LoPECs

A Low-Power Edge Computing System for Real-Time Autonomous Driving Services

Problem Statement

Autonomous vehicles must run multiple intelligent services—such as object detection, SLAM, and voice recognition—in real time. However, deploying these on low-power embedded systems like Jetson TX1 is challenging due to limited compute and energy resources. While cloud-based processing can offload heavy tasks, it suffers from latency and network dependency, which is risky in safety-critical environments.

To solve this, there is a need for a lightweight, adaptive edge computing system that can dynamically offload tasks between the vehicle and nearby edge servers, balancing real-time performance and energy efficiency. This project explores LoPECS: a Low-Power Edge Computing System designed to address these challenges in real-world autonomous driving scenarios.

Why it is Edge

1. Real-Time Responsiveness Required
Autonomous systems must react instantly to events like obstacles or lane changes — relying on the cloud introduces unsafe latency.

2. Insufficient Onboard Compute Power
Low-cost embedded devices (e.g., Jetson TX1) cannot handle all AI workloads alone due to limited CPU/GPU and battery capacity.

3. Proximity of Edge Servers Enables Faster Processing
Edge servers, located closer than cloud data centers, reduce round-trip time for heavy tasks, improving both speed and energy usage.

4. Intelligent Offloading Based on Context
The system must continuously decide where to run each task (local vs. edge) by monitoring network quality, power levels, and system load

5. Multi-Objective Optimization Challenge
The system must optimize multiple conflicting parameters—such as latency, power, and reliability—which is a core edge computing challenge.

Objectives and Features

- Ultra-Low Power Consumption
  Design a system that can execute real-time autonomous driving services on embedded devices using as little as ~11W power.

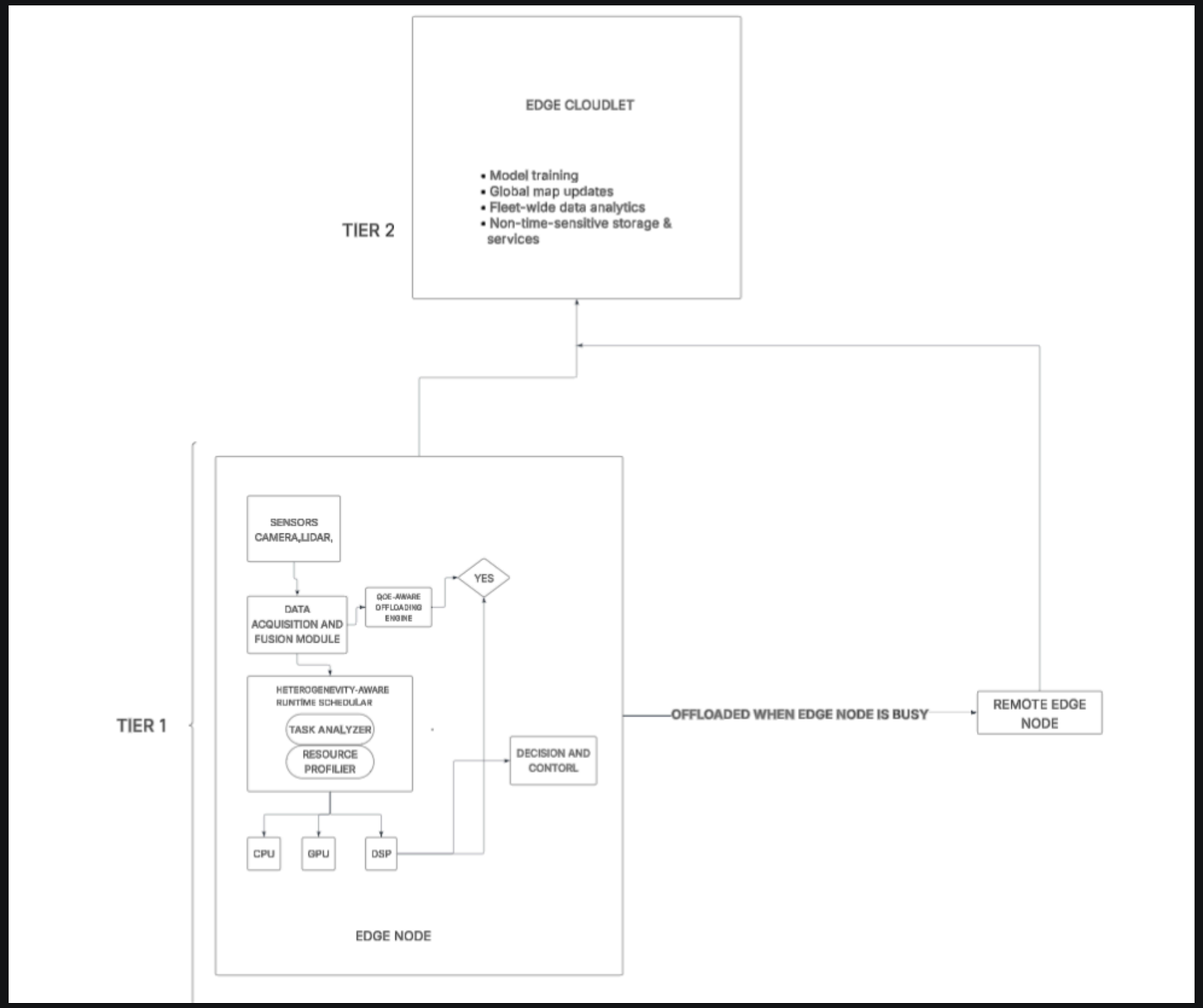
- Real-Time Multi-Service Execution
  Support multiple AI workloads (SLAM, object detection, speech) concurrently with minimal delay.

