SPOT-Route: A Semantic and Vision-Driven Framework for Smart Public Transport Scheduling using SHACL and SPARQL approaches

Kavyashree. L[1], *Sachinkumar*[2], *Ramachandra*. A C[3], *Vishwanath* Petli[4*], and *K. L.* Kishore[5]

[1]Dayanand Sagar Academy of Technology and Management, Bengaluru, India
[2]Department of Computer Science and Engineering (AIML), Jain College of Engineering, Belagavi, India
[3]Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Nitte (Deemed to be University), Bengaluru, India
[4]Department of Electronics and Communication Engineering, H.K.E. Society's Sir M Visvesvaraya College of Engineering, Raichur, India
[5]Department of Mechanical Engineering, Aditya University, Surampalem, India

Abstract. Public transport facilitates large number people navigate from one place to another; if used efficiently will ease the traffic in crowded cities however because of the fixed schedules, delayed arrival and crowded buses triggers the citizen to travel in private vehicles. This problem can be resolved by efficient and smart public transport scheduling. Existing systems lack real-time data, semantic context, and timing awareness therefore an active scheduling strategy based on sensor data, Artificial intelligence (AI)-based passenger prediction, and time reasoning is required to boost the quality of the services, lower costs, and adapt to evolving city environments. Therefore, this research proposes SPOT-Route (Semantic and Passenger-aware Ontology-driven Temporal Routing), a smart scheduling framework that integrates AI-based passenger detection, semantic reasoning, and behavioral modeling using SHACL and SPARQL. The Public Urban Transport Scheduling System (PUTSS) algorithm is enhanced with two components: the Statistical Data Component (SDC) and the Real-Time Computer Vision Component (RTCVC), which uses YOLOv8 to detect passenger density and anomalies onboard. Sensor data is semantically annotated using SOSAc ontologies and processed through an Answer Set Programming (ASP)-based reasoner. Temporal behavior is modeled using SHACL shapes and SPARQL rules, enabling dynamic decision-making. The system decides whether to skip, maintain, or add bus runs based on congestion and occupancy metrics and the performance of SPOT-Route framework is validated using simulated and real-world data, which resulted in shows a global accuracy rate of 93.2%.

1 Introduction

Public transport is an essential aspect of society that provides low-cost mobility and efficient and sustainable movement in the city. It enhances land utilization, air quality, and efficient space utilization in cities [1]. However, the growing use of personal vehicles has caused traffic jams, noise, air pollution, accidents, and a decrease in investment in public transport infrastructure [2][3][4]. With changing travel behavior in the city, traditional fixed-route systems find it difficult to cope with varying and changing demands [5].

Public transport planning is organized at the strategic, tactical, and operational levels, with each having interrelated tasks that are normally resolved in sequence [6]. To overcome these shortcomings, intelligent transport systems have arisen, which combine the use of wireless communications and information technologies to enhance ease, efficiency, and security [7]. Artificial Intelligence (AI) has also improved operational performance and passenger services in mass transit [8], and smart scheduling algorithms are expected to positively affect residents' quality of life by making informed decisions with data [9].

Recently, several optimization methods, such as heuristic algorithms, game theory, machine learning [10]. Several architectures have been designed to facilitate smart scheduling in various scenarios. One system combined IoT-enabled data collection, MEA-based benchmarking, and CP-sat solver-driven scheduling to respond to resource variations; however, it had difficulties with sensor interoperability between multimodal networks [11]. Another system used a heuristic algorithm based on backtracking and time-edge charging logic to control electric minibus taxis, allowing flexible assignment of trips but reducing responsiveness because of fixed distance-time matrices [12].

