Edge AI Surveillance on a Raspberry Pi Cluster

Nauman Iftikhar, Abdur Rahim Nishad, Negar Mohammadi, Ikbela Halili, Muhammad Abdullah Nagori, Muhammad Saleem, Muhammad Furqan Shafique, Krish Tamboli

Frankfurt University of Applied Sciences

Introduction

EdgeGuard is a self-contained edge-computing cluster built from commodity Raspberry Pi hardware. A Pi 4 with a Sony IMX500 AI Camera streams video over ZMQ to a Pi 5 master node, which runs YOLOv8n face-coverage detection on a Hailo-8L NPU (14–17ms/frame). Eight diskless Pi 3B+ nodes (32 cores) boot over PXE/NFS and double as an MPI / HPL benchmarking testbed, letting us characterise the cluster's strong and weak scaling under Amdahl's and Gustafson's Laws.

Cluster Architecture

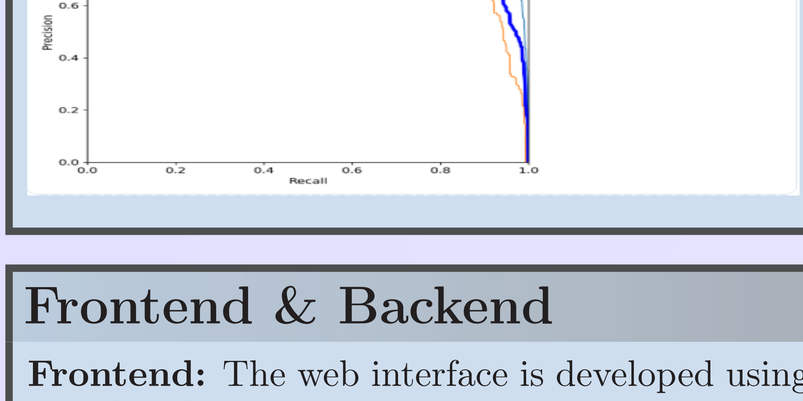
Infrastructure: 10-node heterogeneous Raspberry Pi topology networked over a dedicated 100 Mbps switched local area network (10.10.10.0/24) via two TP-Link TL-SG108E managed switches to completely isolate intensive intra-cluster compute traffic.

Master Node (Pi 5): Primary cluster controller (10.10.10.1) running an 8 GB ARM A76 processor. Hosts the k3s Kubernetes control plane, core gateway routing, and critical network infrastructure automated via `dnsmasq` (DHCP allocation, TFTP boot file delivery, and centralized NFS client storage hosting).

Sensor Node (Pi 4): Edge sensing element (10.10.10.40) running 4 GB RAM. Outfitted with a 1080p Raspberry Pi AI Camera Module featuring an onboard Sony IMX500 sensor, executing localized frame captures and serializing high-frequency image metadata pipelines over a low-latency ZeroMQ PubSub topology.

Worker Nodes (8× Pi 3B+): Distributed 32-core high-performance compute pool (10.10.10.21–28). Nodes `pi3-01` to `pi3-05`, `pi3-07`, and `pi3-08` boot entirely diskless over the network via PXE firmware using separate root filesystems exported from the Pi 5.

Performance Baseline Node: Node `pi3-06` (10.10.10.26) intentionally boots from a volatile local SD card, serving as an isolated control variable to directly compare network-attached filesystem latency against native physical flash storage options during load testing.



Benchmark Results

Figure 1: MPI Example 1 — Monte Carlo Pi

Figure 2: MPI Example 2 — Matrix Multiply

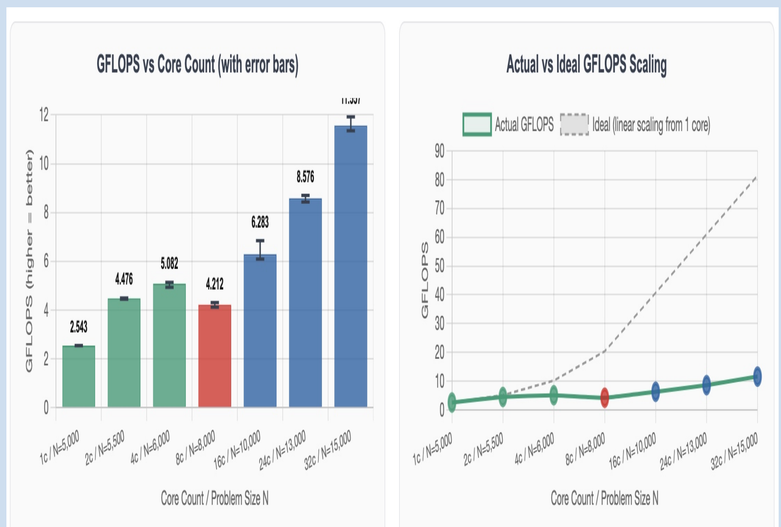


Figure 3: HPL Benchmark Performance (GFLOPS)

