include calculating the distribution of route lengths and the number of stops per route to identify outliers (e.g., very long "express" routes vs. short "circulator" routes).

This initial phase culminates in a clean, validated, and well-understood dataset. The insights gained from EDA will inform the feature engineering and modeling decisions in the subsequent phases. The following table outlines a plan for transforming the raw data into features suitable for the project's machine learning models.

Table 2: Data Sources and Feature Engineering Plan

Raw Data Source	**Raw Columns**	**Engineered Feature**	**Description**	**Target Model**
stops.txt	stop_lat, stop_lon	Stop Coordinates	Geographic location of each stop.	K-Means
stop_times.txt	trip_id	Trip Frequency per Stop	Count of trips servicing a stop per day/hour.	K-Means
stop_times.txt	arrival_time	Passenger Flow Proxy	Aggregated count of arrivals at a stop/zone over time.	ARIMA
Traffic API	N/A	Average Travel Time	Historical average travel time between key points.	NetworkX Graph
trips.txt, routes.txt	route_id, trip_id	Route Density	Number of unique routes serving a stop/zone.	K-Means

Section 4: Phase 2 - Demand Pattern Identification via Clustering

With a clean dataset, the next phase is to apply unsupervised machine learning to identify underlying patterns of transit demand. The objective is to move beyond individual stop locations and group them into meaningful zones or clusters that represent areas with similar service characteristics. These clusters will form the basis for demand forecasting and route optimization.

4.1. Feature Engineering and Scaling

The performance of a clustering algorithm is highly dependent on the features provided as input. The selection and preparation of these features are therefore critical steps.

Feature Selection: The primary features for clustering will be the geospatial coordinates of the transit stops (latitude and longitude).[34] Using only these two features would result in clusters based purely on geographic proximity. To create more meaningful, transit-oriented clusters, it is essential to engineer and include additional features that describe the level of service at each stop. Two powerful features to consider are:

- **Service Frequency:** For each stop, calculate the total number of trips that service it within a specific time window of interest (e.g., the morning peak period, 7:00 AM to 9:00 AM). This can be derived by counting the relevant entries in the stop_times.txt table.
- **Route Diversity:** For each stop, count the number of unique route_ids that pass through it. A stop served by five different routes is functionally more significant than a stop served by only one, even if their total trip frequencies are similar.

Data Scaling: The K-Means algorithm is based on calculating Euclidean distances between points in the feature space. If the input features are on vastly different scales (e.g., latitude ranging from -90 to 90, and service frequency ranging from 1 to 500), the feature with the larger scale will dominate the distance calculation, leading to skewed and uninformative clusters.[34] To prevent this, all features must be scaled to a common range before being fed into the algorithm. The StandardScaler (which scales data to have a mean of 0 and a standard deviation of 1) or the MinMaxScaler (which scales data to a range, typically) from the scikit-learn library are standard tools for this task.

4.2. K-Means Model Implementation

The core of this phase is the application of the K-Means clustering algorithm.

Algorithm: The KMeans implementation provided in the scikit-learn library is a robust and efficient choice.[36] The algorithm works by iteratively assigning each data point (in this case, each transit stop) to one of k clusters and then updating the center (centroid) of each cluster to be the mean of the points assigned to it. This process repeats until the cluster assignments stabilize.[37]

Determining the Optimal Number of Clusters (k): The number of clusters, k, is the most important hyperparameter for the K-Means algorithm and must be specified by the user. Since there is no single "correct" value for k, its selection must be guided by empirical methods and domain knowledge. Two common techniques are:

- **The Elbow Method:** This involves running the K-Means algorithm for a range of k values (e.g., from 2 to 20) and plotting the within-cluster sum of squares (WCSS), also known as inertia. The WCSS measures the compactness of the clusters. As k increases, the WCSS will always decrease. The plot of WCSS against k typically forms an "elbow" shape. The point of inflection on this curve, where the rate of decrease in WCSS slows down significantly, is often considered a good candidate for the optimal k.[5]
- **Silhouette Analysis:** This method provides a more quantitative measure of cluster quality. The silhouette score for a data point measures how similar it is to its own cluster compared to other clusters. The score ranges from -1 to 1, where a high value indicates that the point is well-matched to its own cluster and poorly matched to neighboring clusters. By calculating the average silhouette score for all points for different values of k, one can choose the k that maximizes this score, indicating the most distinct and well-separated clustering.[8]

Model Fitting: Once an appropriate value for k has been determined, the KMeans model is instantiated with this parameter and fitted to the scaled feature data.[34] The output of the fit method is an array of cluster labels, where each transit stop is assigned an integer label corresponding to its cluster.

