Latency-Aware and Priority-Based Task Scheduling for Real-Time Traffic Monitoring Using Edge Computing

TEAM MEMBERS:

1.Balaji Sakthivel Marimuthu-CB.SC.U4CSE23213

2.Sushil V-CB.SC.U4CSE23248

3.Uppara Veeranjaneyulu -CB.SC.U4CSE23351

4.Gadde Punith -CB.SC.U4CSE23321

UNDERSTANDING THE IDEA

- Real-time traffic monitoring systems need ultra-low-latency processing to detect events such as accidents, congestion, and emergency vehicle arrival. Traditional cloud-based architectures are not efficient due to high latency and network congestion.
- This project proposes a latency-aware, priority-based task scheduling algorithm implemented on edge devices (e.g., cameras, embedded servers). Our system processes time-critical traffic data locally and dynamically prioritizes tasks based on urgency and resource availability.
- **Our goal**
- Minimize latency by computing at the edge.
- Dynamically schedule tasks based on urgency.
- Ensure tasks like emergency alerts are never delayed.

- **Problem faced with traditional methods/practices:**
- Centralized cloud systems cause delays in time-sensitive traffic detection.
- Real-time video feeds from road cameras generate high-bandwidth data.
- There is no task prioritization or deadline awareness in naive scheduling systems

- **Edge computing solution**
- Edge devices co-located with traffic cameras.
- Implement foreground detection, vehicle speed estimation, and congestion detection algorithms.
- Scheduler decides task execution based on:
  - Task priority (High, Medium, Low)
  - Network condition
  - Edge resource availability
  - Tasks are processed locally when possible. If overloaded, non-urgent tasks may be deferred or dropped.

Use Case:

- We simulate a real-time traffic management system with:
- Vehicle Detection
- Emergency Vehicle Alerting
- Traffic Flow Estimation
- Congestion Detection
- Each of these is treated as a separate ML task with different priority and latency requirements. Priority is assigned based on application criticality:
  - High: Emergency alerts, congestion
  - Medium: Speed estimation
  - Low: Routine vehicle counts

System architecture:

Sensor Layer: Traffic cameras capture 24/7 video feeds.

Edge Layer: Mini-PCs or embedded devices (e.g., Jetson Nano) execute lightweight ML algorithms.

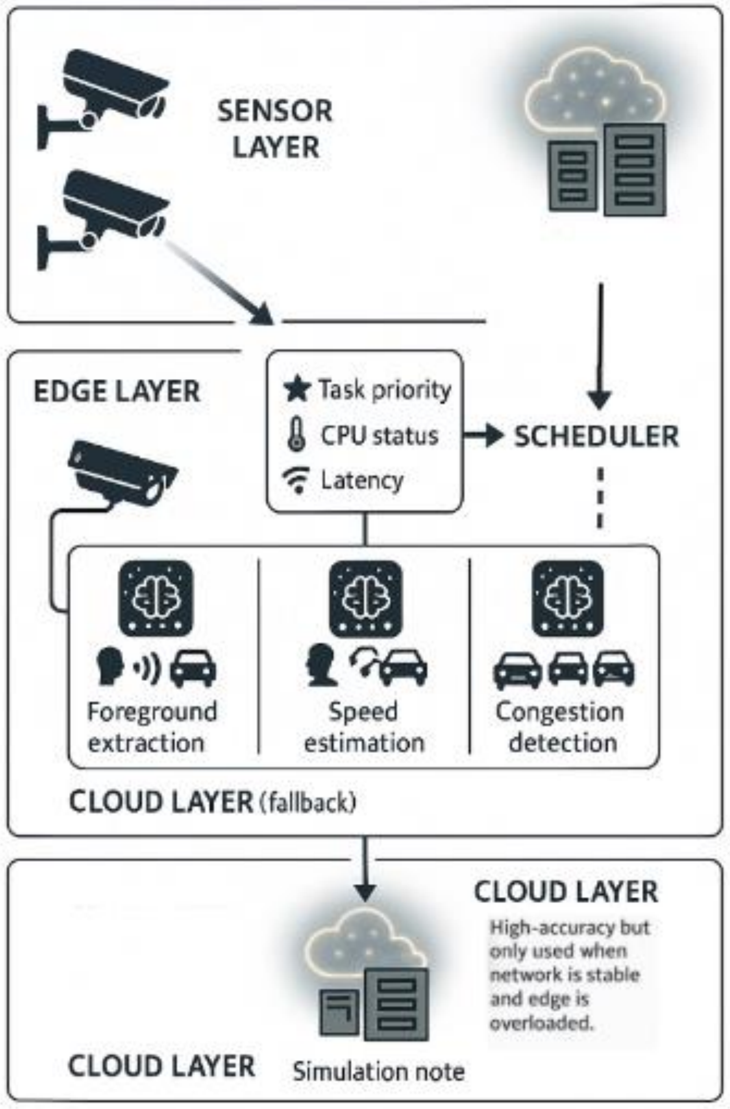
Scheduler: Task priority + local system load + latency threshold determine execution path.

