

Performance Analysis and Optimization of Hybrid Edge-Cloud Architectures for Real-Time Robotics Applications

by

Maddie Salehpoor

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2025
© Maddie Salehpoor 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis explores the optimization of distributed robotic perception systems for real-time applications such as autonomous navigation, smart surveillance, and multi-agent coordination. These systems require fast processing of high-frequency sensor data under tight latency and reliability constraints.

Edge computing offers low-latency inference and privacy, while cloud computing provides centralized scalability and resource efficiency, but suffers from transmission delays and network dependency. Critically, the bandwidth required to transmit raw video streams and LiDAR pointcloud messages to the cloud is often infeasible, necessitating edge-side preprocessing.

To address these challenges, this work proposes and evaluates a hybrid edge–cloud architecture in which each sensor node that is equipped with its own LiDAR and camera performs background removal and motion detection locally at the edge, while the remaining perception tasks are offloaded to the cloud. This design reduces bandwidth usage and enables real-time responsiveness under constrained conditions. While edge modules cannot perform full object classification, they provide fast "reflex-like" responses that enable event filtering, alert triggering, and resource prioritization, reserving the more computationally intensive object recognition for cloud processing.

LiDAR processing was parallelized using Intel TBB and spatial chunking, achieving up to $2\times$ speedups across 1–32 cores. For camera-based perception, Frame Differencing, GMM, and Dense Optical Flow were tested. Frame Differencing proved most effective for edge deployment, achieving 100% message reliability and 80.9% bandwidth savings with a 20.8 ms average processing time.

Scalability tests showed that a 64-core system can support up to 75 nodes at 10 Hz or 50 nodes at 20 Hz with no message loss while meeting real-time constraints. Communication protocol testing revealed latencies ranging from 0.24 ms (ROS) to 2,589 ms (ZeroMQ over WiFi), setting architectural limits for cloud offloading. Finally, cost analysis showed that a 64-core cloud instance could replace 50 edge devices (\$44,950 upfront), offering cost-effective scalability in suitable deployments.

This work delivers: (1) empirical benchmarks that reveal how LiDAR and camera perception scale under parallel processing, (2) motion-based filtering techniques that significantly reduce bandwidth without sacrificing accuracy, (3) real-world measurements of communication protocol latency under 5G and Wi-Fi, and (4) practical deployment guidelines for hybrid edge–cloud robotic systems.

Acknowledgements

Sincere gratitude is extended to my advisors, Professor Amir Khajepour and Professor Samer Al-Kiswany, for their invaluable guidance, unwavering support, and insightful feedback throughout this research journey. Their mentorship played a critical role in shaping the direction and quality of this thesis.

I would also like to thank Minghao Ning for his kind support during the system setup and for his helpful feedback throughout the development process. His technical insights and willingness to assist were instrumental to the success of this work.

Finally, I am grateful to all members of the Mechatronic Vehicle Systems (MVS) Lab for fostering a collaborative and motivating research environment that greatly enriched my academic experience.

Dedication

This is dedicated to my parents.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	xi
List of Tables	xiv
1 Introduction	1
1.1 Overview and Motivation	1
1.2 Bandwidth and Scalability Constraints in Distributed Robotics	2
1.3 Research Context and Positioning	3
1.4 Research Objectives and Key Contributions	3
1.5 System Requirements	4
1.5.1 Functional Requirements	4
1.5.2 Performance Requirements	5
1.5.3 Scalability and Economic Requirements	6
1.5.4 Reliability and Network Requirements	6

1.5.5	Network and Environmental Assumptions	6
1.6	Research Scope and Limitations	7
1.7	Thesis Organization and Methodology Overview	7
2	Literature Review	9
2.1	Introduction	9
2.2	Edge-Cloud Computing Paradigms	10
2.3	5G Networks as Enablers of Hybrid Architectures	11
2.4	Intelligent Data Filtering and Bandwidth Optimization	13
2.4.1	Motion-Based Filtering for Bandwidth Reduction	14
2.5	Communication Protocols for Distributed Robotics	15
2.5.1	Protocol Selection for High-Frequency Data Transmission	16
2.5.2	Protocol Performance Under Variable Network Conditions	16
2.6	Containerization and Parallel Processing in Robotics	17
2.6.1	Containerized Deployment for Robotics Applications	17
2.6.2	Parallel Processing and Scalability Analysis	18
2.7	Research Gaps and Thesis Positioning	19
3	Experimental Setup and Framework	20
3.1	Introduction	20
3.2	Foundational Project Context: MVS Lab 5G Intelligent Transportation System	21
3.2.1	Overview of the Existing System	21
3.2.2	Existing Architecture and Research Motivation	21
3.3	Experimental System Architecture Building on MVS Lab Infrastructure	24
3.3.1	Foundation: MVS Lab Architecture	24
3.3.2	Experimental Hybrid Architecture	25
3.3.3	Justification for Edge-Level Motion Detection Without Object Identity	26