4.3. Cluster Interpretation and Visualization

The raw output of the clustering algorithm—a set of numerical labels—is not inherently meaningful. The final step in this phase is to interpret these clusters and translate them into actionable insights.

Spatial Visualization: The most intuitive way to understand the clusters is to visualize them spatially. This is done by creating a scatter plot of the transit stops on a map, with the color of each point determined by its assigned cluster label.[35] This map will immediately reveal the geographic distribution of the identified demand zones.

Statistical Analysis: To understand *why* the algorithm created these specific groupings, it is necessary to analyze the statistical properties of each cluster. Using the groupby() function in pandas, one can calculate the mean (or other summary statistics) of the original input features for each cluster.[38] For example, one might find that Cluster 0 has a very high average service frequency and route diversity, corresponding to the central business district, while Cluster 3 has low values for these features, corresponding to a suburban residential area. This analysis allows for the creation of descriptive personas for each cluster (e.g., "downtown high-frequency hub," "university campus zone," "suburban commuter zone").

It is important to recognize that by including service-level features like frequency and route diversity alongside geographic coordinates, the resulting clusters represent more than just simple geographic proximity. They represent areas of similar *transit service levels*. The K-Means algorithm, by minimizing distance in this multi-dimensional feature space, groups stops that are not only close to each other geographically but also share similar operational characteristics. Two stops that are physically adjacent could be assigned to different clusters if one is a major transfer hub served by twenty routes and the other is a minor local stop served by only one. The output is therefore not merely a geographic partitioning of the city, but a "transit service zoning." This is a much more powerful and nuanced input for the subsequent forecasting and optimization phases, as it directly reflects the existing structure and intensity of the public transport network.

Section 5: Phase 3 - Passenger Flow Forecasting with Time-Series Models

After identifying spatial demand patterns through clustering, this phase addresses the temporal dimension of demand. The objective is to build a time-series model that can forecast future passenger flow for each of the identified high-demand zones. These forecasts will be the key input for dynamically adjusting routes and schedules in the optimization phase.

5.1. Time-Series Data Preparation

The first step is to construct a suitable time-series dataset from the processed GTFS data.

Data Aggregation: A time series requires a sequence of observations at regular time intervals. In this context, a reliable proxy for passenger flow can be created by counting the number of scheduled vehicle arrivals within each cluster for each time interval. For example, the data can be aggregated to count the total number of arrivals across all stops within Cluster 0 every 15 minutes throughout a typical weekday. This process results in a separate time-series dataset for each demand cluster identified in Phase 2.

Stationarity Testing: A core assumption of the ARIMA model is that the underlying time series is stationary. A stationary series is one whose statistical properties—such as its mean, variance, and autocorrelation—are constant over time.[9] Real-world ridership data is often non-stationary, exhibiting clear trends (e.g., long-term ridership growth) and seasonality (e.g., daily peaks, weekly cycles).

- **Visual Inspection:** The first check for stationarity is to simply plot the time series. Visual inspection can often reveal obvious trends (a consistent upward or downward slope) or seasonal patterns.

- **Statistical Tests:** For a more rigorous assessment, a statistical test for stationarity should be performed. The Augmented Dickey-Fuller (ADF) test is a standard method for this purpose.[9] The null hypothesis of the ADF test is that the time series is non-stationary. Therefore, if the test yields a high p-value (typically > 0.05), the null hypothesis cannot be rejected, and the series is considered non-stationary.

Differencing: If the time series is found to be non-stationary, it must be transformed to become stationary before it can be modeled with ARIMA. The most common method for this transformation is differencing, which involves subtracting the previous observation from the current observation.[39] If the series still exhibits non-stationarity after one round of differencing, the process can be repeated on the already-differenced series (second-order differencing). The number of differencing operations required to make the series stationary determines the d (for Integrated) parameter in the ARIMA(p, d, q) model.

5.2. ARIMA Model Identification and Fitting

Once a stationary time series has been prepared, the next step is to identify the appropriate parameters for the ARIMA model and fit it to the data.