Cloud Layer (fallback): High-accuracy but only used when network is stable and edge is overloaded.

Scheduler design;

- **Input:** Task queue from edge sensors
- **Decision Factors:**
    - Task priority (HIGH, MEDIUM, LOW)
    - Deadline threshold (e.g., 2s for emergency detection)
    - Edge node CPU utilization and bandwidth
- **Output:** Task-to-node assignment in cloud or rejection if deferred or dropped(not required)

Architecture Diagram



Implementation Strategy

1. System Modules overview

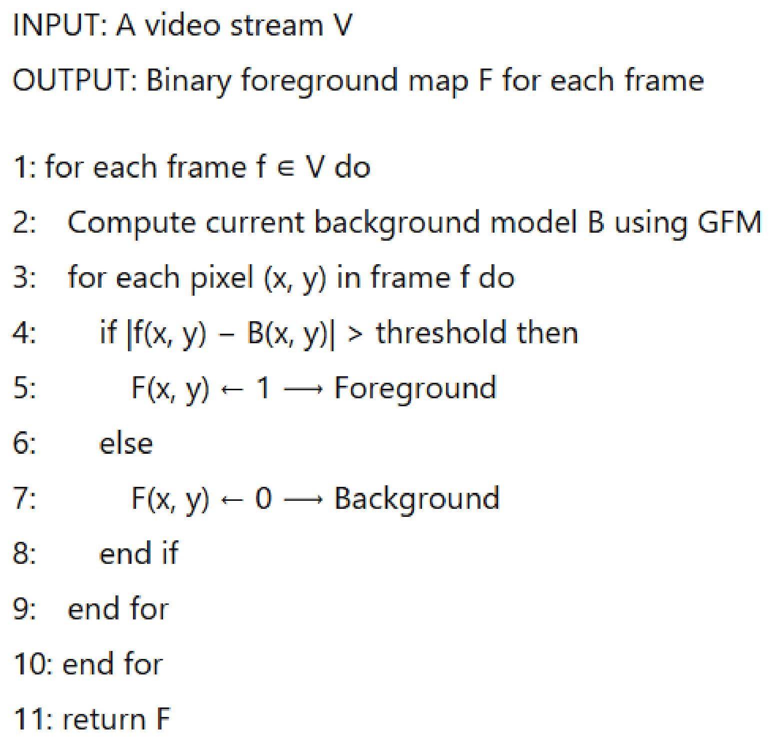
Module	Description
Video Input Module	Streams real-time traffic footage from roadside CCTV/IP cameras
ML Task Processor	Applies vision-based algorithms (foreground detection, tracking, etc.)
Edge Scheduler	Allocates compute tasks based on latency budget & task priority
Monitoring & Metrics Engine	Logs task performance: latency, CPU load, deadline success rate

2. Machine Learning Algorithms Used

We adopt lightweight, edge-compatible ML and CV techniques for traffic analytics:

- **2.1 Foreground Object Detection**
- Used for identifying moving vehicles in the scene.
- Algorithm: Gaussian-based Foreground Model (GFM)