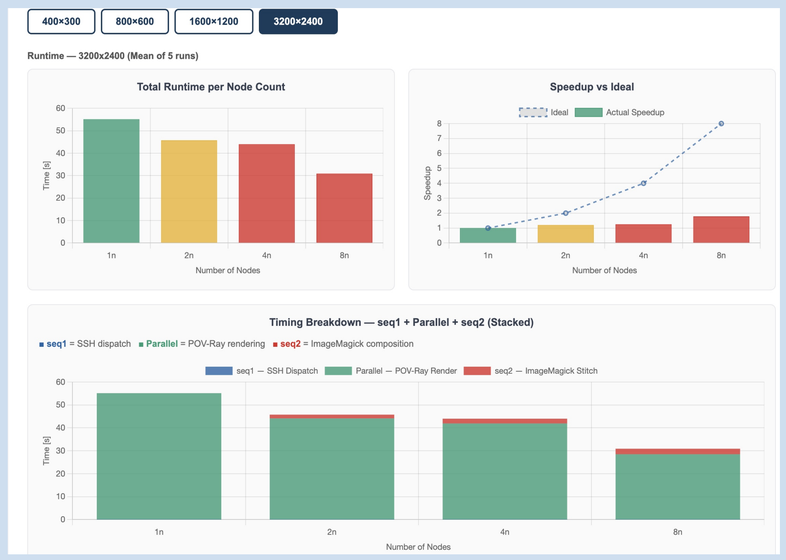
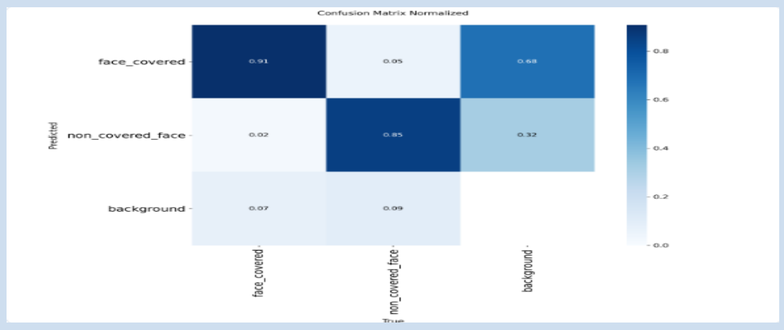


Figure 4: Non-MPI Task Scaling Results

Object Detection & AI Inference

The surveillance pipeline captures video on a Raspberry Pi 4 sensor node and streams it via ZeroMQ to a Raspberry Pi 5 master node. There, a dedicated Hailo-8L NPU executes binary face-coverage classification, automatically logging suspicious events to a PostgreSQL database and triggering real-time Telegram alerts.



PXE Diskless Boot

Benefits:

- **Centralized Admin:** Update all 7 workers simultaneously by editing one NFS root.
- **No SD Failures:** Zero local storage usage completely eliminates SD card corruption.
- **Rapid Re-image:** Provisioning takes ∼2 min via NFS vs ∼20 min per volatile flash card.
- **Shared Storage:** Unified network filesystem hosts all cluster binaries and hostfiles.

Drawbacks:

- **Master SPOF:** Central Pi 5 failure immediately knocks down all dependent worker nodes.
- **Network Latency:** All disk I/O traverses the LAN, reducing heavy database performance.
- **RAM Constraints:** Local `/tmp` is bound to a ∼450MB `tmpfs`, stalling massive renders.
- **Setup Complexity:** Intricate alignment of TFTP, NFS, and `dnsmasq` rules fails silently.

Frontend & Backend

Frontend: The web interface is developed using **React**, providing a responsive single-page application (SPA) with **React Router** for seamless navigation. The application is served by **Nginx** and deployed as two **k3s** replicas for high availability.

How the Frontend Talks to the Backend: Every page in EdgeGuard needs data from the backend. The pattern is always the same: React component mounts → calls backend API with JWT token → receives JSON → renders data. Live pages (Camera, Monitoring) poll the API every few seconds to keep data fresh.

Backend: An **event-driven architecture** connects the edge devices with the web application. Detection results generated on the Raspberry Pi cluster are transmitted via **ZeroMQ**, processed by a **Flask REST API**, stored in a **PostgreSQL** database, and immediately forwarded as **Telegram notifications** for rapid incident response.

Workflow: The complete processing pipeline follows the sequence: **AI Camera → Pi 4 Sensor → ZeroMQ → Pi 5 (Flask API) → PostgreSQL → Web Dashboard & Telegram Alerts**.

Auto-Scaling

Auto-Scaling: A custom auto-scaler monitors cluster CPU usage with **Prometheus** and automatically adds or removes the Raspberry Pi 4 node based on workload.

Scale-Out: High CPU → Start k3s-Agent → Pi 4 Joins → Telegram Alert

Scale-In: Low CPU → Cordon & Drain → Remove Node → Telegram Alert

References

Scan to access the EdgeGuard project website.