- Heterogeneity-Aware Runtime Scheduling
  Leverage CPU, GPU, and other compute units dynamically based on workload type and system state.

- A Dynamic Edge Offloading
  Enable real-time task offloading to nearby edge servers to balance energy and performance trade-offs.

- Lightweight OS and Communication Framework
  Replace heavy ROS with a custom π-OS to reduce overhead and support low-latency inter-service communication.

Architecture diagram



Tier 1 (edge node)

-present inside the vehicle, performs real time critical decision making

Components:

•Sensors (Camera, LiDAR, etc.)

Collect raw environment data in real time.

•Data Acquisition & Fusion Module

Combines multi-sensor inputs to form a coherent understanding of surroundings.

•Heterogeneity-Aware Runtime Scheduler

Efficiently assigns tasks to:

•CPU – Control logic

•GPU – Parallel tasks (e.g. image processing)

•DSP – Signal processing

•QoE-Aware Offloading Engine

Decides if and when tasks should be offloaded based on:

•Latency

•Bandwidth

•Load status

•Task urgency

•Decision & Control

Executes critical decisions locally (e.g., brake, turn, alert).

✅ Main Functions

🔹 **Ultra-Low Latency Response**

→ Executes real-time tasks for immediate driving decisions.

🔹 **Real-Time Object Detection & Motion Planning**

→ Detects obstacles and plans routes instantly based on sensor data.

🔹 **Safety-Critical Logic Execution**

→ Handles emergency braking, steering, and other critical functions.

🔹 **Local Resource Management**

→ Efficiently utilizes onboard processors (CPU, GPU, DSP) for balanced performance.

Tier 1(remote edge server)

- Located near road infrastructure or RSUs; connected via 5G/V2X

Components:

•Model Training (incremental updates)

Trains or fine-tunes models using recent vehicle data

•Global Map Updates

Maintains high-definition global/local maps based on inputs from multiple vehicles

•Fleet-Wide Data Analytics

Aggregates and analyses data across many vehicles for better predictions

•Non-Time-Sensitive Storage & Services

Stores logs, performance stats, non-critical video feeds

✅Main Functions

🔹 **Offloads Heavy but Non-Urgent Tasks**

→ Handles tasks like model updates and analytics without delaying real-time decisions.

🔹 **Enables Collaborative AI Learning**

→ Shares insights across vehicles for improved collective intelligence.

🔹 **Enhances Environmental Awareness**

→ Analyzes data from nearby vehicles to inform decisions in local regions.

🔹 **Reduces Load on Vehicle Systems**

→ Prevents Tier 1 overload by handling secondary processing at the edge.

•

Tier 2(edge cloudlet)

-Farther away — used when both edge node and cloudlet are overloaded or for deep analytics

Components:

•Deep Model Training on Massive Datasets

Uses GPU/TPU clusters to retrain full models using data from the entire fleet

•Data Warehousing

Stores long-term data, logs, and training datasets

•OTA (Over-The-Air) Updates

Pushes model updates, firmware, or map updates back to vehicles or cloudlets

•Policy and Strategy Updates

Centralized tuning of algorithms and driving policies

Main Functions:

•Global intelligence and AI model refinement
•Historical data processing and trend analysis
•Strategic planning and improvement over time
•Least preferred for real-time due to high latency

Justification – How Barriers Were Solved

✅Manual Configuration for Services

Pre-configured service templates and scripts are used. Once deployed, most services run autonomously without manual intervention.