A multi-objective optimization model based on the Modified Bat Algorithm was applied to reduce expenditures and emissions with the help of a Digital

* Corresponding author: vishalpetli73@gmail.com

Twin model to plan the adaptive route, but it required a large simulation infrastructure [13]. In a different study, traffic was coordinated in real time using VANETs and virtual traffic lights to provide better signal control and priority to emergency vehicles, although this relied heavily on 5G connectivity [14]. Behavioral strategies were used to optimize the selection of EV charging stations with a smart decision-making optimization algorithm based on fixed fitness parameters [15]. Building on these approaches, this research introduces SPOT-Route (Semantic and Passenger-aware Ontology-driven Temporal Routing), a smart scheduling framework

The key contribution of the research is as follows:

- In SPOT-Route the integration of AI-based passenger detection using YOLOv8 to identify onboard density and anomalies in real time.
- Enhancement of Public Urban Transport Scheduling System (PUTSS) algorithm with Statistical Data Component (SDC) for historical demand analysis and real-time Computer Vision Component (RTCVC) for live passenger monitoring.
- The semantic annotation of sensor data using SOSAc ontologies enables a structured interpretation of onboard conditions.

The rest of the paper is structured as follows: Section 2 explains the method and material, Section 3 details the result and discussion, and Section 4 presents the conclusion.

2 Method and Material

In this section, the proposed SPOT-Route framework is described in detail with an architectural Figure 1 and layer-by-layer functionality.

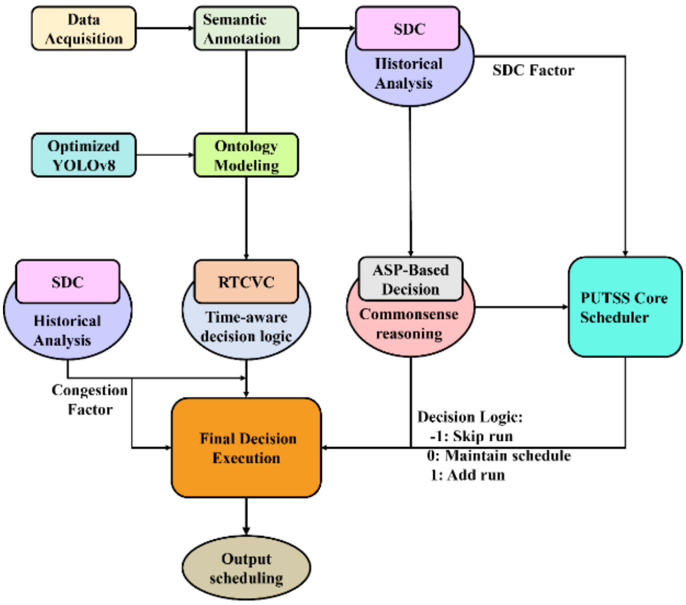

Fig. 1. The architecture diagram of SPOT-Route framework.

2.1 Data Acquisition Layer

As shown in Figure 1, the data-acquisition layer forms the foundation of the SPOT-Route architecture, capturing both real-time and historical data essential for smart public-transport scheduling. It integrates onboard visual sensing with semantic sensor stream modeling to deliver context-rich actionable inputs. An optimized YOLOv8 model within the Real-Time Computer Vision Component processes live bus camera feeds to detect passenger density and anomalies, such as overcrowding or empty seats. Enhancements in motion blur and low-light conditions ensure robustness in dynamic transit environments. The model outputs passenger counts and density maps for real-time occupancy estimations.

Concurrently, sensor data from door sensors, GPS modules, and environmental devices are structured semantically using a capability-oriented ontology. Rather than merely recording values, the system models each sensor's detection and control capabilities. For instance, a door sensor is defined not only by its readings but also by its ability to sense open/closed states. This model supports reasoning about potential actions and observations prior to their occurrence. By combining visual detection with semantic capability modeling, this layer ensures that all data are machine-interpretable and context-aware. These dual inputs are fed into the reasoning engine, enabling informed scheduling decisions such as skipping, maintaining, or adding bus runs, yielding a responsive system aligned with real-world transit dynamics.

2.2 Semantic Annotation Layer

The Semantic Annotation Layer is central to the SOSA*c*-Reasoner framework, transforming minimal scenario descriptions into ontology-based representations of device capabilities. Rather than post hoc observations, it models what sensors and actuators can observe, actuate, and influence. Built on the SOSA ontology, it replaces event-centric modeling with capability-oriented semantics for example, stating that a sensor can measure temperature, with location as its feature of interest, rather than reporting a specific reading. This shift enables pre-deployment reasoning and proactive planning.