- Explanation:
  - Uses adaptive background subtraction to detect moving objects
  - More robust to lighting changes and weather (compared to classic GMM)
  - Ideal for stopped vehicle detection (even under snow/rain)
- Libraries: OpenCV (BackgroundSubtractorMOG2 or custom GFM logic)



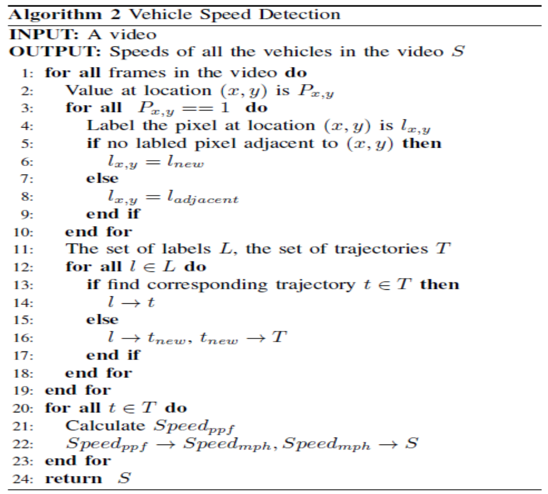
```
INPUT: A video stream V
OUTPUT: Binary foreground map F for each frame

1: for each frame f ∈ V do
2:    Compute current background model B using GFM
3:    for each pixel (x, y) in frame f do
4:       if |f(x, y) − B(x, y)| > threshold then
5:          F(x, y) ← 1 ⟶ Foreground
6:       else
7:          F(x, y) ← 0 ⟶ Background
8:       end if
9:    end for
10: end for
11: return F
```

2.2 Vehicle Tracking and Speed Estimation

- Algorithm: Kalman Filter + Optical Flow
- Explanation:
  - Kalman Filter predicts vehicle motion
  - Lucas-Kanade Optical Flow estimates motion between frames
  - Combined to calculate approximate vehicle speed from trajectory
- Libraries: OpenCV (cv2.calcOpticalFlowPyrLK), NumPy

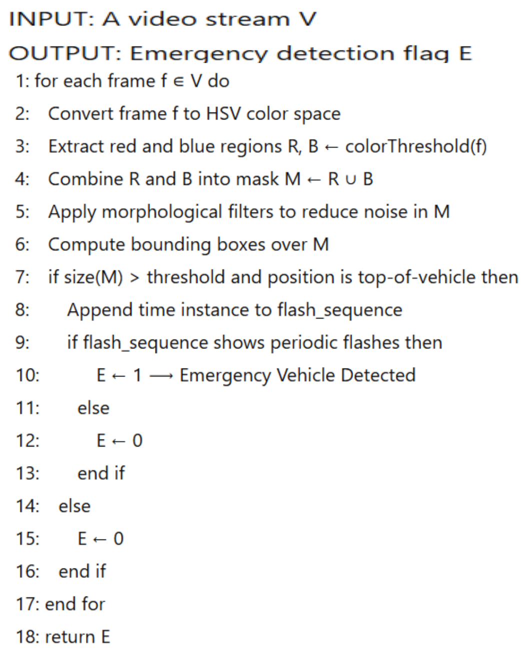
2.3 Emergency Vehicle Detection (Optional Extension)

- Algorithm: Color+Pattern Detection (Red/Blue LED flash patterns)
- Explanation:
  - Uses HSV-based color segmentation and periodic pulse pattern detection
  - Flags emergency vehicle presence to assign highest task priority
- Libraries: OpenCV, Scikit-image