The ARIMA Model: The ARIMA(p, d, q) model is a powerful class of statistical models for analyzing and forecasting time-series data. It is a composite model that combines three components [11]:

- **Autoregression (AR - parameter p):** This component models the relationship between an observation and a number of lagged observations (i.e., observations from previous time steps). The parameter p defines how many past observations are included in the model.
- **Integrated (I - parameter d):** This component represents the use of differencing to make the time series stationary, as described above. The parameter d is the order of differencing used.
- **Moving Average (MA - parameter q):** This component models the relationship between an observation and the residual errors from a moving average model applied to lagged observations. The parameter q defines the size of the moving average window.

Determining p and q: The parameters p and q are identified by inspecting the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots of the stationary time series.

- **ACF Plot:** This plot shows the correlation of the time series with its own lagged values. It is used to identify the order of the MA component (q). A sharp cut-off in the ACF plot after a certain number of lags suggests a potential value for q.[40]
- **PACF Plot:** This plot shows the correlation between an observation and its lagged values, after removing the effects of the intermediate lags.

- It is used to identify the order of the AR component (p). A sharp cut-off in the PACF plot suggests a potential value for p.[40]

Model Fitting: With the parameters (p, d, q) identified, the model can be fitted to the training portion of the time-series data. The statsmodels library in Python provides a comprehensive and easy-to-use ARIMA class for this purpose.[41]

5.3. Forecasting and Model Evaluation

The final step in this phase is to use the fitted model to generate forecasts and to rigorously evaluate its performance.

Forecasting: The fitted statsmodels ARIMA object has methods such as forecast() or get_prediction() that can be used to predict future values of the time series.[11] These methods will generate a forecast for a specified number of future time steps.

Evaluation: The accuracy of the model must be assessed by comparing its forecasts against a held-out test set of actual data.

- **Performance Metrics:** Standard regression metrics are used to quantify the model's error. The Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) are common choices. The RMSE is often preferred as it is in the same units as the original data, making it more interpretable.[11]
- **Model Diagnostics:** The summary output of the fitted statsmodels model provides valuable diagnostic information. The Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) are measures of model fit that penalize model complexity. These values are useful for comparing different ARIMA(p, d, q) configurations on the same dataset; lower values generally indicate a better model.[39] Additionally, it is crucial to examine the residuals (the errors) of the model. For a good model, the residuals should be uncorrelated and resemble white noise. A plot of the residuals' ACF should show no significant correlations.[9]

A key design decision in this phase is the choice of the time interval for the series (e.g., 15 minutes vs. 1 hour). This choice has a direct causal impact on the potential responsiveness of the final optimized system. The ARIMA model produces one forecast value for each time interval. This forecast then becomes an input to the graph optimization model in the next phase. If the forecast is generated on an hourly basis, the "optimized" routes can only be updated once per hour. In contrast, a 15-minute forecast interval allows for the system to adapt its route recommendations four times per hour. This creates a fundamental trade-off: shorter time intervals enable more dynamic and responsive scheduling but may lead to a noisier time-series signal, potentially reducing forecast accuracy.

Furthermore, a higher forecast frequency necessitates more frequent re-computation of the optimal routes, increasing the computational burden. This choice of temporal granularity is a critical design parameter that must be carefully considered and justified in the final project report.

Section 6: Phase 4 - Network Modeling and Route Optimization

This phase is where the spatial and temporal analyses from the preceding phases converge. The public transport system is translated into a mathematical graph structure, and algorithms are applied to this graph to find optimal routes. The novelty of the approach lies in dynamically weighting the graph based on the demand forecasts, allowing the optimization process to adapt to changing conditions.

6.1. Constructing the Transit Network Graph

The first step is to create a formal mathematical representation of the transit network. The NetworkX library in Python is the ideal tool for this task, as it provides a rich set of data structures and algorithms for graph analysis.[44]

Graph Representation: A directed graph (DiGraph) is the most appropriate structure, as travel between two stops is often not symmetric (e.g., due to one-way streets).

- **Nodes:** Each unique public transport stop, identified by its stop_id from the stops.txt file, will be represented as a node in the graph.
- **Edges:** Edges represent direct travel links between nodes. Two types of edges must be created:
    1. **Transit Edges:** A directed edge exists from stop A to stop B if there is at least one transit trip that travels directly from A to B as consecutive stops. This information is derived from the stop_times.txt file by examining records with consecutive stop_sequence numbers for the same trip_id.
    2. **Walking Transfer Edges:** To model the ability of passengers to transfer between nearby stops or different routes, it is essential to add "walking" edges to the graph. For each pair of stops, if the geographic distance between them is below a reasonable threshold (e.g., 400 meters, representing a 5-minute walk), a pair of directed edges should be added between them (one in each direction).

