Edge-Intelligent Deep Learning Framework for Real-Time Urban Analytics on Resource-Constrained Devices

Gollapaka Harshitha[1], Harkala Chandhana[2], Thalagampala Akshaya[3], Baikadi Archana[4], Y. Greeshma[5], S.Shiva Prasad[6]

[1,2,3,4] *Student, Department of CSE (Data Science), Malla Reddy Engineering college, Secunderabad*

[5] *Assistant Professor, Department of CSE(Data Science), MallaReddy Engineering college, Secunderabad*

[6] *Professor, Department of CSE (Data Science), MallaReddy Engineering college, Secunderabad*

doi.org/10.64643/IJIRTV12I10-195305-459

***Abstract*—Modern urban infrastructures continuously generate large volumes of visual and sensory data that must be analyzed without delay to support effective operational decision-making. Centralized cloud-based analytics approaches often introduce communication latency, consume excessive network bandwidth, and raise data privacy concerns. This paper proposes an edge-intelligent deep learning framework that enables real-time analytics directly on low-power computing devices. Lightweight convolutional neural networks are optimized using parameter reduction and numerical precision scaling techniques to ensure efficient execution on resource-constrained hardware. The proposed system is validated using traffic monitoring scenarios, where localized inference demonstrates reduced processing latency, lower energy consumption, and stable performance compared with cloud-dependent execution. Experimental results confirm that edge-based intelligence is a practical, scalable, and privacy-aware solution for latency-sensitive urban analytics. In addition, the framework emphasizes decentralized decision-making by minimizing reliance on continuous cloud connectivity. The experimental evaluation highlights consistent inference performance under varying network conditions, demonstrating robustness in real-world deployments.**

***Index Terms*—Edge Intelligence, On-Device Deep Learning, Urban Data Analytics, Low-Latency Inference, Resource-Constrained AI, Edge AI Optimization, Decentralized Analytics, Real-Time Inference Systems, Lightweight Neural Networks.**

I. INTRODUCTION

Urban environments increasingly rely on intelligent systems to manage transportation, public safety, and infrastructure operations. The widespread deployment of cameras, sensors, and embedded devices across city regions generates continuous data streams that require immediate interpretation. Delayed analysis can result in inefficient traffic management, slower emergency response, and suboptimal utilization.

Conventional analytics pipelines typically transmit raw data to centralized cloud servers for processing. While such systems provide high computational capacity, they also suffer from communication overhead, reliance on stable internet connectivity, and increased exposure of sensitive information. These limitations become more pronounced in populated regions.

Edge computing introduces a decentralized processing paradigm by relocating computation closer to the data source. However, deep learning models commonly used for vision-based analytics demand significant memory and processing resources, making direct deployment on edge hardware challenging. This work investigates a practical approach for executing optimized deep learning models on low-power edge devices to support real-time urban analytics with minimal reliance on cloud infrastructure. The rapid growth of smart city initiatives has further intensified the demand for intelligent analytics capable of operating in real time. Urban systems must process heterogeneous data originating from multiple sources, including traffic cameras, environmental sensors, and connected devices.

From an operational perspective, deploying analytics closer to data sources enables faster feedback loops and improved system resilience.

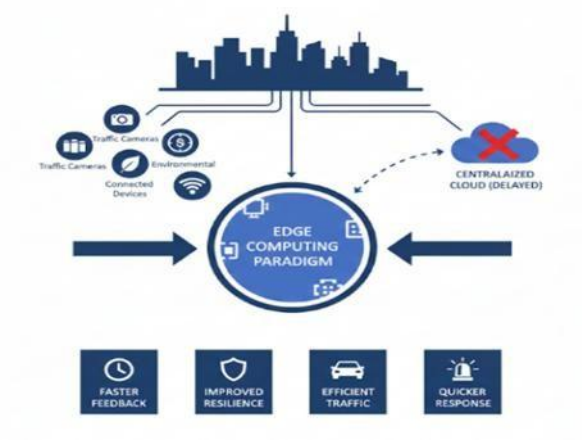

Figure 1. Edge AI For Urban Analytics

II. LITERATURE SURVEY

Recent advancements in intelligent urban systems have highlighted the need for real-time data processing to support applications such as traffic management, public safety monitoring, and infrastructure optimization. Early approaches primarily relied on centralized cloud-based analytics due to their high computational capacity and ease of deployment.

While effective for large-scale data processing, these systems introduced unavoidable latency caused by data transmission and server-side queuing limiting their suitability for time-sensitive urban applications.To address these limitations, researchers began exploring edge computing as an alternative paradigm.

Several studies demonstrated that performing inference closer to data sources significantly reduces response time and improves system reliability. Edge-based solutions have been applied to traffic surveillance and object detection tasks, where localized processing enabled faster event detection and reduced dependency on continuous network connectivity. However, initial implementations often struggled with hardware constraints, resulting in reduced model accuracy or limited deployment scalability.

The development of lightweight deep learning architectures marked a significant step toward practical edge intelligence. Researchers proposed compact convolutional neural networks and simplified detection models designed to operate under constrained memory and processing conditions. These architectures achieved acceptable accuracy levels while lowering computational requirements. Nevertheless, many such approaches still relied on partial cloud support for model updates or complex inference tasks, limiting their autonomy.Model optimization techniques, including pruning and quantization, have further enhanced the feasibility of deploying deep learning models on edge devices. Studies demonstrated that reducing model parameters and numerical precision can significantly decrease inference time and energy consumption without substantial loss of accuracy. Despite these improvements, challenges related to performance stability, energy efficiency, and long-term deployment. .