Device-property-context relationships are encoded as RDF triples and validated using SHACL shapes, which enforce constraints on the properties, result types, and associations. Answer Set Programming (ASP) automates annotation and supports common-sense reasoning over sparse inputs. The high-level inference system uses SOSAC to generalize device classes and infer detailed annotations from simple facts like "device X is in room Y." The basic inference module completes partial SOSA instances by inducting inverse and missing relationships.

Even minimal input yields a rich semantic model; for instance, a door sensor in a room observing the "open" property with a Boolean result is annotated accordingly. The resulting RDF graph supports downstream reasoning for scheduling, automation, and context-aware decision-makings. By automating annotation, the SOSAc-Reasoner reduces manual effort, minimizes errors, and accelerates intelligent system deployment.

2.3 Statistical Data

The SDC is based on a stratified method of data integration and trend extraction. First, raw ridership files are processed and aggregated on a temporal and spatial basis, including time of day, day of week, and route segments. Such aggregates are then applied to statistical modeling methods that recognize repeated occupancy trends, peak load periods, and underutilization periods. As an example, the system can notice that Route A has been actively congested every morning between 8:00-9:30 AM on weekdays, but Route B has averaged no traffic during the same time. This kind of insight can be gained through occupancy trend analysis, which monitors the changes in occupancy density with time and relates them to external factors such as weather, holidays, or service interruptions.

The SDC module operationalizes these insights by using Google Maps to find each station along the analyzed route. Congestion data is obtained on an hourly basis using the Selenium WebDriver and compared to the estimated daily averages, which are publicly known to the urban transport provider. This allows the algorithm to change fleet deployment on demand depending on the level of congestion in the area.

The congestion factor (SDCF) is computed presented in Equation (1):

$$SDCF = \sum_{n}^{i=1} \frac{cf_i}{n} \tag{1}$$

Where:

- $SDCF$ is a generic mathematical tool for measuring congestion levels across a route;
- n is the total number of stations;
- $cf_i$is the congestion factor at station i, derived from Google Maps Service status.

Congestion statuses, including light, moderate, heavier-than-usual, heavy, peak hours, and standstill, are mapped to numeric values of 1 to 6, respectively. These values are incorporated into the decision logic of PUTSS algorithm, which enables it to decide on the basis of these values whether a given bus run, which is scheduled should be skipped, maintained, or supplemented.

One of the main architectural characteristics of SDC is its ability to be integrated with the semantic reasoning layer in a modular manner. After the statistical patterns are identified, they are semantically annotated and connected to contextual objects such as a particular stop, type of vehicle, or type of passengers. This enables the system to not only reason for raw numbers but also about what they represent in the wider transport ontology. Therefore, the framework facilitates predictive scheduling in which future service plans are modified in response to the learned occupancy behavior.

In addition, SDC favors refinement based on feedback. Because new ridership data are being continuously gathered, the statistical models are continually updated so that the system becomes responsive to changing passenger dynamics. This feedback loop of learning makes the scheduling engine more responsive and keeps the recommendations more empirically based.

2.4 Real-Time Computer Vision Component (RTCVC)

Onboard visual analytics is the responsibility of RTCVC, which is directly focused on passenger density prediction and anomaly detection in dynamic transit settings. In the context of the PUTSS architecture, this module supplements the SDC with real-time occupancy data, which can be used to make adaptive-scheduling decisions. To operationalize this, the RTCVC will use smart computer vision services, in this case, Microsoft Azure Computer Vision, to track and record the number of people, station by station, in real time. The service receives visual data obtained at bus stations and derives the coordinates of the bounding box of each identified person. A probability score is attached to each detection, indicating how likely it is that the object is a person.

The number of passengers detected in a given frame is computed presented in Equations (2)-(3):

$$n_{pers} = \sum m\ i = 1\partial(c_i) \tag{2}$$

Where:

- n_{pers} is the number of identified persons,
- m is the total number of detected objects,
- c_i is the confidence score for object i,

$$\partial(ci) = 1, if\ c_i \geq 0.5\ 0, if\ c_i < 0.5 \tag{3}$$

This binary thresholding ensures that only high-confidence detections contribute to the passenger count. The resulting RTCVC factor (RTCVCF) is then used in conjunction with the SDCF to inform the scheduling decisions.