6.2. Assigning Dynamic Edge Weights

The "cost" of traversing each edge in the graph is represented by its weight. A simple model might use only the scheduled travel time as the weight. However, to achieve dynamic optimization, the edge weights must be a function of both travel time and predicted demand.

Base Cost (Travel Time): The foundational component of the edge weight is the travel time.

- For **transit edges**, this is the scheduled travel time between the two stops, calculated as

the difference between the arrival_time at the destination stop and the departure_time at the origin stop from the stop_times.txt file.
- For **walking transfer edges**, the weight is the estimated walking time, calculated by dividing the distance between the stops by an average walking speed (e.g., 1.4 meters/second).

Dynamic Cost (Demand-Based): This is the core innovative element of the model. The base travel time is modified by a penalty term that is proportional to the forecasted passenger flow in the destination node's cluster. This simulates the effects of congestion and crowding, making travel into high-demand areas "costlier" from an optimization perspective. A simple but effective formula for the dynamic weight could be:

$w_{dynamic} = w_{time} \times (1 + \alpha \times F_{c,t})$

where:

- $w_{dynamic}$ is the final dynamic weight of the edge.
- w_{time} is the base travel time cost.
- $F_{c,t}$ is the forecasted passenger flow for the cluster c (to which the destination node belongs) at time t, obtained from the ARIMA model.
- α is a tunable hyperparameter that controls the sensitivity of the model to demand. A higher value of α means that forecasted demand will have a stronger influence on the routing decisions.

6.3. Route Optimization Using Graph Algorithms

With the dynamically weighted graph constructed, the final step is to use a shortest-path algorithm to generate optimized route suggestions.

Algorithm: Dijkstra's algorithm is a classic and efficient algorithm for finding the single-source shortest path in a weighted graph with non-negative edge weights. NetworkX provides highly optimized implementations of this algorithm, such as nx.single_source_dijkstra (which finds paths from a source to all other nodes) and nx.dijkstra_path (which finds the path between a specific source and target).[46]

Generating Route Suggestions: The optimization process follows a clear workflow:

1. **Define Optimization Scenario:** Select a set of key origin-destination (OD) pairs for which to generate optimized routes. These could be, for example, pairs of centroids of high-demand clusters identified in Phase 2.
2. **Update Graph Weights:** For a specific time of day (e.g., 8:00 AM), retrieve the corresponding passenger flow forecasts for all clusters from the ARIMA models. Use these forecasts to calculate and update the dynamic_weight for every edge in the NetworkX graph.

3. **Compute Shortest Path:** For each OD pair, run Dijkstra's algorithm on the newly weighted graph. The algorithm will find the path that minimizes the sum of the dynamic edge weights.
4. **Output Optimized Route:** The output of the algorithm is a sequence of stop IDs representing the optimal path. This sequence constitutes a new, dynamically generated route suggestion that is tailored to the predicted demand conditions for that specific time of day.

It is important to acknowledge that this approach simplifies the complex, multi-faceted nature of real-world transit planning. The project defines "optimality" as finding the shortest path according to a single, composite cost function. In practice, transit network design is a multi-objective optimization problem, where planners must simultaneously balance numerous, often competing, goals. These include minimizing passenger journey times, minimizing agency operational costs (which are related to total vehicle-kilometers traveled), maximizing service coverage to ensure equitable access for all communities, and minimizing environmental impact.[4] The single-objective Dijkstra-based approach is a valid and powerful method for an academic prototype. However, a comprehensive understanding of the problem requires recognizing this simplification. Future extensions of this work could reframe the problem as a multi-objective one, potentially employing different algorithms like A* search or metaheuristics (e.g., genetic algorithms, ant colony optimization) to explore the trade-offs between these different objectives and find a set of Pareto-optimal solutions.[4]

Section 7: Phase 5 - System Integration, Simulation, and Validation

The final phase of the project involves integrating the individual components into a cohesive system, simulating its performance, and rigorously validating its effectiveness against the existing static transit system. This phase is crucial for demonstrating the tangible benefits of the proposed AI-based approach.

7.1. Prototype System Integration

To create a functional prototype, the workflows from the previous phases must be orchestrated into a single, executable pipeline. This can be accomplished by developing a master Python script that calls the various components in the correct sequence.

Integrated Workflow:

1. **Data Loading:** The script begins by loading and preprocessing the GTFS data as detailed in Phase 1.
2. **Clustering:** It then runs the K-Means clustering algorithm on the stop data to define the city's demand zones.
3. **Forecasting:** For a user-specified target time period (e.g., "tomorrow from 8:00 AM to 9:00 AM"), the script loads the pre-trained ARIMA models for each cluster and generates

passenger flow forecasts.