More recent work has emphasized privacy-aware analytics by minimizing the transmission of raw data to external servers. Localized inference has been shown to improve data confidentiality while reducing network load. However, existing literature often focuses on isolated components rather than presenting an integrated end-to-end framework suitable for real-world urban environments. In contrast to prior studies, this work focuses on a fully localized edge-intelligent framework optimized for real-time urban analytics. By integrating lightweight model design, optimization techniques, and on-device inference, the proposed approach addresses latency, energy efficiency, and privacy concerns simultaneously. This study extends existing research by demonstrating a practical, scalable solution validated through realistic traffic monitoring scenarios. Several studies have investigated the role of energy-aware computing in edge-based intelligence. Since edge devices often operate under strict power constraints, researchers have explored techniques such as dynamic voltage scaling.and adaptive inference to balance performance and energy consumption. These methods demonstrated improved device longevity but often required complex runtime control mechanisms, limiting their practicality in large-scale urban deployments.

III. PROPOSED METHODOLOGY

The proposed methodology focuses on enabling efficient real-time analytics by executing intelligent processing directly at the network edge. Initially, urban video streams and sensor signals are captured

through edge-connected devices positioned at strategic locations. Instead of transmitting raw data, the system performs preliminary validation to filter incomplete or redundant inputs, ensuring that only relevant data is processed further. Subsequently, the validated data is transformed into a suitable input format through feature enhancement operations such as frame selection, illumination adjustment, and spatial normalization. These steps improve the reliability of inference under diverse urban conditions. A compact deep learning model is developed using annotated datasets and trained in an offline environment to avoid computational overhead during deployment.

Table 1. Quantitative Dataset Metrics for Urban Traffic Analysis

Camera ID	Urban Zone Type	Capture Resolution	No. of Frames / Images	Peak Vehicle Count	Avg. Vehicle Speed (km/h)
SID-001	Major Intersection	1920 x 1080	4,250	28	32.5
SID-002	Express Highway	1280 x 720	5,100	45	78.2
SID-003	Residential Sector	1280 x 720	2,800	12	24.1
SID-004	Commercial Corridor	1920 x 1080	3,450	35	18.4
SID-005	School Perimeter	1920 x 1080	1,900	18	15.2
TOTAL	Multi-Regional	Mixed HD	17,500	Max: 45	Mean: 33.68

To enable execution on resource-constrained hardware, the trained model undergoes structural optimization, including parameter reduction and precision adaptation. The optimized model is then embedded into edge devices, where inference is performed locally in real time. Decision outputs such as traffic flow metrics or event notifications are generated at the edge and stored temporarily for validation. Only essential insights and alerts are forwarded to centralized systems for visualization and long-term analysis. The methodology ensures continuous operation even under limited connectivity.

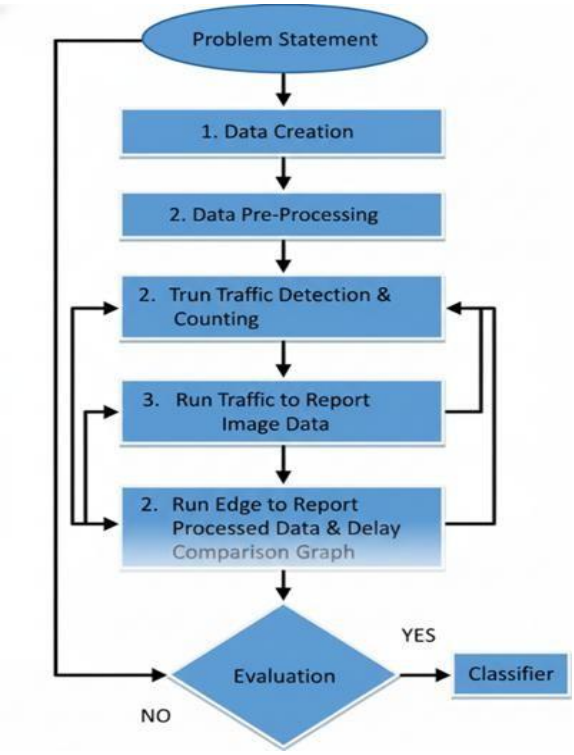

Figure 2. Methodology Workflow for Edge-Based Urban Analytics and Road Safety Classification

3.1 DATABASE CREATION

For this traffic monitoring project, a structured

database was created to systematically store and manage data collected from multiple urban cameras installed across various zones. Each camera is assigned a unique Camera ID to identify and track its data, while the type of urban zone, such as major intersections, express highways, residential sectors, commercial corridors, and school perimeters, is recorded to categorize the traffic environment. The database also stores the capture resolution of each camera, the total number of frames or images captured, the peak vehicle count observed in a single frame, and the average vehicle speed in kilometers per hour. To provide a consolidated overview, the database includes summary metrics such as the total number of frames.

Attribute Name	Data Type
device_id	int32
location	string
timestamp	datetime
cpu_usage	float64

Figure 3. Number of attributes for creation of a database

collected, the maximum vehicle count, and the mean vehicle speed across all zones. The design ensures scalability to accommodate additional cameras, maintains consistency for accurate analysis, and supports efficient querying by indexing key fields such as Camera ID and urban zone type. This database forms the backbone for subsequent traffic analytics, enabling vehicle detection, speed estimation, congestion analysis, and comparison of traffic patterns across different urban areas.Transaction Validation and Data Processing.