Algorithm 2 Vehicle Speed Detection

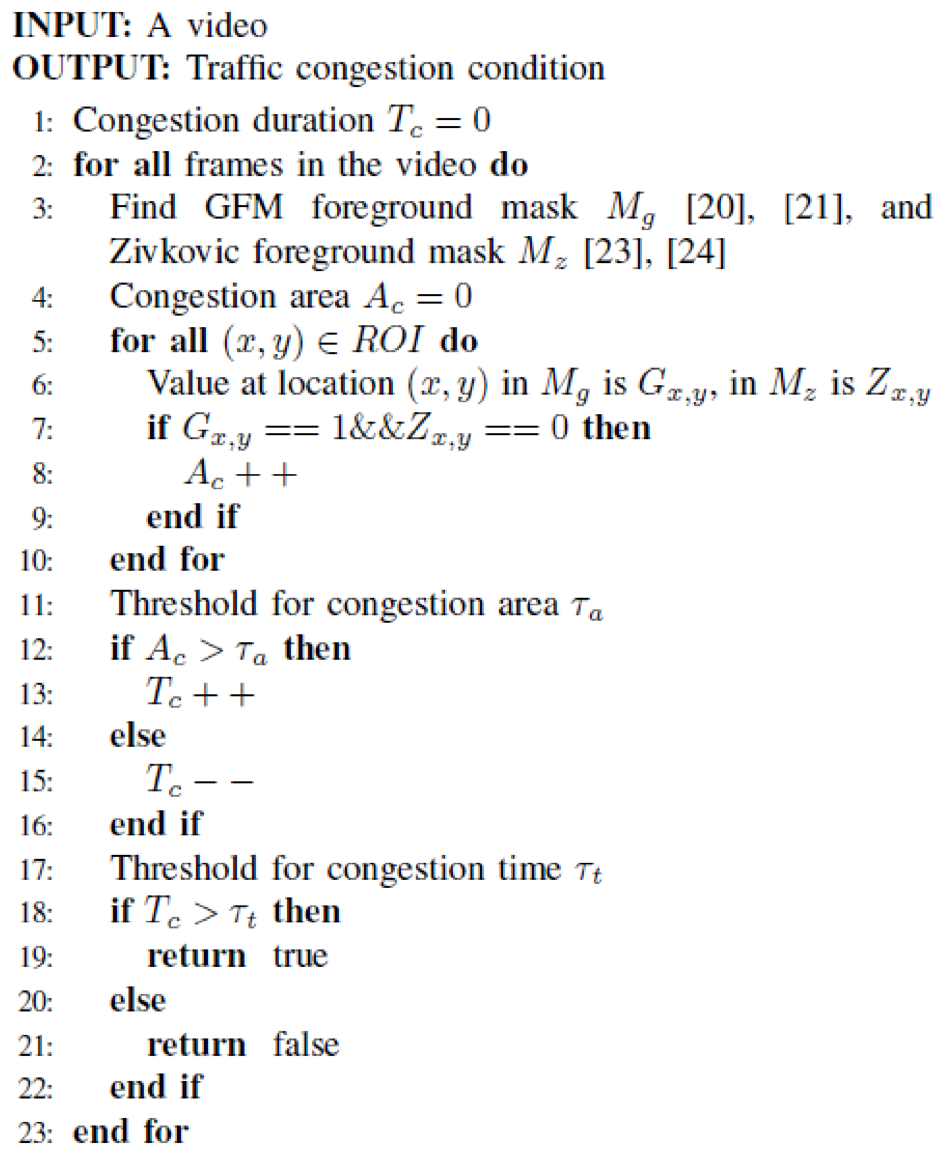
```
INPUT: A video
OUTPUT: Speeds of all the vehicles in the video $S$
 1: for all frames in the video do
 2:    Value at location $(x, y)$ is $P_{x,y}$
 3:    for all $P_{x,y} == 1$ do
 4:       Label the pixel at location $(x, y)$ is $l_{x,y}$
 5:       if no labled pixel adjacent to $(x, y)$ then
 6:          $l_{x,y} = l_{new}$
 7:       else
 8:          $l_{x,y} = l_{adjacent}$
 9:       end if
10:    end for
11:    The set of labels $L$, the set of trajectories $T$
12:    for all $l \in L$ do
13:       if find corresponding trajectory $t \in T$ then
14:          $l \rightarrow t$
15:       else
16:          $l \rightarrow t_{new}, t_{new} \rightarrow T$
17:       end if
18:    end for
19: end for
20: for all $t \in T$ do
21:    Calculate $Speed_{ppf}$
22:    $Speed_{ppf} \rightarrow Speed_{mph}, Speed_{mph} \rightarrow S$
23: end for
24: return $S$
```



```
INPUT: A video stream V
OUTPUT: Emergency detection flag E
 1: for each frame f ∈ V do
 2:    Convert frame f to HSV color space
 3:    Extract red and blue regions R, B ← colorThreshold(f)
 4:    Combine R and B into mask M ← R ∪ B
 5:    Apply morphological filters to reduce noise in M
 6:    Compute bounding boxes over M
 7:    if size(M) > threshold and position is top-of-vehicle then
 8:       Append time instance to flash_sequence
 9:       if flash_sequence shows periodic flashes then
10:          E ← 1 ⟶ Emergency Vehicle Detected
11:       else
12:          E ← 0
13:       end if
14:    else
15:       E ← 0
16:    end if
17: end for
18: return E
```