4. **Graph Construction:** The script constructs the NetworkX graph of the transit network.
5. **Dynamic Weighting:** It updates the edge weights in the graph using the forecasts generated in the previous step, applying the dynamic cost function.
6. **Optimization:** Finally, it runs the shortest-path algorithm for pre-defined origin-destination pairs to generate a set of new, optimized route suggestions for the target time period.

7.2. Simulation Environment

To quantitatively evaluate the performance of the dynamically generated routes, it is necessary to compare them against the baseline (the existing static routes) in a controlled simulation environment. While sophisticated, open-source traffic simulators like CityFlow exist, they may be overly complex for the scope of this project.[52] A simplified, custom-built agent-based simulation in Python can be highly effective for this purpose.

Simplified Agent-Based Model:

1. **Demand Generation:** Define a population of "passenger agents." Each agent is assigned an origin and a destination stop. The distribution of these origins and destinations should be based on the demand patterns identified in Phase 2 (e.g., more agents should originate from clusters identified as high-demand residential zones during the morning peak).
2. **Scenario Simulation:** Run two parallel simulations for the same population of agents:
    - **Scenario A (Baseline):** Agents travel using the existing static routes as defined in the GTFS data.
    - **Scenario B (Optimized):** Agents travel using the new, dynamically generated routes produced by the optimization pipeline.
3. **Travel Time Calculation:** For each agent in both scenarios, the simulation must calculate their total journey time. This is the sum of their initial **wait time** (the time from their arrival at the origin stop until a vehicle arrives) and their **in-vehicle time** (the time spent traveling on the transit vehicles).

7.3. Validation and Performance Evaluation

The final step is to analyze the results of the simulation and quantify the impact of the optimization system. This requires defining a set of clear, measurable Key Performance Indicators (KPIs) that capture both the passenger experience and the operational efficiency of the system. The abstract claims the system will "reduce wait times and improve vehicle allocation," and these claims must be substantiated with data from the simulation.[1]

The following table outlines a structured set of evaluation metrics to compare the baseline and optimized scenarios. This framework provides a rigorous, data-driven basis for assessing the project's success.

Table 3: Model Evaluation Metrics

Category	**Metric**	**Description**	**Baseline (Static Routes)**	**Optimized (Dynamic Routes)**
Passenger Experience	Average Wait Time	The average time a passenger waits for a vehicle.	Calculated from simulation.	Calculated from simulation.
	Average Journey Time	The average total time from origin to destination for a passenger.	Calculated from simulation.	Calculated from simulation.
Operational Efficiency	Vehicle Utilization	The average number of passengers per vehicle during a trip.	Calculated from simulation.	Calculated from simulation.
	Total Vehicle Kilometers	The total distance traveled by all vehicles in the system.	Calculated from GTFS data.	Calculated from simulation.
	Service Coverage	Percentage of demand zones adequately served.	Assessed via EDA.	Assessed via simulation.

By comparing the values in the "Baseline" and "Optimized" columns for each KPI, a clear, quantitative assessment of the system's performance can be made. For example, a significant reduction in Average Wait Time and Average Journey Time would provide strong evidence that the system improves the passenger experience. Similarly, an increase in Vehicle Utilization would demonstrate more efficient use of transit resources. The results of this validation process will form the core of the project's conclusions.

Part III: Concluding Remarks and Future Directions

This final part of the report synthesizes the project's contributions, acknowledges its inherent limitations, and proposes a clear path forward for future research and development.

Section 8: Synthesis and Recommendations for Future Work

8.1. Summary of Contributions

This project successfully demonstrates the design and implementation of an end-to-end, open-source prototype for the AI-based optimization of public transport routes and schedules. The primary contribution is the successful integration of three distinct machine learning and algorithmic paradigms into a cohesive workflow:

1. **Spatiotemporal Demand Analysis:** Unsupervised clustering (K-Means) was effectively used to segment the urban environment into meaningful zones based on both geographic location and transit service levels.
2. **Predictive Forecasting:** Time-series analysis (ARIMA) was employed to forecast passenger flow dynamics for these zones, introducing a predictive temporal dimension to the model.
3. **Algorithmic Optimization:** Graph theory (NetworkX) and shortest-path algorithms (Dijkstra's) were used to translate the transit network into a mathematical model and generate optimized route suggestions based on dynamically updated, demand-sensitive costs.