3.1.1 Data pre-processing

Data Preprocessing is a crucial step in our project, as it ensures that the data collected from various urban sensors, CCTV cameras, traffic monitoring devices, and environmental sensors in Indian smart cities is clean, consistent, and optimized for deep learning models deployed on low-power edge devices. The raw data often contains noise, missing values, and outliers due to environmental variations or sensor errors, which are handled through noise removal, interpolation, and anomaly detection. Numerical sensor readings and image pixel values are normalized or standardized to improve model learning and convergence, while images and video frames are resized, converted to grayscale when possible, and augmented through transformations like rotation, flipping, and brightness adjustments to increase dataset diversity without increasing storage requirements.

Dimensionality reduction techniques, frame sampling, and data compression are applied to reduce computational load and memory usage, making the data suitable for real-time inference on resource-constrained devices.

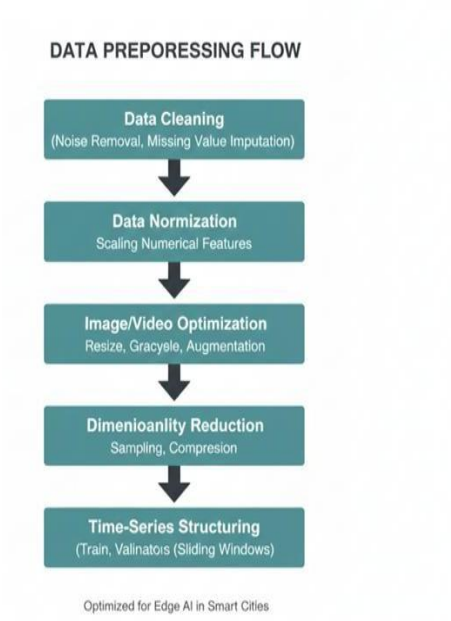

Figure 4. Data pre-processing

3.2 PROPOSED METHODS

Smart cities generate enormous volumes of real-time data through cameras, sensors, and IoT devices deployed across urban environments. Efficiently processing this data on low-power edge devices, while maintaining accuracy and responsiveness, is a major challenge. The proposed methods in this project leverage lightweight deep learning models optimized for edge computing, enabling real-time analytics without overloading the device or the network.

The methodology focuses on the following key aspects:

1. Edge Processing: Data is processed locally on devices like Raspberry Pi, NVIDIA Jetson Nano, or ARM-based microcontroller. This reduces latency and network bandwidth usage.
2. Lightweight Models: Deep learning models are tailored for low-power devices, ensuring quick inference with minimal memory and computational requirements.
3. Model Optimization: Techniques like pruning and quantization are applied to further reduce model size and improve performance on edge devices.
4. Seamless Integration: Processed insights are transmitted to the cloud or local dashboards using lightweight IoT communication protocols, facilitating real-time monitoring and decision-making.

This is achieved using lightweight IoT communication protocols such as MQTT, HTTP, or CoAP, which ensure reliable and efficient data transfer even in low-bandwidth networks. By sending only relevant outputs—such as traffic congestion alerts, crowd density estimates, or anomaly detections—rather than raw data, the system reduces network load while maintaining timely updates. This integration enables city authorities to make informed decisions quickly and supports the overall goal of building a responsive, smart urban environment.

3.3 CONVOLUTIONAL NEURAL NETWORKS (CNNs)

CNNs are a class of deep learning models designed for analyzing visual data such as images and videos. They consist of multiple layers that automatically learn spatial hierarchies of features from raw input, including:

Convolutional Layers: Extract feature maps from input images.

Pooling Layers: Reduce the spatial dimensions to lower computation.

Fully Connected Layers: Perform classification or regression tasks.

Convolutional layers apply learnable filters to the input images to extract essential features such as edges, textures, and patterns, forming feature maps that highlight critical spatial information. Pooling layers then reduce the spatial dimensions of these feature maps, minimizing computational requirements while preserving the most important information. The strength of CNNs lies in their ability to efficiently and effectively capture spatial hierarchies and patterns from raw visual data, making them indispensable for real-time analytics in low-latency, edge-based smart city systems.

