Latency-Aware and Priority-Based Task Scheduling for Real-Time Traffic Monitoring UsingEdgeComputing

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

5.JUTURU NAGA ABHINAVA SAI – CB.SC.U4CSE23563

6.SREESHANTH REDDY POTHULA -CB.SC.U4CSE23640

UPDATES FROM REVIEW1

- Smart traffic model implementation using:
- 1.DOCKER
- 2.SIMPY
- 3.YAFS

DOCKER

- Docker is a containerization platform.
- Instead of running whole virtual machines, Docker packages an application + all its dependencies into a container that runs consistently on any computer.
- Containers share the host OS kernel but are isolated (like lightweight virtual machines).
- Each container can have its own CPU, memory, and network limits (so we can simulate weaker hardware like Raspberry Pi).

WORKING OF DOCKER

- You write a Dockerfile → defines the software environment (OS, libraries, Python, OpenCV, etc).
- Build → creates an image (template).
- Run → starts a container (instance of that image).
- You can run many containers in parallel, each emulating an edge device.
- Docker uses a network bridge so containers can talk to each other or to the host system.

HOW WE USED IT IN OUR PROJECT

- We use Docker to emulate edge devices (like Raspberry Pis) without buying many boards.
- Each container runs a Python program that:
- Reads video (or simulated tasks).
- Processes tasks locally if possible.
- Offloads tasks to the cloud (another container).
- By limiting CPU and memory per container (--cpus, --memory flags), we can simulate Raspberry Pi resource constraints.
- Helps us demonstrate multi-node scalability on a single laptop.

DOCKER SAMPLE RESULTS

SIMPY

- SimPy is a Python-based discrete-event simulation library.
- It lets you model systems where events happen over time (arrivals, processing, waiting in queues).
- You define resources (like CPU, queues, network) and events (like task arrival, task completion).
- SimPy then advances simulation time and logs what happens.
- How SimPy Works
- Everything runs inside a simulation environment (env).

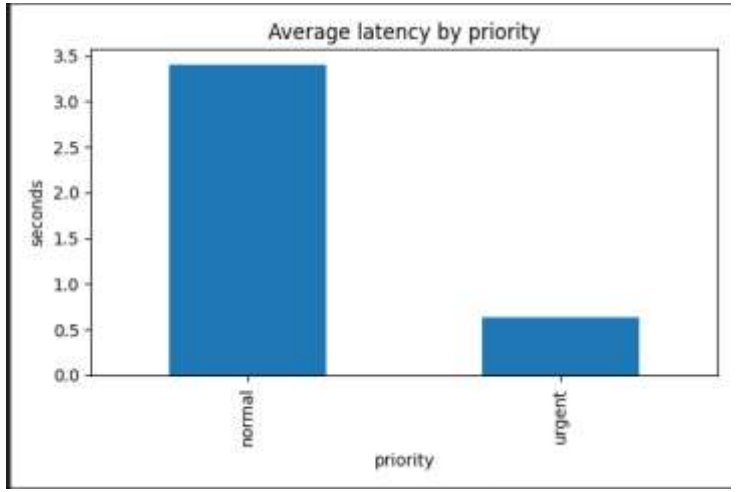
WORKING OF SIMPY

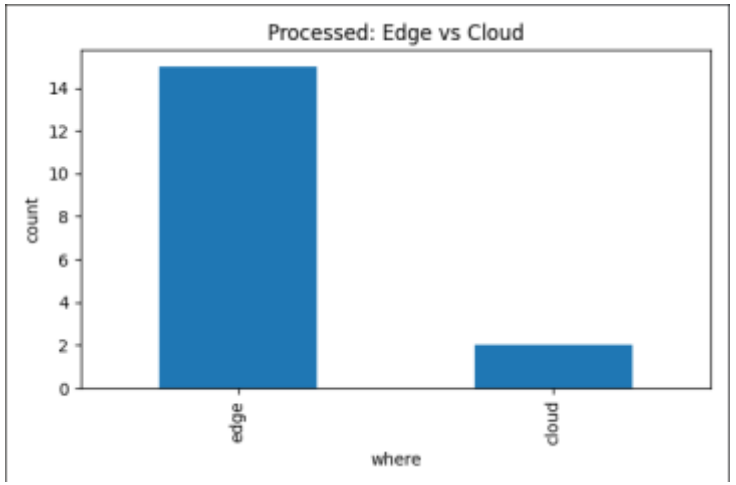
- Task arrivals (env.process(task_generator()))
- Queues (Store, Resource)
- CPU time (simulate service time based on MI/MIPS).
- The simulation can run much faster than real time — it doesn’t process actual video, just events.