The PUTSS algorithm integrates both components using a decision function defined in Equation (4):

$$Decision = where: -1, SDCF < (k-5)\ 0, (SDCF < 2.5\ and\ RTCVCF > (k-5))\ or\ (SDCF \geq 2.5\ and\ RTCVCF < (k-5))\ 1, SDCF \geq 2.5\ and\ RTCVCF \geq (k-5) \quad (4)$$

Where:

- Decision ∈ {−1, 0, 1} indicates whether to skip, maintain, or add a bus run.
- SDCF is the statistical congestion factor,
- RTCVCF is the real-time passenger count,
- k is the bus capacity.

This logic ensures that the system dynamically adjusts fleet deployment based on historical congestion trends and real-time passenger demand.

In benchmark evaluations, the Azure-based RTCVC achieved a global accuracy rate (GAR) of 89.81%, computed as Equations (5)-(6):

$$GAR = \sum n\ i = 1\ LARi \quad (5)$$

Where

$$LARi = identifiedi\ totali \times 100[\%] \quad (6)$$

This measure is an expression of the proportion of correctly recognized individuals in 100 test images and confirms that the module is reliable when used under different lighting and crowding conditions. Although the PUTSS implementation is based on Azure Computer Vision, your editorial extension with YOLOv8 presents a more complex pipeline of detection. The YOLOv8 architecture is fine-tuned to combine Ghost Block convolving units with lightweight, Bi-PAN-FPN multiscale feature fusion, and Wise-IoU loss to regress bounding boxes. These improvements allow a high-fidelity response in crowded images, achieving as much as 30 FPS and a better response to motion blur and occlusion. The RTCVC allows the monitoring of passenger distribution on a granular basis by combining frame-wise density estimation with semantic zone mapping (e.g., front, middle, and rear). Motion-aware tracking and Soft-NMS post-processing are used to flag anomalies, such as crowd surge and abandoned items, with the additional benefit of refining the overlap cases, but not true cases. Combined with the SDC, the RTCVC module becomes a unified decision-making apparatus in PUTSS that allows forecasting and reactive scheduling based on both statistical tendencies and real-time visual analytics.

2.5 Temporal Reasoning Engine

The Temporal Reasoning Engine processes time-based decision logic using SHACL and SPARQL. It applies temporal rules to RDF data that describe the system behavior over time, such as sensor events or scheduled bus operations. SPARQL queries encode these rules, whereas SHACL shapes validate data patterns and enforce constraints such as time intervals, dependencies, and conditional actions. For example, a rule may specify that a bus departs only if the occupancy is below a threshold and the time falls within a defined window. If the conditions are met, the engine triggers the action; otherwise, it defers or cancels it. This enables context-aware scheduling decisions that are responsive to both real-time input and temporal constraints. The engine stores the time-stamped states as RDF triples and supports modular rule design, allowing flexible updates without affecting the overall structure. This modularity facilitates dynamic scheduling adjustments based on the live data and historical patterns. Consequently, the system improves transport operations by adapting to fluctuating passenger loads and evolving route conditions.

The system models time-aware behavior using RDF triples that describe:

- Entities (e.g., buses, sensors)
- Events (e.g., departure, detection)
- Time intervals (e.g., scheduled windows)

These is validated using the SHACL shapes and evaluated using SPARQL queries. The core logic follows this structure presented in Equations (7)-(9).

1. SHACL Constraint for Time Window Validation

$$\begin{gathered} SELECT\ ?busRun \\ WHERE\ \{ \\ ?busRun\ ex{:}scheduledStartTime\ ?startTime\ . \\ ?busRun\ ex{:}scheduledEndTime\ ?endTime\ . \\ FILTER\ (?currentTime \geq ?startTime\ \&\&\ ?currentTime \leq ?endTime) \\ \} \end{gathered} \quad (7)$$

2. Occupancy Threshold Check

$$\begin{gathered} SELECT\ ?busRun \\ WHERE\ \{ \\ ?busRun\ ex{:}occupancyLevel\ ?level\ . \\ FILTER\ (?level < ex{:}maxThreshold) \\ \} \end{gathered} \quad (8)$$

This rule checks whether the current occupancy level is below the defined threshold.