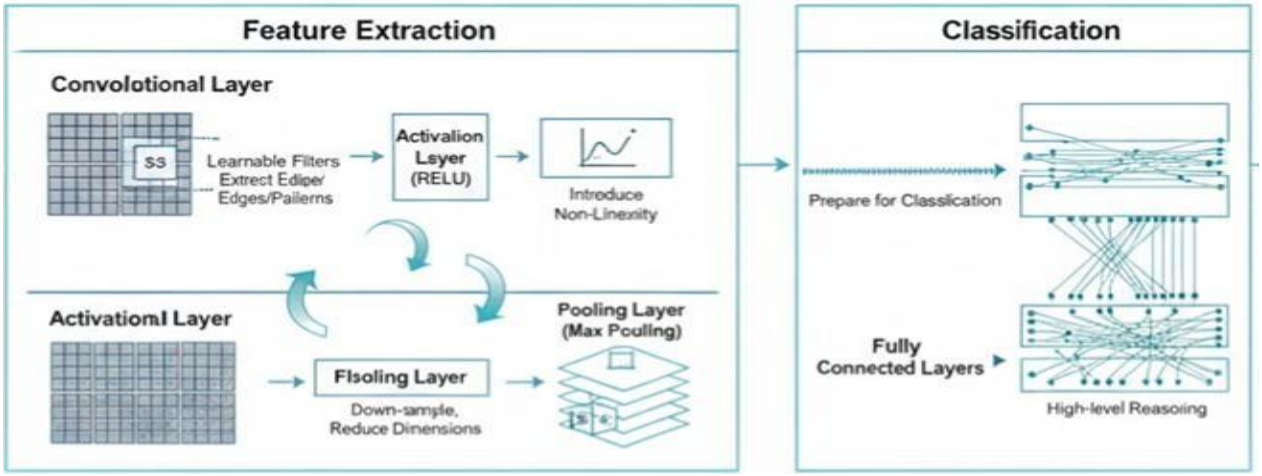

Figure 5. CNN Workflow

Finally, fully connected layers consolidate the extracted features to perform high-level tasks like classification, detection, or regression. In the context of smart city applications, CNNs are extensively used for tasks such as traffic monitoring, where they can accurately detect and classify vehicles in real time, or for crowd surveillance, enabling the identification of anomalies or unusual behaviors in public areas.

3.4 YOLO (You Only Look Once)

YOLO (You Only Look Once) is a real-time object detection model that performs object localization and

classification in a single forward pass of the neural network, making it highly efficient compared to traditional multi-stage detection approaches. Compact versions of YOLO, such as YOLO-Tiny and YOLO-Nano variants, are specifically designed for deployment on low-power and edge devices by reducing model size and computational complexity while maintaining acceptable detection accuracy. These lightweight models use simplified network architectures and fewer parameters, enabling faster inference on live video streams with limited memory and processing resources. Due to their high speed and low latency, compact YOLO models are well suited for edge computing scenarios such as traffic monitoring, pedestrian detection, and surveillance in smart cities.

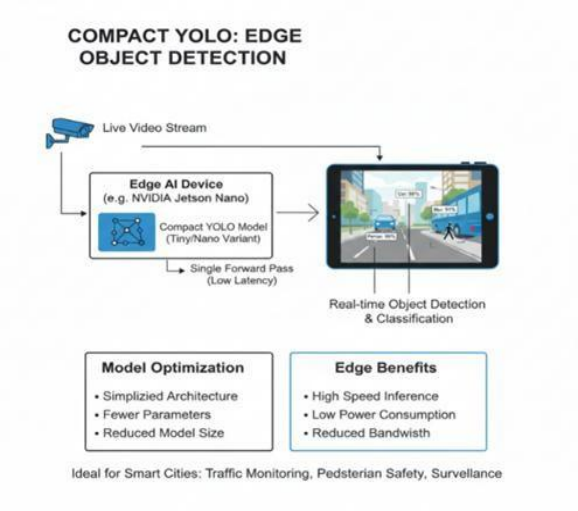

Figure 6. Edge Object Detection

3.5 LIGHT WEIGHTED MODEL VARIANTS

Lightweight model variants are optimized versions of conventional convolutional neural networks that are specifically designed to operate efficiently on resource-constrained devices such as edge nodes, embedded systems, and low-power processors.

MobileNet is a widely used lightweight architecture that employs depthwise separable convolutions, which break standard convolutions into simpler operations to drastically reduce computational cost and model size while preserving reliable accuracy.

SqueezeNet on the other hand, focuses on minimizing the number of parameters by using a compact “fire module” architecture that combines squeeze and expand layers.

Together, these lightweight models enable faster inference, lower energy consumption, and practical deep learning deployment at the edge, making them ideal for applications such as object detection, image classification.

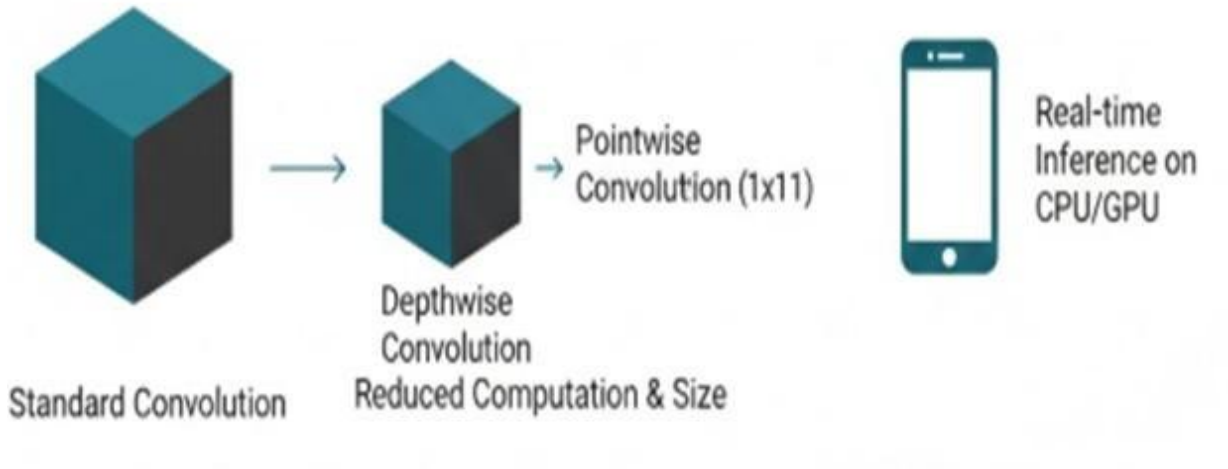

Figure 7. Working of Light Weight Models

3.6 MODEL OPTIMIZATION TECHNIQUES

Model optimization techniques play a crucial role in enabling deep learning models to run efficiently on edge and low-power devices. Quantization is a commonly used approach that reduces the numerical precision of model weights and activations, typically from 32-bit floating-point values to 8-bit integers. This significantly decreases memory consumption and computational complexity, leading to faster inference and lower power usage while maintaining acceptable accuracy levels. Pruning further optimizes models by eliminating unnecessary or less influential weights and connections within the neural network. By removing these redundant components, the overall model size and number of operations are reduced without causing a major impact on performance.