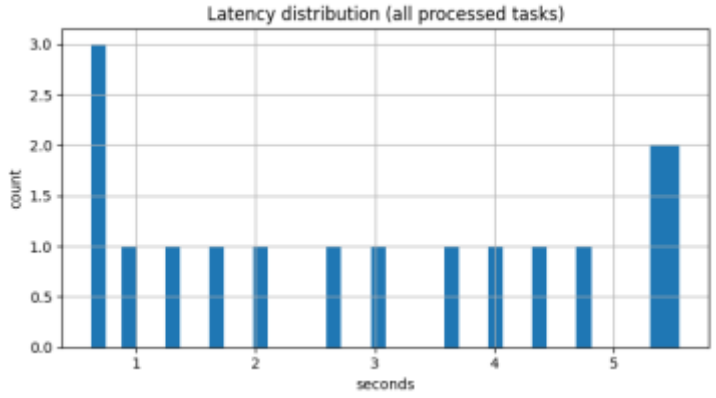
HOW WE USED IT IN OUR PROJECT

- SimPy is used to prototype our scheduler quickly:
- Define edge nodes and cloud as resources.
- Define tasks with priorities, deadlines, MI requirements.
- Run scheduling logic: process locally, offload, or drop.
- We also used SimPy to convert video frames into task events (by analyzing vehicles from video and mapping them into tasks).
- Output → CSV + graphs (latency histograms, edge vs cloud ratios).

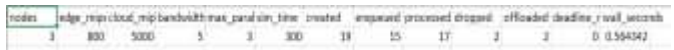
SIMPY SAMPLE RESULTS







task_id	where	priority	latency
t0	edge	normal	0.625
t1	edge	normal	0.983333
t2	edge	normal	1.341667
t3	edge	normal	1.7
t4	edge	normal	2.058333
t5	edge	normal	2.683333
t16	cloud	urgent	0.64
t17	cloud	urgent	0.64
t6	edge	normal	3.041667
t7	edge	normal	3.666667
t8	edge	normal	4.025
t9	edge	normal	4.383333
t10	edge	normal	4.741667
t11	edge	normal	5.366667
t13	edge	normal	5.458333
t15	edge	normal	5.55
t18	edge	normal	5.375



Edge Node	Queue Len	Processing
edge-0	0	0
edge-1	0	0
edge-2	8	1

Sim t=6.00s

Edge Node	Queue Len	Processing
edge-0	0	0
edge-1	0	0
edge-2	5	1

Sim t=9.00s

Edge Node	Queue Len	Processing
edge-0	0	0
edge-1	0	0
edge-2	1	1

```
Total created: 19
Enqueued: 15
Processed: 17
Dropped: 2
Offloaded: 2
Edge processed: 15
Deadline misses: 0
Priority normal: count=15 avg_latency=3.400s
Priority urgent: count=2 avg_latency=0.640s
```

YAFS(Yet Another Fog Simulator)

- YAFS is a specialized simulator for Fog/Edge/Cloud systems, built on top of SimPy.
- While SimPy is general-purpose, YAFS is domain-specific for network + application placement + fog computing.
- It already includes:
- Network topologies (graph of cloud/edge nodes + links).
- Message routing (through nodes/links with latency & bandwidth).
- Application modules (which can process tasks).
- Placement strategies (where to deploy modules).
- Metric collectors (latency, message delivery, resource usage).