2.4 Congestion Detection

- Logic: Area Coverage Ratio + Object Count
- Explanation:
  - ○ Compute % of frame occupied by vehicles using bounding boxes
  - ○ High coverage over sustained period → congestion alert
- Threshold: > 60% frame occupied for > 5 seconds → trigger event



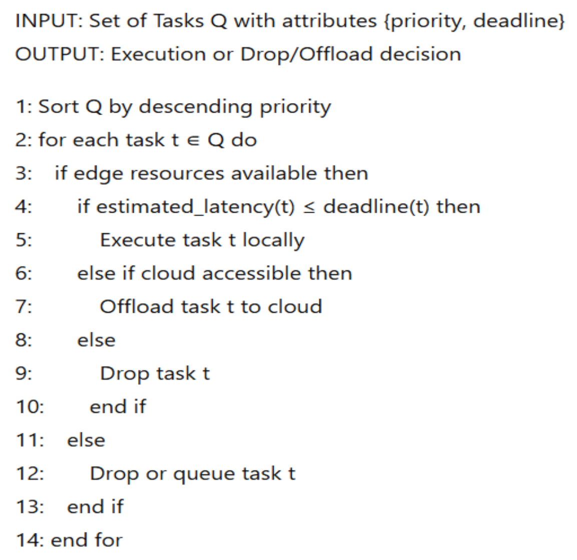
```
INPUT: A video
OUTPUT: Traffic congestion condition
Congestion duration $T_c = 0$
for all frames in the video do
    Find GFM foreground mask $M_g$ [20], [21], and Zivkovic foreground mask $M_z$ [23], [24]
    Congestion area $A_c = 0$
    for all $(x, y) \in ROI$ do
        Value at location $(x, y)$ in $M_g$ is $G_{x,y}$, in $M_z$ is $Z_{x,y}$
        if $G_{x,y} == 1 \&\& Z_{x,y} == 0$ then
            $A_c{+}{+}$
        end if
    end for
    Threshold for congestion area $\tau_a$
    if $A_c > \tau_a$ then
        $T_c{+}{+}$
    else
        $T_c{-}{-}$
    end if
    Threshold for congestion time $\tau_t$
    if $T_c > \tau_t$ then
        return true
    else
        return false
    end if
end for
```

3. Priority-Based Task Scheduling Logic

The edge scheduler runs on each node and uses the following:

Parameter	Role
Task Priority	HIGH (Emergency), MEDIUM (Speed), LOW (Counting)
Latency Budget	Max acceptable delay (e.g., 2s for emergency, 5s for speed)
Node Resource Status	CPU %, RAM, current task queue
Network Bandwidth	Optional — used to decide whether to offload to cloud

- **Scheduling Algorithm:**
- Sort incoming tasks by priority.
- For each task:
  a. If enough resources → execute locally
  b. Else if cloud is reachable and delay is tolerable → offload
  c. Else → queue or drop based on urgency
- Deadlines are embedded for each task type.