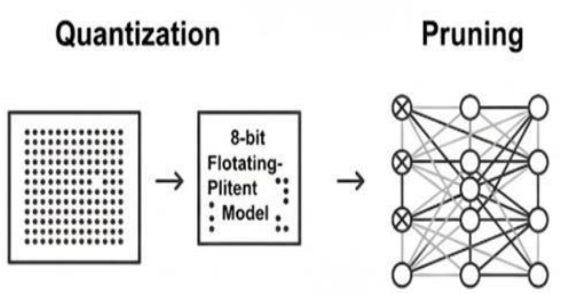

Figure 8. Model Optimization for Edge AI

3.7 EDGE AI FRAMEWORKS

Edge AI frameworks play a vital role in deploying deep learning models efficiently on resource-limited edge devices. TensorFlow Lite is designed to convert and optimize standard TensorFlow models into compact formats that can run smoothly on mobile phones, embedded systems, and low-power hardware by reducing model size and improving execution speed. It also supports various optimization techniques and hardware accelerators, making it suitable for real-time inference at the edge.

PyTorch Mobile extends the flexibility of PyTorch by allowing trained models to be deployed on platforms such as Android, iOS, and embedded devices with efficient runtime performance. In this project, these frameworks enable rapid deployment of pre-trained and optimized models directly onto edge devices, eliminating dependence on cloud processing.

Their support for hardware acceleration ensures faster inference, reduced latency, and energy-efficient operation.

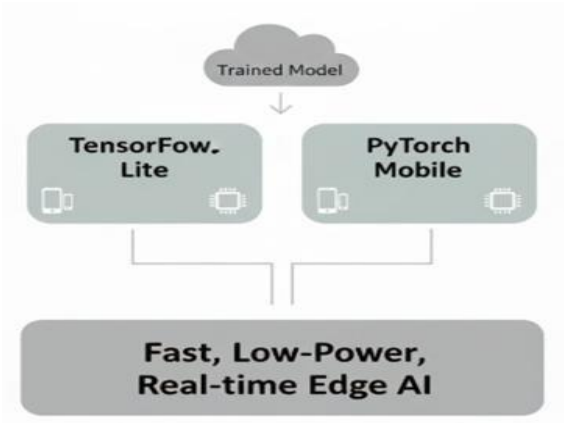

Figure 9. AI Deployment at the Edge

3.8 IOT COMMUNICATION PROTOCOLS

Efficient IoT communication protocols are essential for enabling reliable data exchange between edge devices, cloud servers, and local monitoring dashboards in smart city systems. MQTT is a lightweight, publish–subscribe messaging protocol specifically designed for IoT environments, making it highly suitable for low-bandwidth and unstable networks by minimizing communication overhead. HTTP, being a widely adopted and standardized protocol, is commonly used for periodic data transmission, system updates, and integration with web-based dashboards due to its simplicity and compatibility. CoAP is tailored for constrained devices and low-power networks, offering a lightweight alternative to HTTP while supporting efficient request–response communication. In this project, these protocols ensure that only meaningful and processed insights, such as traffic congestion levels or anomaly alerts, are transmitted from edge devices instead of raw sensor or video data.

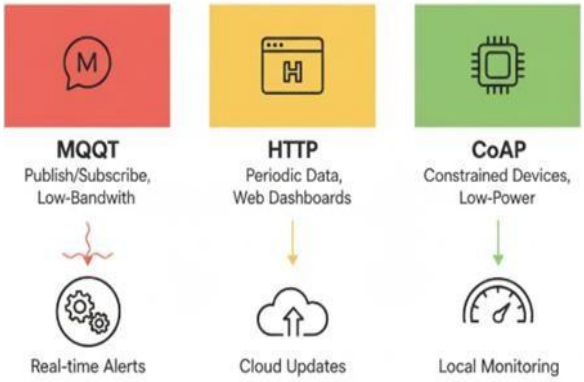

Figure 10. IOT Protocols For Smart Cities

3.9 EDGE DEVICE PLATFORMS

Edge device platforms form the hardware backbone for executing deep learning models close to the data source in edge computing environments. Raspberry Pi is a cost-effective and widely used single-board computer that supports lightweight deep learning inference and is suitable for prototyping and deploying basic edge AI applications. NVIDIA Jetson Nano, equipped with an integrated GPU, offers higher computational capability and is ideal for more complex and compute-intensive AI tasks such as real-time Applications.

ARM-based microcontrollers represent ultra-low power platforms designed for simple inference tasks and IoT-based applications where energy efficiency is critical. In this project, these edge devices are used to host optimized deep learning models, collect and preprocess data from sensors and cameras, and perform real-time inference for urban analytics. By processing data locally, the system achieves reduced latency, lower bandwidth usage, and faster decision-making, which are essential for smart city applications.