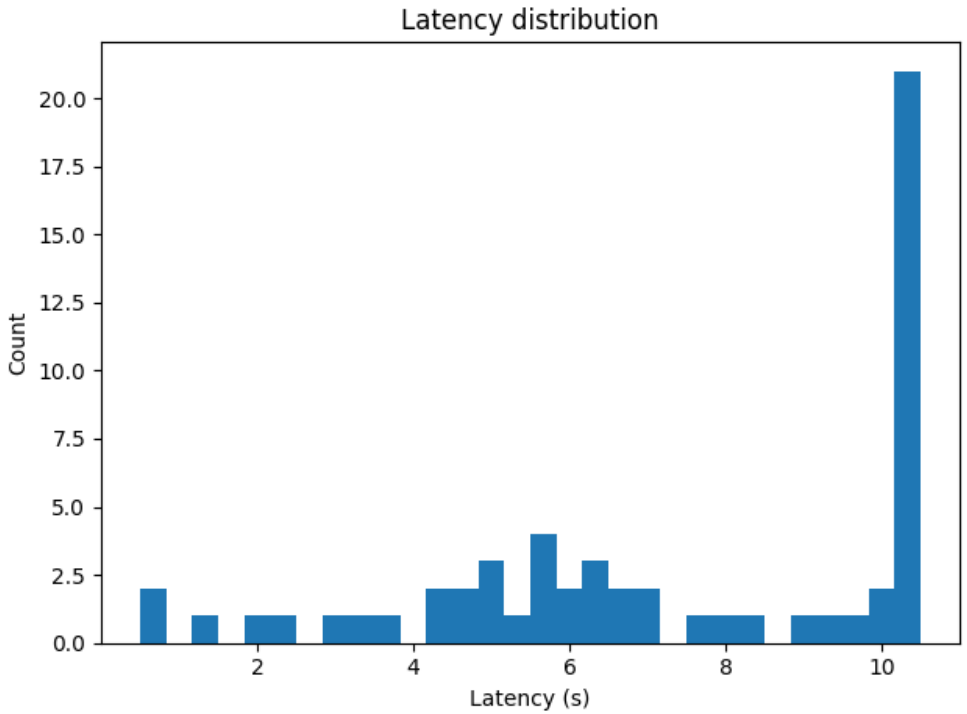
WORKING OF YAFS

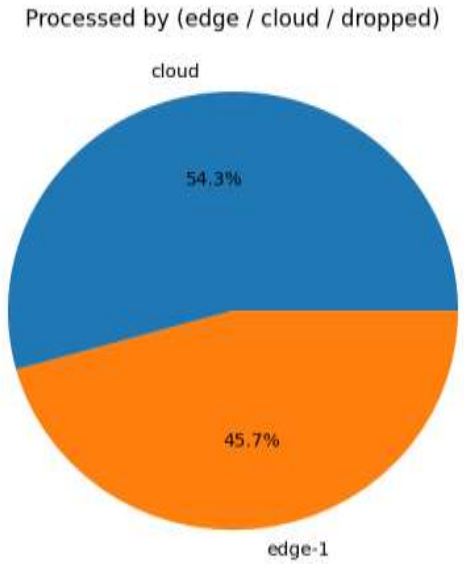
- Topology (network graph, node capacities, link bandwidth/latency).
- Application (modules like Sensor, Processor, Cloud).
- Placement (which node runs which module).
- Workload population (when and where messages/tasks are generated).
- YAFS uses SimPy internally to simulate all these events, but gives you ready-made data collectors.

HOW WE USED IT IN OUR PROJECT

- We use YAFS for validation and network-level experiments.
- After generating tasks (via SimPy + video input), we can feed them into YAFS to see:
- How network latency and bandwidth affect task deadlines.
- How different placement strategies (processing at edge vs cloud) affect performance.
- Collect standard metrics automatically (latency, deadline miss ratio, offload ratio).
- This makes our project scalable and scientifically valid, not just a demo.