The project culminates in a simulation framework that allows for the quantitative validation of the proposed system against the existing static network, using a clear set of performance indicators. The entire system is built on an accessible and cost-effective stack of open-source Python libraries, making the methodology reproducible and extensible for future academic and practical applications.

8.2. Limitations of the Current Approach

A rigorous scientific inquiry requires a transparent acknowledgment of the study's limitations. The current approach, while effective as a proof-of-concept, has several key constraints that bound its real-world applicability:

- **Reliance on Static Data:** The system's foundation is the static GTFS feed, which represents the *scheduled* plan, not the operational reality. As such, the optimization is predictive, not reactive. It cannot respond to real-time events such as traffic jams, vehicle breakdowns, or unexpected surges in demand.
- **Baseline Forecasting Model:** The use of ARIMA as the forecasting model, while a valid and common approach, is limited in its ability to capture complex, non-linear patterns in passenger demand. More advanced phenomena, such as the impact of special events or weather, are not easily modeled by this method.
- **Single-Objective Optimization:** The route optimization process is simplified to a single-objective problem: minimizing a composite cost of travel time and predicted congestion. This does not capture the multi-objective nature of real-world transit planning, which must balance passenger convenience, operational cost, service equity, and environmental factors.

8.3. Avenues for Future Research

These limitations directly point to several promising avenues for future research that could build upon the foundation established by this project.

Integration of Real-Time Data: The most impactful extension would be the incorporation of a GTFS-Realtime data feed.[19] This would provide live data on vehicle positions, trip delays, and service alerts. Integrating this data stream would allow the system to transition from a purely predictive planning tool to a truly dynamic, reactive operational control system. The optimization algorithms could then be re-run at high frequency to adjust routes and schedules in response to on-the-ground conditions as they unfold.

Advanced Predictive Models: The forecasting component could be significantly enhanced by replacing or augmenting the ARIMA models with more sophisticated deep learning techniques. Models like Long Short-Term Memory (LSTM) networks or Transformers are specifically designed to learn complex, non-linear patterns and long-term dependencies in time-series data, and have been shown to outperform traditional statistical models in many transportation forecasting tasks.[12]

Multi-Objective Optimization: A more realistic model of the routing problem would frame it as a multi-objective optimization task. Future work could explore the use of metaheuristic algorithms, such as genetic algorithms or ant colony optimization, to solve this more complex problem.[4] Instead of producing a single "optimal" route, these methods can generate a set of Pareto-optimal solutions, presenting planners with a range of choices that represent different trade-offs between competing objectives (e.g., a route that is fastest for passengers vs. a route that is cheapest for the agency to operate).

Reinforcement Learning for Scheduling: For the problem of dynamic schedule optimization (e.g., deciding when to dispatch the next vehicle or adjust headways), reinforcement learning (RL) offers a powerful paradigm. An RL agent could be trained in a simulated environment to learn a policy for making real-time dispatching decisions that maximizes a reward function combining metrics like low passenger wait times and high vehicle utilization.[51] This would represent a significant step towards a fully autonomous and adaptive transit management system.

Works cited

1. Format of Draft Project Abstract.pdf
2. (PDF) AI-Powered Route Optimization Reducing Costs and Improving Delivery Efficiency, accessed on October 16, 2025, https://www.researchgate.net/publication/389987796_AI-Powered_Route_Optimization_Reducing_Costs_and_Improving_Delivery_Efficiency
3. Artificial Intelligence in Logistics and Distribution: The function of AI in dynamic route planning for transportation, including self-driving trucks and drone delivery systems, accessed on October 16, 2025, https://journalwjarr.com/sites/default/files/fulltext_pdf/WJARR-2025-0214.pdf
4. Enhancing Route Optimization in Road Transport Systems Through Machine Learning: A Case Study of the Dakhla-Paris Corridor - MDPI, accessed on October 16, 2025, https://www.mdpi.com/2673-7590/5/2/60
5. (PDF) K-means clustering method in organizing passenger ..., accessed on October 16, 2025, https://www.researchgate.net/publication/395639504_K-means_clustering_method_in_organizing_passenger_transportation_in_a_smart_city
6. Yurii Matseliukh†, Vasyl Lytvyn† and Myroslava Bublyk - CEUR-WS.org, accessed on October 16, 2025, https://ceur-ws.org/Vol-3983/paper17.pdf
7. Application of the K-means clustering method in the organisation of passenger transport in a smart city | INNOVATIVE TECHNOLOGIES AND SCIENTIFIC SOLUTIONS FOR INDUSTRIES, accessed on October 16, 2025, https://journals.uran.ua/itssi/article/view/328662
8. LSTM Forecasting and K-Means Clustering for Passenger Mobility Management at Bus Terminals | Journal of Information Systems and Informatics, accessed on October 16, 2025, https://journal-isi.org/index.php/isi/article/view/1159
9. (PDF) Short-term Passenger Flow Prediction of Urban Rail Transit ..., accessed on October 16, 2025, https://www.researchgate.net/publication/377744826_Short-term_Passenger_Flow_Prediction_of_Urban_Rail_Transit_based_on_ARIMA_Model
10. 12 XII December 2024 https://doi.org/10.22214/ijraset.2024.65723, accessed on October 16, 2025, https://www.ijraset.com/best-journal/a-dynamic-r-user-interface-utilizing-arima-model-for-visualizing-passenger-volume-forecast-for-improved-public-decision-making
11. A Guide to Time Series Forecasting with ARIMA in Python 3 | DigitalOcean, accessed on October 16, 2025, https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3
12. Long-Term Passenger Flow Forecasting for Rail Transit Based on Complex