✅Uncertainty in Deploying on Heterogeneous Devices

All models are converted to device-agnostic formats like ONNX, TensorRT, and TFLite. This ensures broad compatibility with minimal effort.

✅Scalability Concerns for Future Projects

While LoPECS isn’t plug-and-play like ROS, it’s scalable within its mission domain. Additional services can be added via containerized deployments.

✅Limited Access to Debugging Tools

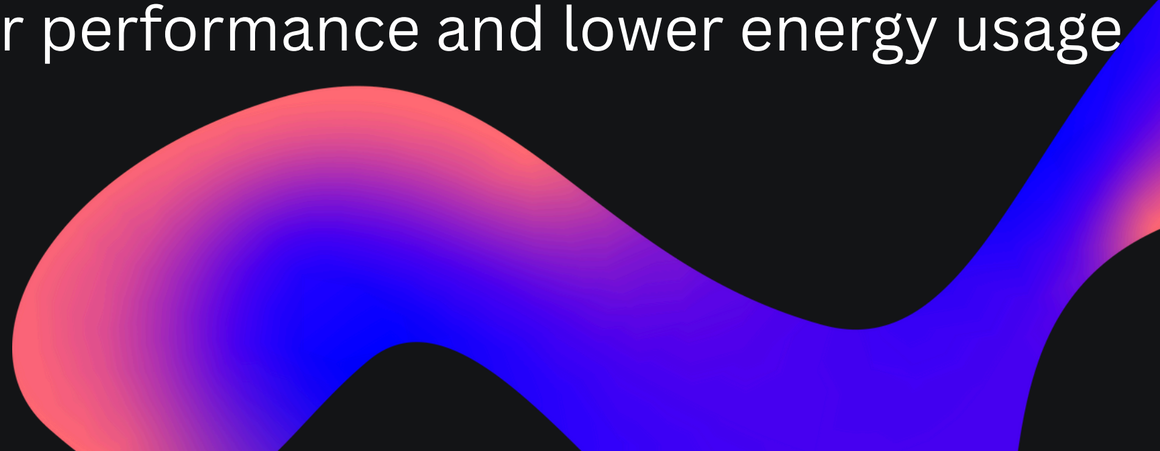
Debugging is done using lightweight logging and modular testing. The small scope allows for controlled testing environments and reduces tool dependency.

✅π-OS lacks the rich libraries, debugging tools, and drivers that a standard middleware like ROS offers

LoPECS is a highly focused system with a fixed set of services (SLAM, object detection, speech). The team only needed lightweight, custom inter-service communication, so full ROS capabilities were not required. This reduces bloat and improves performance.

✅Adding new services or modules in π-OS may require manual coding and integration, unlike plug-and-play ROS packages

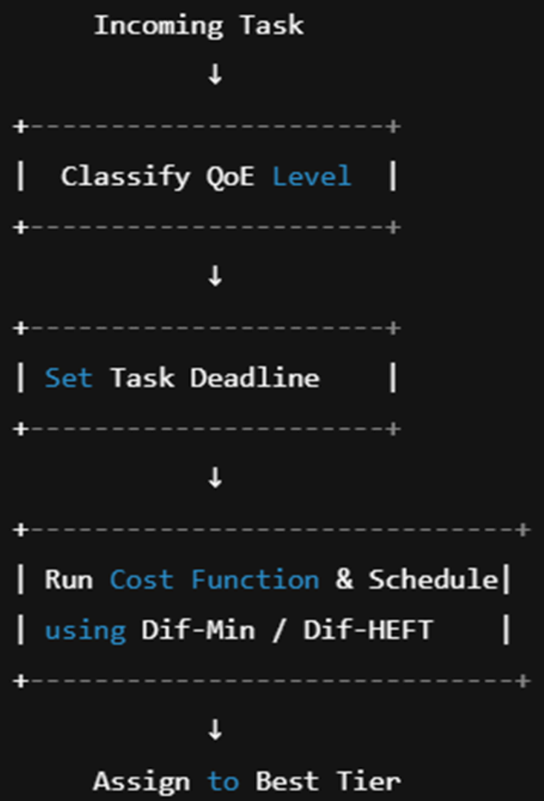
LoPECS is designed for mission-specific edge applications (e.g., in campuses, factories).The scope of supported services is well-defined and stable. The overhead of extensibility is acceptable in exchange for better performance and lower energy usage