YAFS SAMPLE RESULTS





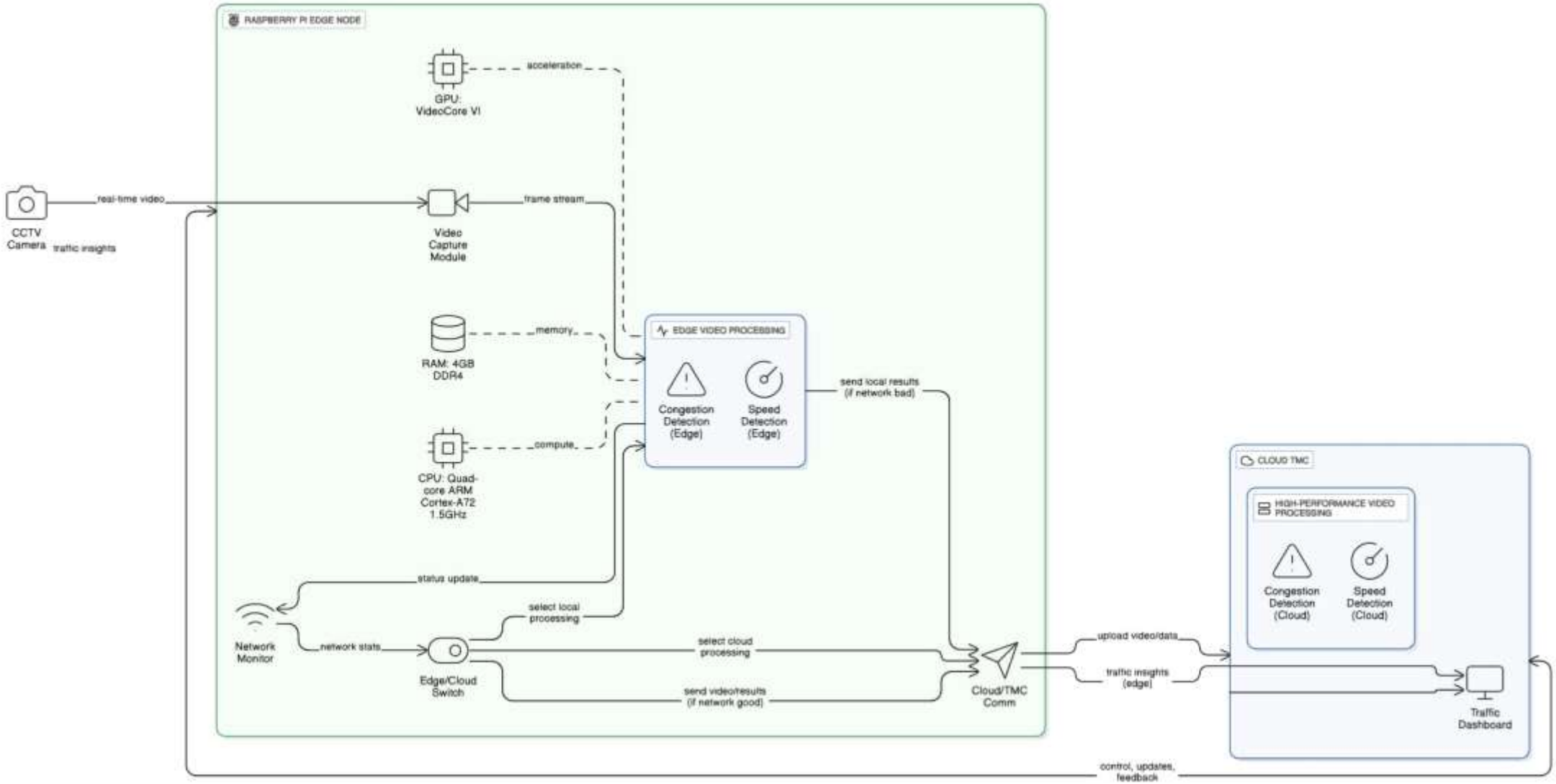
```
(simenv) PS C:\yafs_project_demo> python .\simpy_edge_cloud.py --tasks .\tasks.csv --nodes 3 --edge-mips 2000 --cloud-mi
ps 5000 --bandwidth_mbps 2 --max-queue 20 --sim-time 600 --offload-policy queue --out-log .\results_edge_new.csv
Starting sim for 600.0 seconds with 3 nodes...
Simulation finished.
Wrote results to .\results_edge_new.csv
Saved latency_hist.png
Saved ratios.png
Done.
(simenv) PS C:\yafs_project_demo>
```