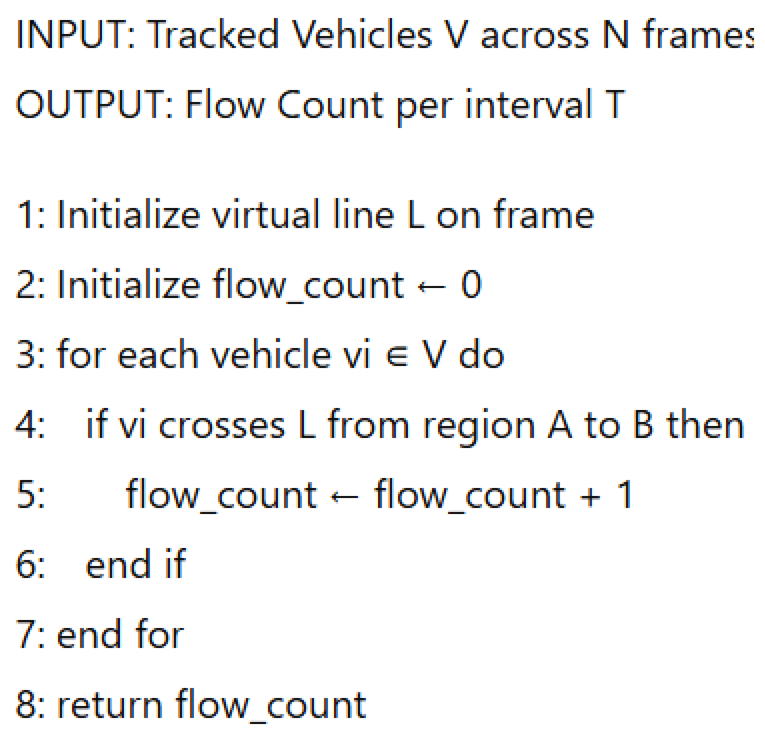
```
INPUT: Set of Tasks Q with attributes {priority, deadline}
OUTPUT: Execution or Drop/Offload decision

Sort Q by descending priority
for each task t ∈ Q do
    if edge resources available then
        if estimated_latency(t) ≤ deadline(t) then
            Execute task t locally
        else if cloud accessible then
            Offload task t to cloud
        else
            Drop task t
        end if
    else
        Drop or queue task t
    end if
end for
```

Traffic Flow Estimation

- Traffic flow estimation is a key component for understanding congestion trends, dynamic signal control, and adaptive traffic management. It uses vehicle detection and tracking over time to estimate flow rate.
- Algorithm Used:
- Vehicle Counting + Flow Rate Estimation using Centroid Tracking
- Explanation:
- Vehicles are detected using foreground segmentation (GFM) or a light object detector.
- Each detected object is assigned a unique ID using centroid-based tracking (e.g., SORT or a simplified version).
- A virtual line or zone is placed on the frame (e.g., near road exit).
- When a vehicle crosses the line in a specific direction, it's counted.
- Count per unit time gives flow rate:
  Flow rate = Vehicle Count / Time Interval

Output Metric:

- Vehicles per minute or per 5 seconds
- Direction-wise flow (e.g., northbound vs. southbound)
- Libraries:
- OpenCV (object detection, line drawing)
- Python (for logic)
- Optional: DeepSORT / light YOLOv3-tiny if higher accuracy is needed



```
INPUT: Tracked Vehicles V across N frame
OUTPUT: Flow Count per interval T

1: Initialize virtual line L on frame
2: Initialize flow_count ← 0
3: for each vehicle vi ∈ V do
4:    if vi crosses L from region A to B then
5:        flow_count ← flow_count + 1
6:    end if
7: end for
8: return flow_count
```

Integration with Scheduler:

- Flow rate output is fed into the congestion detection module.
- If flow drops below a threshold while vehicle density is high → congestion suspected.
- The system then raises the priority of congestion estimation tasks to ensure faster reporting.

STATE OF ART LITERATURE

Task	**Algorithm(s)**	**Priority**	**Output**
Foreground Detection	GFM (Edge Adaptive)	Medium	Binary motion map
Speed Estimation	Optical Flow + Kalman Filter	Medium	Vehicle speed (km/h)
Emergency Vehicle Detect	Color Segmentation + Flash Logic	High	Emergency alert
Congestion Detection	Frame Coverage Ratio	High	Congestion alert
Traffic Flow Estimation	Centroid Tracking + Counting	Medium	Vehicles per minute

- **Simulation:**
- Simulated traffic videos with known vehicle flows used to validate accuracy.
- Performance measured in:
  - Detection Accuracy
  - Counting Precision
  - Time to detect flow drop

- The growing demand for real-time traffic analytics in smart cities has led to extensive research in edge computing and computer vision for traffic monitoring Traditional cloud-based solutions, while scalable, suffer from high latency and bandwidth consumption.