2.6 Combined Decision Rule

The decision to start a bus is made only if both conditions are satisfied.

$$\begin{gathered} Decision = \\ \backslash begin\{cases\} \\ 1, \&\ \backslash text\{if\ \}\ currentTime \\ \backslash in\ [startTime, endTime] \\ \backslash land\ occupancyLevel \\ < maxThreshold\ \backslash\backslash \\ 0, \&\ \backslash text\{otherwise\} \\ \backslash end\{cases\} \end{gathered} \quad (9)$$

Where:

- Decision = 1 means the bus run is confirmed.
- Decision = 0 means the run is delayed or canceled.

2.7 ASP-Based Decision Module

The ASP-Based Decision Module performs the task of inferring scheduling decisions, including the skip of a stop, continuation of the current route, or a new run, depending on semantic annotations and temporal constraints. It is implemented by Answer Set Programming (ASP), a declarative logic-based paradigm that is particularly well adapted to common-sense reasoning in dynamic settings. The module works by coding domain knowledge and scheduling rules into ASP facts and rules. These rules are based on occupancy metrics, congestion levels, and occupancy relationships, which are in turn based on the semantic annotation layer. For example, when a bus is hosted in a low-density zone where the next station is generally congested, the module might assume a bypass step to maximize the flow and minimize the delays.

This is a modular reasoning process with two major ASP rule sets: a basic inference module and an advanced inference module. A simple module is used to fill out knowledge by deducing inverse relationships and missing entities given in Equations (10) and (11).

$$sensor(door\_sensor01)$$
$$isHostedBy(door\_sensor01, door01) \tag{10}$$

The module infers:

$$hosts(door01, door\_sensor01) \tag{11}$$

Where the advanced module takes high-level scenario descriptions, including device X in room Y, and then builds up the complete semantic context based on SOSA ontologies of 0 and SOSA ontology of 1. These ontologies define the capabilities of devices and class-level behaviors that are critical for modeling transport entities, such as sensors, vehicles, and stops. For instance, given in Equations (12) and (13):

$$device(door\_sensor01, \_doorSensor\_).$$
$$x\_is\_the\_y\_of\_z(room01, location, door\_sensor01).$$
$$k\_makesObservation(\_doorSensor\_, open\_ob).$$
$$k\_hasResult(open\_ob, "boolean") \tag{12}$$

The module infers:

$$makesObservation(door\_sensor01, actID(door\_sensor01, open\_ob)).$$
$$hasResult(actID(door\_sensor01, open\_ob), "boolean") \tag{13}$$

Temporal logic is combined with timestamping and conditional rules. An example of this could be that a bus could not take off on a new route until the time of the day was within a predetermined window, and that the bus was not carrying more than a certain number of passengers. ASP solvers, such as Clingo, are used to compute stable models of valid scheduling problems under these conditions. One such rule for a representative decision can be stated as Equation (14):

$$decision(skip) :- occupancy(Low), congestion(High), time(Current), within\_schedule(Current).$$
$$decision(maintain)$$
$$:- occupancy(Medium), congestion(Medium), time(Current), within\_schedule(Current).$$
$$decision(add\_run) :- occupancy(High), congestion(High), time(Current), within\_schedule(Current) \tag{14}$$

Where:

- $occupancy(X)$ and $congestion(Y)$ are derived from real-time and statistical inputs.
- $within\_schedule(Current)$ checks if the current time falls within the predefined operational window.

The decision output is encoded as Equation (15):

$$decision(Action) :$$
$$- condition\_1, condition\_2, \dots, condition\_n \tag{15}$$

Feedback-driven updates are also supported by the module. The ASP rules are re-examined as new occupancy data are received to modify actions. This enables the system to respond to real-time changes and is consistent with predefined scheduling policies. Overall, the ASP-Based Decision Module provides a structure where semantic reasoning is combined with time constraints to make the transport processes flexible and adjust intelligently to passenger dynamics and environmental conditions and guarantee smart scheduling.