results_edge_new.csv

task_id	created_at	queued_at	started_at	finished_at	processed	scheduling	mi	data_mb	latency_s
t0	0	0	0	0.5	edge-1	enqueued	1000	0.05	0.5
t1	0.167	0.167	0.5	1	edge-1	enqueued	1000	0.05	0.833
t2	0.167	0.167	1	1.5	edge-1	enqueued	1000	0.05	1.333
t3	0.167	0.167	1.5	2	edge-1	enqueued	1000	0.05	1.833
t4	0.167	0.167	2	2.5	edge-1	enqueued	1000	0.05	2.333
t5	0.167	0.167	2.5	3	edge-1	enqueued	1000	0.05	2.833
t6	0.334	0.334	3	3.5	edge-1	enqueued	1000	0.05	3.166
t7	0.334	0.334	3.5	4	edge-1	enqueued	1000	0.05	3.666
t8	0.334	0.334	4	4.5	edge-1	enqueued	1000	0.05	4.166
t9	0.5	0.5	4.5	5	edge-1	enqueued	1000	0.05	4.5
t10	0.5	0.5	5	5.5	edge-1	enqueued	1000	0.05	5
t11	0.667	0.667	5.5	6	edge-1	enqueued	1000	0.05	5.333
t12	0.834	0.834	6	6.5	edge-1	enqueued	1000	0.05	5.666
t13	0.834	0.834	6.5	7	edge-1	enqueued	1000	0.05	6.166
t14	1.001	1.001	7	7.5	edge-1	enqueued	1000	0.05	6.499
t15	1.001	1.001	7.5	8	edge-1	enqueued	1000	0.05	6.999
t36	7.841	8.041		8.241	cloud	offloaded	1000	0.05	
t37	7.841	8.041		8.441	cloud	offloaded	1000	0.05	
t16	4.171	4.171	8	8.5	edge-1	enqueued	1000	0.05	4.329
t38	7.841	8.041		8.641	cloud	offloaded	1000	0.05	
t40	8.008	8.208		8.841	cloud	offloaded	1000	0.05	
t17	4.338	4.338	8.5	9	edge-1	enqueued	1000	0.05	4.662
t41	8.175	8.375		9.041	cloud	offloaded	1000	0.05	
t42	8.175	8.375		9.241	cloud	offloaded	1000	0.05	
t43	8.341	8.541		9.441	cloud	offloaded	1000	0.05	
t18	4.338	4.338	9	9.5	edge-1	enqueued	1000	0.05	5.162
t44	8.341	8.541		9.641	cloud	offloaded	1000	0.05	
t46	8.508	8.708		9.841	cloud	offloaded	1000	0.05	
t19	4.504	4.504	9.5	10	edge-1	enqueued	1000	0.05	5.496
t47	8.675	8.875		10.041	cloud	offloaded	1000	0.05	
t48	8.675	8.875		10.241	cloud	offloaded	1000	0.05	
t49	8.842	9.042		10.441	cloud	offloaded	1000	0.05	
t20	4.671	4.671	10	10.5	edge-1	enqueued	1000	0.05	5.829
t50	8.842	9.042		10.641	cloud	offloaded	1000	0.05	
t52	9.176	9.376		10.841	cloud	offloaded	1000	0.05	
t21	4.838	4.838	10.5	11	edge-1	enqueued	1000	0.05	6.162
t53	9.343	9.543		11.041	cloud	offloaded	1000	0.05	
t54	9.343	9.543		11.241	cloud	offloaded	1000	0.05	
t56	9.509	9.709		11.441	cloud	offloaded	1000	0.05	
t22	5.005	5.005	11	11.5	edge-1	enqueued	1000	0.05	6.495
t57	9.676	9.876		11.641	cloud	offloaded	1000	0.05	
t58	9.676	9.876		11.841	cloud	offloaded	1000	0.05	
t23	6.84	6.84	11.5	12	edge-1	enqueued	1000	0.05	5.16
t59	9.843	10.043		12.041	cloud	offloaded	1000	0.05	
t60	9.843	10.043		12.241	cloud	offloaded	1000	0.05	
t65	11.845	12.045		12.441	cloud	offloaded	1000	0.05	
t24	6.84	6.84	12	12.5	edge-1	enqueued	1000	0.05	5.66
t67	12.179	12.379		12.641	cloud	offloaded	1000	0.05	
t68	12.346	12.546		12.841	cloud	offloaded	1000	0.05	
t25	7.007	7.007	12.5	13	edge-1	enqueued	1000	0.05	5.993
t69	12.346	12.546		13.041	cloud	offloaded	1000	0.05	
t71	12.679	12.879		13.241	cloud	offloaded	1000	0.05	
t72	12.846	13.046		13.441	cloud	offloaded	1000	0.05	
t26	7.174	7.174	13	13.5	edge-1	enqueued	1000	0.05	6.326
t27	7.174	7.174	13.5	14	edge-1	enqueued	1000	0.05	6.826
t28	7.341	7.341	14	14.5	edge-1	enqueued	1000	0.05	7.159
t76	14.347	14.547		14.747	cloud	offloaded	1000	0.05	
t29	7.341	7.341	14.5	15	edge-1	enqueued	1000	0.05	7.659
t78	14.681	14.881		15.081	cloud	offloaded	1000	0.05	
t79	14.848	15.048		15.281	cloud	offloaded	1000	0.05	
t80	14.848	15.048		15.481	cloud	offloaded	1000	0.05	
t30	7.507	7.507	15	15.5	edge-1	enqueued	1000	0.05	7.993