Networks and Informer - MDPI, accessed on October 16, 2025, https://www.mdpi.com/1424-8220/24/21/6894
13. Round-Based Public Transit Routing - Microsoft, accessed on October 16, 2025, https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/raptor_alenex.pdf
14. A Graph Theoretical Model of Public Transportation Network for Efficient Trip Planning, accessed on October 16, 2025, https://www.researchgate.net/publication/335436845_Public_Transportation_Graph_A_Graph_Theoretical_Model_of_Public_Transportation_Network_for_Efficient_Trip_Planning
15. Optimal Routing in Urban Road Networks: A Graph-Based Approach Using Dijkstra's Algorithm - MDPI, accessed on October 16, 2025, https://www.mdpi.com/2076-3417/15/8/4162
16. Innovative GTFS Data Application for Transit Network Analysis Using a Graph-Oriented Method - Digital Commons @ USF - University of South Florida, accessed on October 16, 2025, https://digitalcommons.usf.edu/cgi/viewcontent.cgi?article=1541&context=jpt
17. Leveraging the General Transit Feed Specification for Efficient Transit Analysis - ResearchGate, accessed on October 16, 2025, https://www.researchgate.net/publication/269807772_Leveraging_the_General_Transit_Feed_Specification_for_Efficient_Transit_Analysis
18. THE MANY USES OF GTFS DATA OPENING THE DOOR TO TRANSIT AND MULTIMODAL APPLICATIONS | Request PDF - ResearchGate, accessed on October 16, 2025, https://www.researchgate.net/publication/268097444_THE_MANY_USES_OF_GTFS_DATA_OPENING_THE_DOOR_TO_TRANSIT_AND_MULTIMODAL_APPLICATIONS
19. From Raw GPS to GTFS: A Real-World Open Dataset for Bus Travel Time Prediction - MDPI, accessed on October 16, 2025, https://www.mdpi.com/2306-5729/10/8/119
20. GTFS-realtime Tutorials - Google Groups, accessed on October 16, 2025, https://groups.google.com/g/gtfs-realtime/c/bDcPo675Cjw
21. sakshi170920/Public-Transport-System-Optimization - GitHub, accessed on October 16, 2025, https://github.com/sakshi170920/Public-Transport-System-Optimization
22. transport-optimization · GitHub Topics, accessed on October 16, 2025, https://github.com/topics/transport-optimization
23. route-planning · GitHub Topics, accessed on October 16, 2025, https://github.com/topics/route-planning?o=desc&s=updated
24. Laboratorio de Sistemas Inteligentes de Movilidad · GitHub, accessed on October 16, 2025, https://github.com/simovilab/
25. san089/Optimizing-Public-Transportation: A real-time event pipeline around Kafka Ecosystem for Chicago Transit Authority. - GitHub, accessed on October 16,