- To address this, researchers have proposed hierarchical computing frameworks, edge-assisted AI models,
- and low-latency scheduling mechanisms.

This section summarizes key advancements and how they relate to our proposed work.

- **Edge-Based Traffic Video Analytics** :

- Liu et al. (2021), IEEE T-ITS, Smart Traffic Monitoring System Using Computer Vision and Edge Computing

- **Key Contributions:**

- Proposed a two-tier edge-cloud framework where roadside edge devices pre-process traffic video feeds.
- Applied foreground detection (GFM) and deep learning (YOLOv3-tiny) for vehicle detection and counting.
- Demonstrated significant latency improvements (from >2s to ~0.5s) by processing detection at the edge.

Latency-Aware Task Scheduling :

- Fang et al. (2019), IEEE TCC – Latency-Aware Scheduling for Mobile Edge Computing .
- EdgeCloudSim Simulation Framework (Sonmez et al., 2018)

Key Contributions:

- Prioritized task scheduling based on delay sensitivity and network

resource availability.

- EdgeCloudSim enables simulation of realistic task arrival, delay,

and mobility scenarios.

Relevance to Our Work:

- We extend EdgeCloudSim with our custom priority scheduler logic.
- Our work focuses on deadline success and resource-aware task selection.

- We implement deadline-based decisions where tasks are dropped, executed,
or offloaded based on system
- load and latency budgets.

Congestion and Speed Estimation via Computer Vision

- Liu et al. (2020) – Vision-based Speed Estimation using Vehicle Trajectory Tracking
- Shi et al. (2018) – GFM-based Congestion Pattern Detection

- **Key Contributions:**

- Trajectory tracking and lane-based speed estimation using warped perspectives.
- Frame-based vehicle coverage ratio used to detect congestion thresholds.

- **Relevance to Our Work:**
- We replicate congestion logic using bounding box density (threshold >60% sustained).
- We use lightweight motion tracking via Kalman filters to estimate vehicle speeds at the edge

Real-World Deployment Insights

- Smart PKU Monitor (2024) – Real-time health alerts from wearable sensors processed at edge
- EdgeAIGC (2025), DCAN Journal – Intelligent edge model caching with RL

Key Contributions:

- Scheduling of urgent tasks (e.g., abnormal ECG) using priority queues at the fog layer.
- Reinforcement Learning-based offloading in resource-constrained edge environments.

Relevance to Our Work:

- Our priority model mimics PKU logic: critical tasks never dropped.
- Future extension: incorporate RL-based scheduling to learn adaptive policies from traffic load patterns.

Comparison Table

Feature / Aspect	Liu et al. (2022)	Fang et al. (2019)	Shi et al. (2018)	Smart PKU Monitor (2024)
Domain	Traffic Monitoring	Edge Computing (General)	Traffic Congestion Detection	Health Monitoring (Edge Alerts)
Architecture	Hybrid (Edge + Cloud)	Edge-based MEC Scheduling	Edge-based Video Analytics	Edge-based Alerting System
Task Type	Congestion and Speed Detection	General Task Offloading	Congestion Detection (GFM)	Emergency Health Alerts
Scheduling / Task Management	Adaptive (bandwidth-based)	Latency-Aware Scheduling	Not addressed	Priority-Based Scheduling
Task Priority (High/Medium/Low)	Not implemented	Not implemented	Not implemented	Implemented
Deadline Handling	Not implemented	Not implemented	Not implemented	Conceptually Present
Overload Handling / Task Dropping	Not addressed	Not addressed	Not addressed	Present
Simulation / Testing	Real-world testbed	Algorithmic simulation	Visual Dataset Testing	Health Device Testing
Use of GFM (Foreground Modeling)	Indirectly via Shi et al.	Not used	Used	Not used
Traffic Application Specific	Yes	No	Yes	No