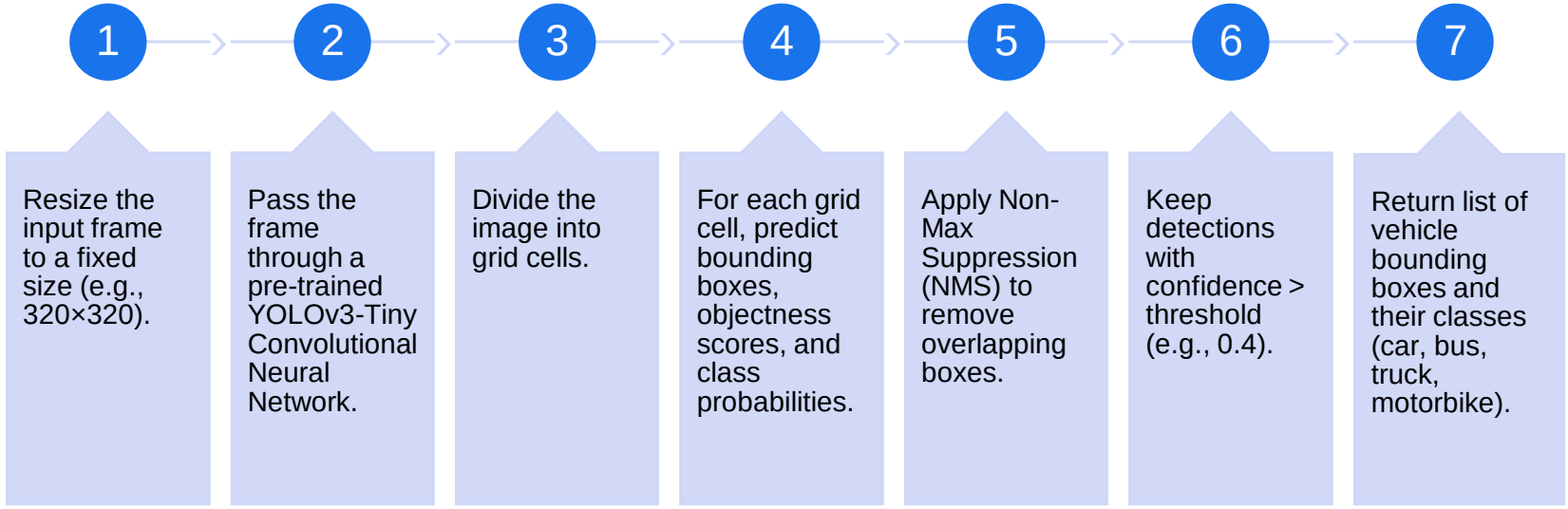
t82	15.182	15.382		15.681	cloud	offloaded	1000	0.05	
t83	15.348	15.548		15.881	cloud	offloaded	1000	0.05	
t31	7.507	7.507	15.5	16	edge-1	enqueued	1000	0.05	8.493
t84	15.348	15.548		16.081	cloud	offloaded	1000	0.05	
t85	15.348	15.548		16.281	cloud	offloaded	1000	0.05	
t86	15.348	15.548		16.481	cloud	offloaded	1000	0.05	
t32	7.507	7.507	16	16.5	edge-1	enqueued	1000	0.05	8.993
t88	15.515	15.715		16.681	cloud	offloaded	1000	0.05	
t89	15.515	15.715		16.881	cloud	offloaded	1000	0.05	
t33	7.674	7.674	16.5	17	edge-1	enqueued	1000	0.05	9.326
t90	15.682	15.882		17.081	cloud	offloaded	1000	0.05	
t91	15.849	16.049		17.281	cloud	offloaded	1000	0.05	
t93	16.183	16.383		17.481	cloud	offloaded	1000	0.05	
t34	7.674	7.674	17	17.5	edge-1	enqueued	1000	0.05	9.826
t94	16.183	16.383		17.681	cloud	offloaded	1000	0.05	
t95	16.35	16.55		17.881	cloud	offloaded	1000	0.05	
t35	7.674	7.674	17.5	18	edge-1	enqueued	1000	0.05	10.326
t96	16.35	16.55		18.081	cloud	offloaded	1000	0.05	
t97	16.35	16.55		18.281	cloud	offloaded	1000	0.05	
t99	16.516	16.716		18.481	cloud	offloaded	1000	0.05	
t39	8.008	8.008	18	18.5	edge-1	enqueued	1000	0.05	10.492
t100	16.516	16.716		18.681	cloud	offloaded	1000	0.05	
t101	16.516	16.716		18.881	cloud	offloaded	1000	0.05	
t45	8.508	8.508	18.5	19	edge-1	enqueued	1000	0.05	10.492
t102	16.683	16.883		19.081	cloud	offloaded	1000	0.05	
t103	16.683	16.883		19.281	cloud	offloaded	1000	0.05	
t104	16.85	17.05		19.481	cloud	offloaded	1000	0.05	
t51	9.009	9.009	19	19.5	edge-1	enqueued	1000	0.05	10.491
t105	16.85	17.05		19.681	cloud	offloaded	1000	0.05	
t107	17.017	17.217		19.881	cloud	offloaded	1000	0.05	
t55	9.509	9.509	19.5	20	edge-1	enqueued	1000	0.05	10.491
t108	17.184	17.384		20.081	cloud	offloaded	1000	0.05	
t109	17.184	17.384		20.281	cloud	offloaded	1000	0.05	