STATE OF THE ART

YEAR	APPROACH/TECHNOLOGY	KEY FEAUTRES	IMPACT
2018	Neurosurgeon (Split DNN)	Layer-wise partitioning between edge & cloud.	Reduced latency & energy via adaptive splitting.
2019	YOLOv3-Tiny on Edge	Lightweight CNN for embedded deployment.	Real-time object detection on low-power devices.
2020	LoPECS Architecture	3-layer edge hierarchy with smart offloading.	Optimized compute-location decisions.
2021	EdgeTPU + Federated Learning	On-device model updates without central data storage.	Privacy-preserving real-time learning.
2023	YOLOv5n, MobileNetV3	Ultra-light object detection for Jetson/RPi.	High FPS with reduced power and memory footprint.

Implementation of LoPECS



Our Implementation focus :

- QoE-Based Task Classification
- Workload Estimation
- Heterogeneity-Aware Scheduling
- Smart Tier Selection

Quality of Experience (QoE):

Measure how well system meets latency, energy and accuracy

```
if task.latency < 10ms:
    task.qoe_type = 'QoE-Time'
elif task.energy > threshold:
    task.qoe_type = 'QoE-Energy'
else:
    task.qoe_type = 'QoE-Accuracy'
```

Heterogeneity-Aware Runtime Design

- **Two Scheduling Algorithms:**
  - **Dif-Min**: Faster, greedy heuristic
  - **Dif-HEFT**: Optimized, accuracy-oriented
- **Workflow:**
- Model delay, energy for each task-device pair
- Use cost function:

  Cost : w1 * computational delay + w2 * communication delay
- Computational delay – processing time.
- Communication delay – data transfer between edge nodes.
- Weight w1 and w2.

  Select node with minimum cost under QoE latency.

```
for task in Tasks:
    min_cost = ∞
    selected_node = None

    for node in Nodes:
        # Step 1: Compute delays
        comp_delay = task.workload / node.processing_speed
        comm_delay = task.data_size / node.network_bandwidth + node.network_latency

        # Step 2: Compute weighted cost
        cost = W1 * comp_delay + W2 * comm_delay

        # Step 3: Check QoE latency requirement
        if cost <= task.qoe_latency:
            if cost < min_cost:
                min_cost = cost
                selected_node = node

    if selected_node:
        assign(task, selected_node)
    else:
        handle_failure(task)  # Fallback strategy
```

Visualization and monitoring :

•**Live Dashboard** (Conceptual):
•Task placement (vehicle ↔ edge)
•Delay / energy trends
•Makespan comparison (Naive vs Dif-Min/Dif-HEFT)

•**Result Evaluation**:
•Bar charts (e.g., latency, energy for each task)
•Timeline graphs of task execution across nodes

Tools You Can Use:
•**Simulation**: Python with Matplotlib for scheduling comparison
•**Deployment (Optional)**: Jetson board for real-time SLAM/SSD task execution

Block diagram:
Input Task → QoE Classifier → Scheduler → Runtime Monitor → Selected Node → Feedback loop

Task Split

Darshan R Onboard edge module	Prasanth PM Remote Node Module
Kapilan V Implementation and Testing	Jeevan Raj Action Module

Questions and Answers:

Question 1:Are you guys using software simulation or physical implementation If software simulation what are the key tools used?

Answer 1:

We are simulating using software services and the key tools used are

- Networkx
- Python,Matplotlib and Numpy
- YOLO
- OPEN CL

Question 2:How is the nearest remote node connection established when the local onboard edge is busy?

Answer 2:

The QoE-Aware Offloading Engine continuously evaluates **latency**, **bandwidth**, **load**, and **urgency**, then uses 5G/V2X connectivity to offload tasks with the **minimum cost** node selected by the scheduler (Dif-Min or Dif-HEFT).The LoPECS architecture includes a **vehicle-edge coordinator** that dynamically offloads tasks to nearby cloudlets based on network conditions and energy considerations

Question 3:How did you guys come up with the power consumption value of 11w?

Answer 3:

The **~11 W** number comes from **measured power consumption** during real-world LoPECS tests on an NVIDIA Jetson TX1.

In the research setup, they:

- Ran the target workloads (**SLAM**, **object detection**, **speech recognition**) concurrently on the Jetson.
- Used a **power meter** or Jetson's onboard monitoring tools (like tegrastats) to log the system's actual wattage under load.
- Optimized the runtime with **π-OS**, **heterogeneity-aware scheduling**, and **task offloading** so that even with all services active, average power draw stayed around **11 W** instead of the typical 12–15 W for similar setups.