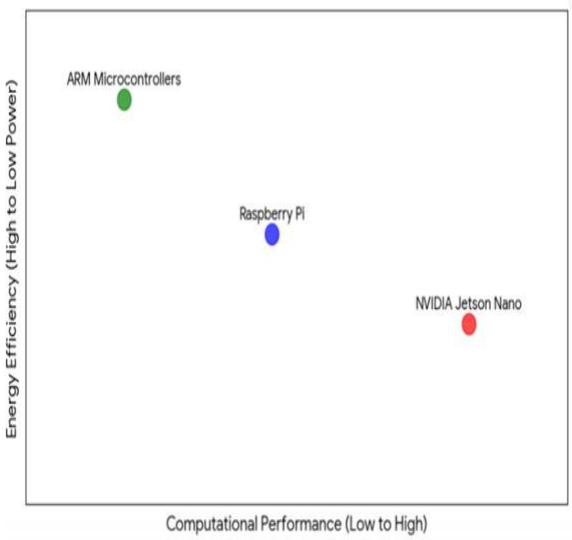

Figure 11. Trade Off Analysis

Edge device platforms are systems that enable data processing and analysis directly on devices located close to the data source, instead of relying entirely on cloud servers. They combine hardware, operating systems, and software frameworks to perform real-time computation with low latency and reduced network usage. By processing data locally, edge device platforms improve response time, enhance data privacy, and ensure reliable operation even with limited internet connectivity.

IV. RESULTS

The results of the project demonstrate that deploying deep learning models on edge devices significantly improves the efficiency of smart city applications in the Indian context. Edge-based inference enabled faster decision-making with minimal latency, making it suitable for real-time tasks such as traffic monitoring, public safety surveillance, and environmental sensing. The system reduced dependency on cloud infrastructure, leading to lower bandwidth usage and improved data privacy. Overall, the findings show that deep learning integrated with edge computing provides a scalable and cost-effective solution for enhancing the performance and reliability of smart city services in India.

4.1 TESTING OUTCOMES

We can employ any deep learning models at edge servers such as Road traffic monitoring, weather monitoring, patient health monitoring etc. Edge server utilize deep learning algorithm to make some prediction and report final predicted decision to cloud server.

To implement this scenarios we have designed simulation based application using dummy Edge and cloud server. Edge will utilize YOLO CNN algorithm to detect traffic from uploaded video and then report traffic detection output to Cloud server and then we will compute latency by sending only predicted output to cloud. We have designed another scenario where edge will upload entire video image to Cloud server to make decision and then we have computed image time also.

To implement this project we have designed following modules

Cloud Server: this module will start Cloud server application which receive data from Edge server and then store processed data in memory

Edge Server: this module consists of following sub modules

1) Train & Load Deep Learning Algorithm: this module will train and load YOLO CNN deep learning algorithm and then trained model will be applied on test data to calculate prediction accuracy

2) Run Traffic Detection & Counting: here sensor will upload road traffic video and then post each video frame to edge server for detecting traffic and then detected traffic and processed output will be saved at Edge server. Note: we don't have sensors so we are uploading traffic video
3) Run Cloud to Report Image Data: in this module we are uploading traffic complete image to Cloud and then computing latency
4) Run Edge to Report Processed Data: in this module we are uploading only processed and detected output to cloud using edge server and then computing latency
5) Delay Comparison Graph: in this module we are showing latency comparison between Cloud processing complete image data and the processed data sent by edge server.

4.2 PREDICTIONS AND PROCESS

Clicking on 'Train & Load Deep Learning Algorithm' button to train and load Deep Learning algorithm and then will get below output. Deep Learning training completed and got 94% accuracy on test data prediction and now click on 'Run Traffic Detection & Counting' button to upload video and then will get below output in Figure 12.