t110	17.351	17.551		20.481	cloud	offloaded	1000	0.05	
t61	10.01	10.01	20	20.5	edge-1	enqueued	1000	0.05	10.49
t111	17.351	17.551		20.681	cloud	offloaded	1000	0.05	
t113	17.684	17.884		20.881	cloud	offloaded	1000	0.05	
t62	10.51	10.51	20.5	21	edge-1	enqueued	1000	0.05	10.49
t114	17.684	17.884		21.081	cloud	offloaded	1000	0.05	
t115	17.851	18.051		21.281	cloud	offloaded	1000	0.05	
t116	17.851	18.051		21.481	cloud	offloaded	1000	0.05	
t63	11.511	11.511	21	21.5	edge-1	enqueued	1000	0.05	9.989
t118	18.018	18.218		21.681	cloud	offloaded	1000	0.05	
t119	18.018	18.218		21.881	cloud	offloaded	1000	0.05	
t64	11.678	11.678	21.5	22	edge-1	enqueued	1000	0.05	10.322
t120	18.185	18.385		22.081	cloud	offloaded	1000	0.05	
t121	18.185	18.385		22.281	cloud	offloaded	1000	0.05	
t122	18.185	18.385		22.481	cloud	offloaded	1000	0.05	
t66	12.012	12.012	22	22.5	edge-1	enqueued	1000	0.05	10.488
t123	18.185	18.385		22.681	cloud	offloaded	1000	0.05	
t124	18.351	18.551		22.881	cloud	offloaded	1000	0.05	
t70	12.512	12.512	22.5	23	edge-1	enqueued	1000	0.05	10.488
t126	18.518	18.718		23.081	cloud	offloaded	1000	0.05	
t73	13.68	13.68	23	23.5	edge-1	enqueued	1000	0.05	9.82
t74	13.68	13.68	23.5	24	edge-1	enqueued	1000	0.05	10.32
t75	14.181	14.181	24	24.5	edge-1	enqueued	1000	0.05	10.319
t77	14.514	14.514	24.5	25	edge-1	enqueued	1000	0.05	10.486
t81	15.015	15.015	25	25.5	edge-1	enqueued	1000	0.05	10.485
t87	15.515	15.515	25.5	26	edge-1	enqueued	1000	0.05	10.485
t92	16.016	16.016	26	26.5	edge-1	enqueued	1000	0.05	10.484
t98	16.516	16.516	26.5	27	edge-1	enqueued	1000	0.05	10.484
t106	17.017	17.017	27	27.5	edge-1	enqueued	1000	0.05	10.483
t112	17.517	17.517	27.5	28	edge-1	enqueued	1000	0.05	10.483
t117	18.018	18.018	28	28.5	edge-1	enqueued	1000	0.05	10.482
t125	18.518	18.518	28.5	29	edge-1	enqueued	1000	0.05	10.482