Literature SummaryTable

Research Work	Contribution	How We Extend or Modify It
Liu et al. (2021), IEEE T-ITS	Edge-based traffic video analytics	Add task priority + latency-aware scheduling
Fang et al. (2019), IEEE TCC	Latency-sensitive MEC scheduling	Customized to traffic tasks and deadlines
Smart PKU Monitor (2024)	Priority-based edge alerts	Adapted to traffic task categories
EdgeCloudSim (Sonmez et al., 2018)	Simulation of edge networks	Extended with task priority modules
Shi et al. (2018), Congestion Estimation	Frame occupancy logic for traffic alerts	Used threshold-based congestion trigger

TASK ALLOCATION

NAME	TASK ALLOCATED
SUSHIL V	• Configure EdgeCloudSim with priority-based scheduler • Implement task decision logic (drop, defer, cloud-offload) • Evaluate deadline success and CPU utilization metrics | • Foreground Detection (GFM/GMM)
GADDE PUNITH	• Speed Estimation using Optical Flow + Kalman • Congestion detection • Analyze metrics from simulations (latency, drop rate)

NAME	TASK ALLOCATED
BALAJI SAKTHIVEL M	• Emergency Vehicle detection logic (pattern + color) using OpenCV | • Compare with referenced studies (e.g., Liu et al., 2021) • Build tables, graphs and contribute to the "Results" and "State of the Art" sections |
UPPARA VEERANJANEYULU	• Traffic Flow detection • Define overall architecture (sensor/edge/cloud layers) • Diagram system design & task flow • Compile the report, future work, and manage reference formatting |

Summary:

Latency-Aware and Priority-Based Task Scheduling for Real-Time Traffic Monitoring Using Edge Computing

- —
- This project proposes an edge computing–based architecture for intelligent traffic monitoring, aimed at reducing latency and improving responsiveness by scheduling real-time tasks based on urgency and resource availability. Instead of sending traffic data to the cloud for analysis, our system utilizes edge nodes (such as Jetson Nano or Raspberry Pi) co-located with traffic cameras to process data closer to the source.
- The project is inspired by research published in IEEE Transactions on Intelligent Transportation Systems, particularly Liu et al. (2021), which demonstrated the advantages of edge processing in congestion and vehicle detection. We extend this by introducing a latency-aware and priority-based task scheduling mechanism that ensures high-priority tasks are processed within deadlines, even under system load.
- Tasks such as Emergency Vehicle Detection, Congestion Detection, Speed Estimation, and Vehicle Counting are assigned different priority levels. The scheduler uses a decision-making algorithm that considers task priority, latency budget, edge node utilization, and network delay. Tasks that cannot be executed locally within their deadlines are either offloaded to the cloud or dropped based on priority.
- We simulated the system using EdgeCloudSim, configuring 5 edge nodes under variable network conditions. The results showed significant improvements:
- The system integrates lightweight ML algorithms such as GFM for foreground detection, Optical Flow + Kalman Filter for speed tracking, and centroid tracking for traffic flow estimation. Experimental results confirmed the system’s efficiency under different traffic scenarios and weather conditions.

Q&A Session

- **When is a task dropped or deferred in your scheduler?**
- → A task is dropped or deferred when the system load is high and the task is either low-priority or its execution would violate the latency deadline.
- **How is task priority decided in your system?**
- → Tasks are categorized into High, Medium, and Low priority based on their urgency and criticality. For example, emergency vehicle detection is High, speed estimation is Medium, and routine vehicle counting is Low.
- **What metrics do you use to evaluate the system?**
- → We measure average task latency, deadline success rate, task drop rate, and edge node CPU utilization.
- **What are the limitations of your current system?**
- → The system doesn't yet include dynamic learning for priority adjustment. Also, traffic camera feeds are simulated; real deployment would require live camera integration and potentially GPU acceleration.
- **. Why is latency critical in traffic monitoring?**
- → In traffic systems, a delay in processing can lead to safety risks—such as failing to detect an accident or congestion in time. Real-time decision-making is essential for intelligent transport systems.

Thank You