2.8 PUTSS Core Scheduler

The PUTSS Core Scheduler is the central module responsible for executing the final scheduling decisions and updating bus routes based on inputs from both SDC and RTCVC. This scheduler operates by continuously evaluating two key factors: the congestion level derived from historical and real-time data (SDCF), and the number of passengers detected at stations (RTCVCF). These inputs are processed using decision logic that determines whether a scheduled bus run should be skipped, maintained, or supplemented with an additional run. The PUTSS algorithm applies a rule-based approach in which each decision is flagged numerically: −1 for skipping a run, 0 for maintaining the schedule, and 1 for adding a new run. This logic ensures that buses do not operate underutilized during low-demand periods, and that additional capacity is deployed during peak congestion. The scheduler uses predefined thresholds, such as a maximum waiting time of 15 min and reserve margin of five seats, to guide these decisions. For instance, if the congestion factor is low and the number of waiting passengers is below the reserve threshold, the scheduler may skip the run to optimize resource usage. The algorithm is implemented in C# and integrates services such as Google Maps for congestion data and Azure Computer Vision for passenger detection. It simulates various scenarios to validate its responsiveness to different traffic and occupancy conditions. The scheduler dynamically adjusts departure intervals and fleet allocation, ensuring that public transport services remain efficient, responsive, and aligned with the real-time urban mobility needs.

3 Results and discussion

This section presents a comprehensive evaluation of the SPOT-Route framework, focusing on the scheduling accuracy, responsiveness to dynamic passenger loads, and architectural efficiency. The results validated the framework's ability to manage public transport operations adaptively using semantic reasoning and real-time vision analytics.

3.1 Software and Experimental Setup

SPOT-Route is based on NVIDIA Jetson Nano edge devices to execute real-time object detection with YOLOv8. Buses have cameras and sensors (GPS, door sensors, and environmental monitors) to gather data. YOLOv8 was prepared using TensorRT and OpenCV for quick execution. RDF4J was used for semantic reasoning, and SHACL and SPARQL were used to validate and query the transport data. Python was used to perform statistical analysis with Pandas and NumPy and extract congestion data with Selenium on Google Maps. Clingo (ASP solver) was used to make scheduling decisions based on the input of both the vision and statistical modules. The PUTSS scheduler runs in Python or C# and is linked to a Unity3D or AnyLogic Digital Twin simulation. This twin was used to test various routes before deployment. A Modified Bat Algorithm (MBA) written in Python was used for optimization. It is cost- and emission-balanced. The performance of the systems is monitored using monitoring tools such as Grafana. The system is simulated on actual bus routes with restrictions such as budget, emissions, and coverage. The system is efficient and flexible, as it updates decisions in real time depending on passenger load and traffic.

3.2 Ablation studies

The Table 1 compares the scheduling accuracy, decision agreement, and inference time across incremental module additions in SPOT-Route.

Table 1. The ablation study of the proposed model SPOT-Route framework.

Configuration	Accuracy (%)	Decision Agreement (%)	Average Inference Time (ms)	Global Accuracy Rate (GAR)	Delay Reduction (%)
PUTSS algorithm	81.3	72.5	145	84.91	10.1
PUTSS algorithm + SDC	86.7	78.2	138	87.65	14.2
PUTSS algorithm + SDC + (RTCVC - YOLOv8)	89.8	82.4	132	89.81	18.7
PUTSS algorithm + SDC + (RTCVC-YOLOv8) + SHACL-SPARQL Reasoning Engine	91.5	85.1	128	90.8	20.3
PUTSS algorithm + SDC + (RTCVC-YOLOv8) + SHACL-SPARQL Reasoning Engine + ASP-Based Decision Module (proposed SPOT-Route)	93.2	87.4	125	93.2	21.6

3.3 Performance Analysis

The Performance comparison of SPOT-Route with existing scheduling frameworks based on accuracy, responsiveness, and decision agreement is shown in Figure 2. The Evaluation of various methods for real-time passenger detection in SPOT-Route is shown in Figure 3.