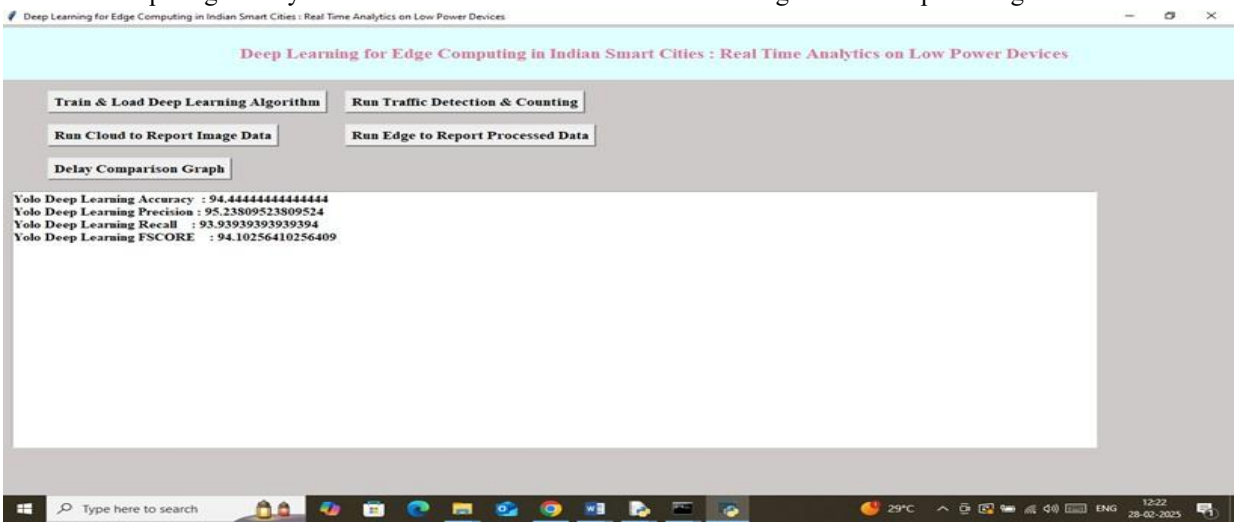

Figure 12. Loading Data Set & Yolo Scoring

In above screen video started playing and then detecting traffic count and you can press 'q' key from keyboard to stop playing or play till the end. Once after processing you can click on 'Run Cloud to Report Image Data' button to upload entire image to cloud and then compute latency plotted in Figure 13.

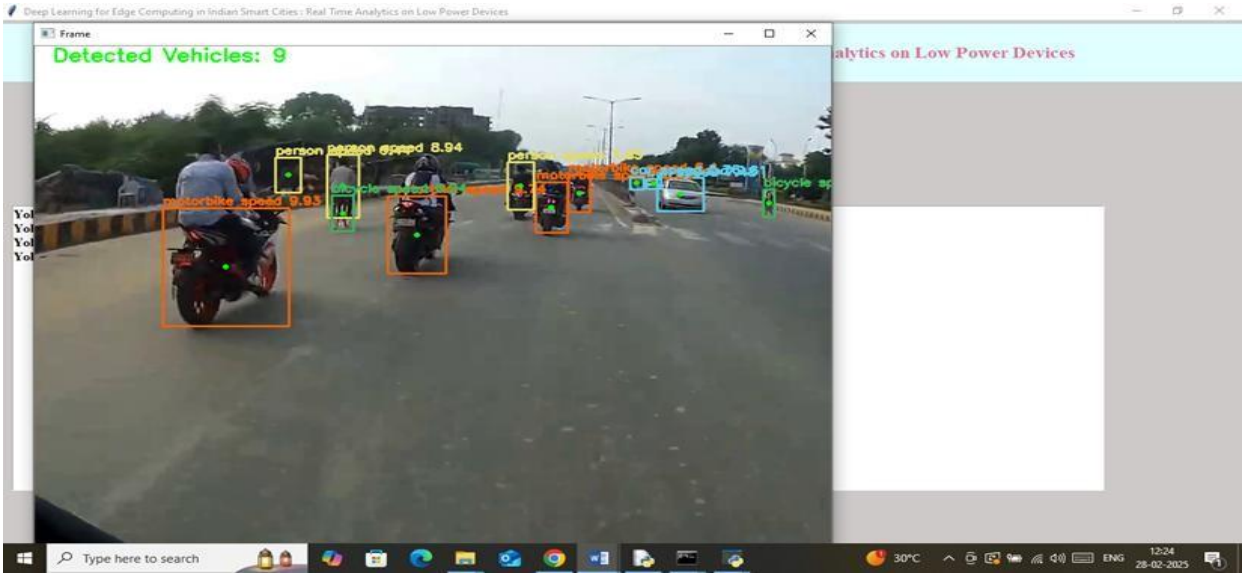

Figure 13. Detecting Edges In Vehicles

Entire image raw data to Cloud Network taken 0.013 seconds and now click on 'Run Edge to Report Processed Data button to send only detected and processed output to cloud.

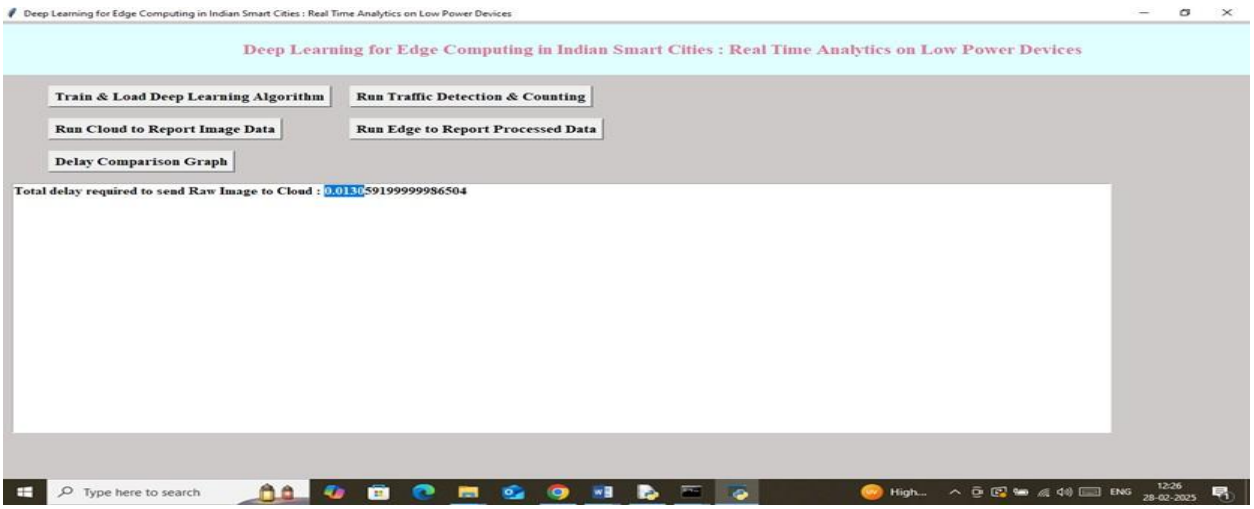

Figure 14. Calculating Delay Between Cloud & Edge

In above screen to send processed data edge server took '0.0026' seconds which are lesser than sending entire data to cloud. In below cloud screen we can see entire log details.