Updated Architecture Diagram



Vehicle Detection (YOLOv3-Tiny)

INPUT:
A video frame
OUTPUT:
List of detected vehicles with bounding boxes and classes



Foreground Detection (Gaussian-based Foreground Model – GFM)

1. For each frame f ∈ V do
2. Convert the frame to grayscale.
3. Compute the current background model B using running average (GFM).
4. For each pixel (x, y) in frame f do
   - If |f(x, y) − B(x, y)| > threshold then
     → F(x, y) ← 1 (Foreground)
   - Else
     → F(x, y) ← 0 (Background)
5. End for
6. Return F (binary foreground mask)

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

Congestion Detection (CNN + Coverage Ratio)

1. For each frame f ∈ V do
2. Compute Mg and Mz in the Region of Interest (ROI).
3. Calculate congestion area Ac:
   - For each pixel (x, y) in ROI do
     - If Mg(x,y) == 255 AND Mz(x,y) == 0 then
       → Ac += 1
4. Compare Ac to area threshold τ_a.
5. If Ac > τ_a for duration τ_t (e.g., 5 seconds) then
   → Return True (Congestion Detected)
   Else
   → Return False

- **INPUT:**
  Video frame and foreground masks Mg (GFM) and Mz (Zivkovic KNN)
- **OUTPUT:**
  Boolean congestion flag

Emergency Vehicle Detection (HSV + Flash Pattern Logic)

1. For each frame f ∈ V do
2. Convert frame f to HSV color space.
3. Extract red regions R using HSV thresholds.
4. Extract blue regions B using HSV thresholds.
5. Combine R and B into mask M = R ∪ B.
6. Apply morphological filters to reduce noise in M.
7. For each detected bounding box in M do
   - If box size > threshold and position is near top of vehicle then
     → Append timestamp to flash_sequence.
8. If flash_sequence shows periodic flashes (≥3 flashes in 2 seconds) then
   → E ← 1 (Emergency detected)
   Else
   → E ← 0
9. End for
10. Return E

- **INPUT:**
  A video frame and YOLO vehicle bounding boxes
- **OUTPUT:**
  Emergency detection flag E

Speed Estimation (Optical Flow + Kalman Filter)

1. For each frame f ∈ V do
2. For each detected vehicle bounding box:
   - Track feature points using Optical Flow (Lucas-Kanade).
3. For each tracked point:
   - Calculate displacement in pixels between frames.
4. Calculate speed in m/s:
   - (Pixel displacement × meters_per_pixel) ÷ time difference.
5. Apply Kalman Filter to smooth speed estimates.
6. Convert speed to km/h.
7. Return list of vehicle speeds.

- **INPUT:**
  Consecutive video frames and detected vehicle bounding boxes
- **OUTPUT:**
  Speeds of vehicles (km/h)

Traffic Flow Estimation (Centroid Tracking + Line Crossing)

1. For each frame f ∈ V do
2. Track vehicle centroids from YOLO bounding boxes.
3. For each vehicle track:
   - Check if centroid crosses the virtual line (e.g., at y = 320 pixels).
4. If centroid crosses line and not counted before →
   → Increment vehicle count.
5. Return cumulative vehicle count.

- **INPUT:**
  Video frame and detected vehicle bounding boxes
- **OUTPUT:**
  Vehicle count (vehicles per minute)

Scheduler Algorithm

1. Sort incoming tasks Q by descending priority.
2. For each task t ∈ Q do:
    a. If CPU usage < threshold and estimated latency ≤ deadline(t) →
        → Execute task locally on Raspberry Pi.
    b. Else if Cloud is reachable and delay is tolerable →
        → Offload task t to cloud.
    c. Else →
        → Defer or drop the task based on urgency.
3. Repeat for all tasks.
4. Return execution decisions.

- **INPUT:**
  Set of tasks Q with attributes (priority, CPU load, deadlines)
- **OUTPUT:**
  Decision to execute locally, offload to cloud, or defer/drop

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