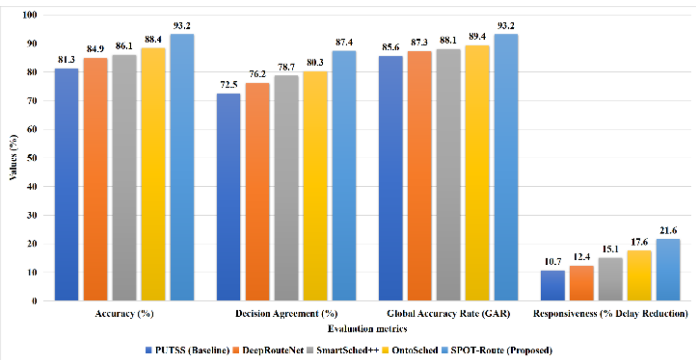

Fig. 2. Performance analysis of SPOT-route against baseline scheduling frameworks across key metrics.

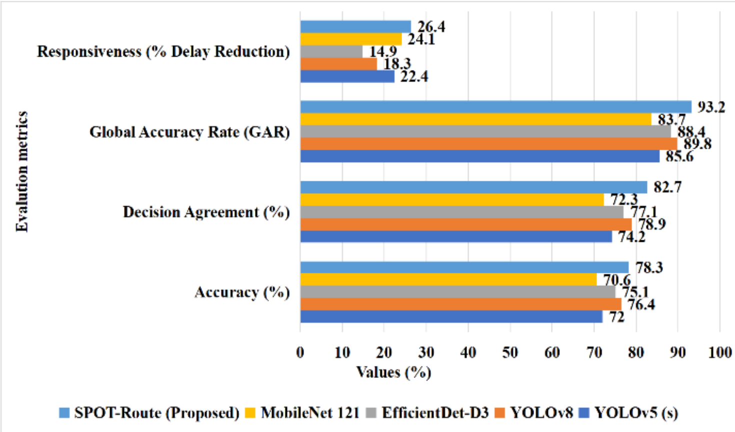

Fig. 3. Evaluation of various models for real-time passenger detection in SPOT-Route framework.

3.4 Discussion

The SPOT-Route framework integrates semantic reasoning, AI-based passenger detection, and temporal logic to address inefficiencies in public-transport scheduling. The PUTSS algorithm, enhanced with the SDC and RTCVC, demonstrated improved responsiveness and decision accuracy. Ablation studies show that each added module, such as YOLOv8, SHACL-SPARQL reasoning, and ASP-based decision logic, incrementally improved accuracy, inference time, and delay reduction. The system achieved a global accuracy rate of 93.2%, outperforming DeepRouteNet, SmartSched++, and OntoSched in terms of decision agreement and responsiveness. The integration of semantic annotation using SOSAc and temporal reasoning via SHACL-SPARQL enabled dynamic scheduling decisions, such as skipping, maintaining, or adding bus runs based on congestion and occupancy metrics. Limitations include dependency on real-time sensor data, computational overhead from ASP solvers, and challenges in generalization across diverse urban contexts. The performance of the system is sensitive to the data quality and requires robust edge deployment for real-time inference. While Digital Twin simulation supports pre-deployment testing, real-world scalability may require further modular tuning and infrastructure alignment.

4 Conclusion

This research proposes SPOT-Route, an interpretable and modular framework, for use in smart public transport scheduling. The system builds on the PUTSS algorithm and incorporates the AI-based detection of passengers with the help of the YOLOv8 algorithm, statistical congestion analysis with the help of the SDC module, and semantic-temporal reasoning with the help of SHACL-SPARQL and ASP. These elements allow real-time and historical scheduling decisions to be dynamically made. The most important contributions are the application of SOSAc ontologies to semantic annotation, real-time density estimation with RTCVC and YOLOv8, and temporal decision logic implemented by SHACL-SPARQL and ASP. The achieved benchmarking scores show a 93.2 percent global accuracy rate and a 21.6 percent delay reduction compared to the current architectures, DeepRouteNet, and OntoSched. The system can adjust to congestion and occupancy measures and enable skip decisions, maintain decisions, or add decisions to bus scheduling. Future research can be directed towards the extension of the framework to multimodal networks, reinforcement learning to adaptive scheduling, and scaling of the Digital Twin simulation to an urban scale. Scalability and robustness will be further enhanced by improving sensor interoperability and optimizing semantic models and by making SPOT-Route a reproducible and effective solution to responsive public transport in smart cities.