2025, https://github.com/san089/Optimizing-Public-Transportation
26. vehicle-routing · GitHub Topics, accessed on October 16, 2025, https://github.com/topics/vehicle-routing
27. How to build Public Transport Accessibility map from scratch with Python - Medium, accessed on October 16, 2025, https://medium.com/@tregub95/how-to-build-public-transport-accessibility-map-from-scratch-with-python-6df9d7755804
28. GTFS and GTFS-RT for beginners | Open Innovations, accessed on October 16, 2025, https://open-innovations.org/blog/2024-12-23-gtfs-and-gtfs-rt-for-beginners
29. A Guide To Using GTFS Data - CARTO, accessed on October 16, 2025, https://carto.com/blog/gtfs-data
30. How to process GTFS data using pandas & geopandas | by Maksym Kozlenko - Medium, accessed on October 16, 2025, https://max-coding.medium.com/how-to-process-gtfs-data-using-pandas-geopandas-4b34f2ad3273
31. MobilityData/awesome-transit: Community list of transit APIs, apps, datasets, research, and software :bus::star2::train - GitHub, accessed on October 16, 2025, https://github.com/MobilityData/awesome-transit
32. GTFS To Public Transit Data Model (Public Transit)—ArcGIS Pro | Documentation, accessed on October 16, 2025, https://pro.arcgis.com/en/pro-app/latest/tool-reference/public-transit/gtfs-to-public-transit-data-model.htm
33. GTFS Data Analysis - RPubs, accessed on October 16, 2025, https://rpubs.com/Billy_Archbold/842095
34. Introduction to k-Means Clustering with scikit-learn in Python - DataCamp, accessed on October 16, 2025, https://www.datacamp.com/tutorial/k-means-clustering-python
35. Geospatial Machine Learning 101: Exploring k-means clustering with geospatial data, accessed on October 16, 2025, https://medium.com/@wambualouis/geospatial-machine-learning-101-exploring-k-means-clustering-with-geospatial-data-a69c6e475838
36. KMeans — scikit-learn 1.7.2 documentation, accessed on October 16, 2025, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
37. 2.3. Clustering — scikit-learn 1.7.2 documentation, accessed on October 16, 2025, https://scikit-learn.org/stable/modules/clustering.html
38. Clustering and Regionalization - Geographic Data Science with Python, accessed on October 16, 2025, https://geographicdata.science/book/notebooks/10_clustering_and_regionalization.html
39. ARIMA for Time Series Forecasting: A Complete Guide - DataCamp, accessed on October 16, 2025, https://www.datacamp.com/tutorial/arima
40. Research on Traffic Flow Prediction Based on ARIMA Model - SciTePress, accessed on October 16, 2025, https://www.scitepress.org/Papers/2024/128878/128878.pdf
41. Time Series Forecasting in Python - Jumping Rivers, accessed on October 16,

2025, https://www.jumpingrivers.com/blog/time-series-forecasting-python-arima/
42. Python ARIMA Tutorial | InfluxData, accessed on October 16, 2025, https://www.influxdata.com/blog/python-ARIMA-tutorial-influxDB/
43. How to Build ARIMA Model in Python for time series forecasting? - ProjectPro, accessed on October 16, 2025, https://www.projectpro.io/article/how-to-build-arima-model-in-python/544
44. Tutorial — NetworkX 3.5 documentation, accessed on October 16, 2025, https://networkx.org/documentation/stable/tutorial.html
45. Network analysis with NetworkX — Reproducible Data Science + Python + Real-World Data, accessed on October 16, 2025, https://valdanchev.github.io/reproducible-data-science-python/notebooks/09_network_analysis.html
46. single_source_dijkstra — NetworkX 3.5 documentation, accessed on October 16, 2025, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.single_source_dijkstra.html
47. dijkstra_path — NetworkX 3.5 documentation, accessed on October 16, 2025, https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.dijkstra_path.html
48. Dijkstra's algorithm | Memgraph's Guide for NetworkX library - GitHub Pages, accessed on October 16, 2025, https://memgraph.github.io/networkx-guide/algorithms/shortest-path/dijkstra/
49. (PDF) AI-Driven Optimization for Efficient Public Bus Operations, accessed on October 16, 2025, https://www.researchgate.net/publication/396403459_AI-Driven_Optimization_for_Efficient_Public_Bus_Operations
50. Public Transport Arrival Time Prediction Based on GTFS Data ..., accessed on October 16, 2025, https://www.researchgate.net/publication/358336330_Public_Transport_Arrival_Time_Prediction_Based_on_GTFS_Data
51. Advancing School Bus Routing: A Machine Learning Approach for Enhanced Efficiency, Safety, and Sustainability - IJFMR, accessed on October 16, 2025, https://www.ijfmr.com/papers/2022/6/16031.pdf
52. CityFlow, accessed on October 16, 2025, https://cityflow-project.github.io/