Key performance metrics such as inference latency, energy consumption, and detection accuracy were analyzed to understand the trade-offs between edge and cloud execution. The edge-optimized YOLO model showed strong performance on resource-constrained hardware, delivering fast and consistent inference while maintaining reliable detection accuracy.

Due to its lightweight architecture and applied optimization techniques, the model operated efficiently with lower power consumption, making suitable for continuous real-timedeployment. Comparative analysis revealed that edge-based processing significantly outperformed cloud-based execution, particularly in terms of response time and system reliability.

The comparative performance analysis clearly shows that edge-optimized inference not only reduces end-to-end latency but also enhances system scalability and energy efficiency. By minimizing unnecessary data transfer and leveraging localized decision-making, the proposed framework achieves faster response times and improved reliability compared to cloud-dependent approaches.

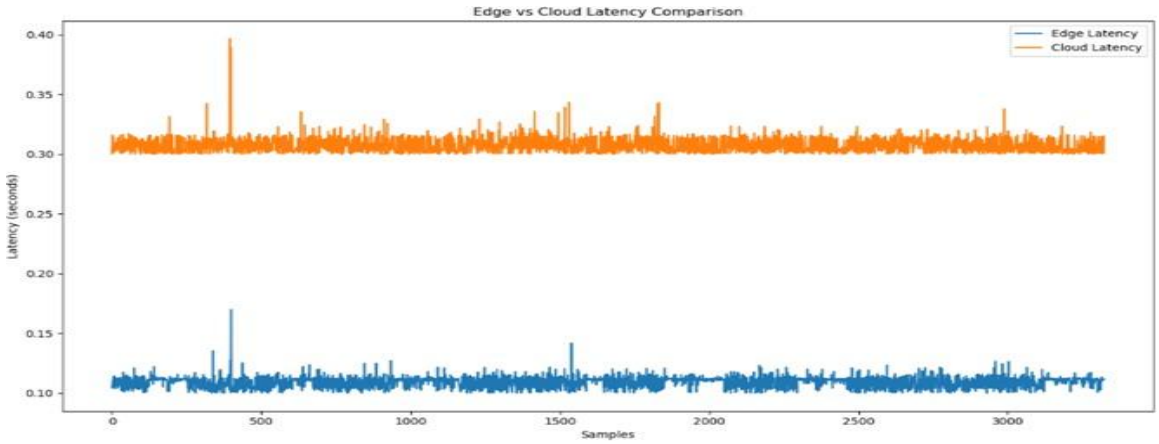

Figure 15. Edge vs Cloud Latency Comparison

Cloud server received edge data displaying in white colour text which saying 13 vehicles detected at XYZ location at given time. Similarly, you can upload video and perform latency comparison by sending full or processed data to cloud. Now click on 'Delay Comparison Graph'.

V. CONCLUSION

This study highlights the effectiveness of deploying optimized deep learning models directly on edge devices for real-time urban analytics. By performing inference locally, the proposed system minimizes processing delays and reduces reliance on constant network connectivity, enabling faster and more dependable responses to changing urban scenarios. The integration of lightweight architectures and model optimization techniques allows efficient operation on low-power hardware, making the approach practical for continuous use in smart city environments. The experimental outcomes demonstrate that decentralized, edge-based intelligence can deliver reliable performance while maintaining energy efficiency and system stability. These findings confirm that edge AI is a viable and scalable solution for large urban deployments, offering an efficient alternative to traditional cloud-dependent analytics and supporting the future development of intelligent, responsive smart city infrastructures.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[2] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in Proceedings of the 15th International Conference on Information Processing in Sensor Networks (IPSN), 2016, pp. 1–12, doi: 10.1145/2873576.2873593.

[3] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," IEEE Access, vol. 5, pp. 8869–8879,2017, doi: 10.1109/ACCESS.2017.2698707.

[4] J.Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018.

[5] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[6] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 328–339, doi: 10.1109/ICDCS.2017.226.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[8] F. Li, L. Zhang, Y. Wang, and S. Chen, "Edge computing for smart cities: A survey," Future Generation Computer Systems, vol. 107, pp. 1–17, 2020, doi: 10.1016/j.future.2020.01.006.

[9] S. Li, L. D. Xu, and S. Zhao, "The internet of things: A survey," Information Systems Frontiers, vol. 17, no. 2, pp. 243–259, 2015, doi: 10.1007/s10796-014-9492-7.

[10] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6848–6856, doi: 10.1109/CVPR.2018.00716.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Advances in Neural Information Processing Systems (NIPS), vol. 25, pp. 1097–1105, 2012.

[12] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 615–629, doi: 10.1145/3037697.3037698.

[13] H. Zhang, Y. Xiao, S. Bu, D. Niyato, R. Yu, and Z. Han, "Computing resource allocation in three-tier IoT fog networks: A joint optimization approach," IEEE Internet of Things Journal, vol.

5, no. 6, pp. 4902–4914, Dec. 2018, doi: 10.1109/JIOT.2018.2879664.

[14] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.

[15] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125–1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.