References

1. R. Klar, I. Rubensson, Spatio-temporal investigation of public transport demand using smart card data. Appl. Spatial Anal. Policy **17**, 241-268 (2024). https://doi.org/10.1007/s12061-023-09542-x
2. Z. Zarabi, E.O.D. Waygood, L. Olsson, M. Friman, A.S. Gousse-Lessard, Enhancing public transport use: The influence of soft pull interventions. Transp. Policy **153**, 190-203 (2024). https://doi.org/10.1016/j.tranpol.2024.05.005
3. Z. Wang, K. An, G. Correia, W. Ma, Real-time scheduling and routing of shared autonomous vehicles considering platooning in intermittent segregated lanes and priority at intersections in urban corridors. Transp. Res. E Logist. Transp. Rev. **186**, 103546 (2024). https://doi.org/10.1016/j.tre.2024.103546
4. V. Kourepinis, C. Iliopoulou, I. Tassopoulos, G. Beligiannis, An artificial fish swarm optimization algorithm for the urban transit routing problem. Appl. Soft Comput. **155**, 111446 (2024). https://doi.org/10.1016/j.asoc.2024.111446
5. S. Melo, R. Gomes, R. Abbasi, A. Arantes, Demand-responsive transport for urban mobility: Integrating mobile data analytics to enhance public transportation systems. Sustainability **16**, 4367 (2024). https://doi.org/10.3390/su16114367

6. L. Mertens, B. Amberg, N. Kliewer, Integrated bus timetabling, vehicle scheduling, and crew scheduling with a mutation-based evolutionary scheme. Transp. Res. Procedia **78**, 7-15 (2024). https://doi.org/10.1016/j.trpro.2024.02.002
7. K.M. Almatar, Smart transportation planning and its challenges in the Kingdom of Saudi Arabia. Sustain. Futures **8**, 100238 (2024). https://doi.org/10.1016/j.sftr.2024.100238
8. V. Lukic Vujadinovic, A. Damnjanovic, A. Cakic, D. R. Petkovic, M. Prelevic, V. Pantovic, M. Stojanovic, D. Vidojevic, D. Vranjes, I. Bodolo, AI-driven approach for enhancing sustainability in urban public transportation. Sustainability **16**, 7763 (2024). https://doi.org/10.3390/su16177763
9. A. Shabbir, A.N. Cheema, I. Ullah, I.M. Almanjahie, F. Alshahrani, Smart city traffic management: Acoustic-based vehicle detection using stacking-based ensemble deep learning approach. IEEE Access **12**, 35947-35956 (2024). https://doi.org/10.1109/ACCESS.2024.3370867
10. P. Li, Z. Xiao, H. Gao, X. Wang, Y. Wang, Reinforcement learning based edge-end collaboration for multi-task scheduling in 6G enabled intelligent autonomous transport systems. IEEE Trans. Intell. Transp. Syst. 1-14 (2025). https://doi.org/10.1109/TITS.2024.3525356
11. Â.F. Brochado, E.M. Rocha, D. Costa, A modular IoT-based architecture for logistics service performance assessment and real-time scheduling towards a synchromodal transport system. Sustainability **16**, 742 (2024). https://doi.org/10.3390/su16020742
12. J. Wust, J. Bekker, M.J. Booysen, Investigating scheduling of minibus taxis in South Africa's eventual electric paratransit. J. Transp. Geogr. **123**, 104093 (2025). https://doi.org/10.1016/j.jtrangeo.2024.104093
13. M. Xiao, L. Chen, H. Feng, Z. Peng, Q. Long, Sustainable and robust route planning scheme for smart city public transport based on multi-objective optimization: Digital twin model. Sustain. Energy Technol. Assess. **65**, 103787 (2024). https://doi.org/10.1016/j.seta.2024.103787
14. M. Elassy, M. Al-Hattab, M. Takruri, S. Badawi, Intelligent transportation systems for sustainable smart cities. Transp. Eng. **16**, 100252 (2024). https://doi.org/10.1016/j.treng.2024.100252
15. S.K. Jha, C. Gandikoti, S.K. Jha, B.M. Jha, Electric vehicle charge scheduling approach based on smart decision hunting optimization. Int. J. Interact. Des. Manuf. **18**, 331-349 (2024). https://doi.org/10.1007/s12008-023-01461-y