3.3.4	Containerized Deployment and ROS Architecture	27
3.4	Sensor Processing Algorithms	28
3.4.1	LiDAR Processing Pipeline	28
3.4.2	Camera Processing Algorithms	31
3.4.3	Communication Protocol Implementation	35
3.5	Network Condition Simulation	37
3.5.1	Traffic Control (tc) Configuration	37
3.5.2	Network Performance Validation	38
3.6	Performance Evaluation Framework	39
3.7	Integration with MVS Lab Dataset and Infrastructure	41
3.7.1	Dataset Utilization from MVS Lab Deployment	41
3.7.2	Trade-Offs and Value Assessment Against Edge-Centric Baselines	42
3.7.3	Experimental Validation	42
3.8	Summary	43
4	From Sequential to Parallel: Performance Evaluation of Transformed Algorithms	44
4.1	Introduction	44
4.2	Parallel Processing Performance Evaluation	45
4.2.1	Performance Metrics and Mathematical Formulations	45
4.2.2	Experimental Setup and Parallel Processing Framework	47
4.2.3	Processing Time Performance Analysis	48
4.2.4	Component-Level Profiling and Performance Summary	49
4.2.5	Amdahl's Law Theoretical Validation	51
4.2.6	Throughput Performance Analysis	54
4.2.7	Optimization Insights and Recommendations	57

5	Multi-Node Scalability Analysis	58
5.1	Scalability Analysis	59
5.1.1	Experimental Setup	60
5.1.2	Scalability Metrics and Trends	60
5.1.3	Resource Utilization Trends	66
5.1.4	CPU Efficiency Across Configurations	67
5.1.5	Scalability of CPU and Memory Utilization Across Systems	68
6	Selective Camera Data Transmission via Edge-Based Motion Detection	71
6.1	Camera Algorithm Performance Comparison	72
6.1.1	Motion Detection Algorithm Overview	72
6.1.2	Evaluation Metrics	75
6.1.3	Camera Dataset Overview	76
6.1.4	Algorithm Performance Results	76
6.1.5	Comprehensive Statistical Analysis	77
6.1.6	Conclusion	82
7	Edge vs Cloud Deployment: Performance, Communication, and Cost Trade-offs	84
7.1	Communication Latency Analysis: ROS vs ZeroMQ	84
7.2	Cost Analysis of Edge and Cloud Deployment	87
7.3	Synthesis: When to Use Edge vs Cloud	87
8	Conclusions and Future Work	89
	References	92
	APPENDICES	99

A	ROS Performance Scalability	100
A.1	ROS Performance Analysis Results	100
A.1.1	Test Configuration	100
A.1.2	Throughput Performance 20Hz	101
A.1.3	Processing Time Analysis 20Hz	101
A.1.4	Speedup Analysis	102
A.1.5	Resource Efficiency 20Hz	103
A.1.6	Performance Summary 20Hz	103
A.1.7	Test Execution Framework	103

List of Figures

3.1	MVS Lab 5G Intelligent Transportation System: 14 ISNs deployed across indoor and outdoor environments [40].	22
3.2	Multi-sensor data fusion demonstration showing complementary information from multiple LiDAR sensors [40].	23
3.3	MVS Lab Distributed System Architecture [40].	24
3.4	LiDAR processing pipeline: raw point clouds undergo background subtraction to remove static features, followed by clustering to group dynamic objects.	30
3.5	Chunk-based parallelization strategy for LiDAR processing.	31
3.6	Frame differencing processing pipeline	32
3.7	Gaussian Mixture Model (GMM) processing pipeline	33
3.8	Dense optical flow processing pipeline	34
3.9	Hybrid Communication Protocol Architecture: ROS topics enable local edge communication; ZeroMQ handles hybrid edge-cloud messaging; MQTT provides reliable cloud integration for final result transmission.	35
4.1	Processing time distributions for (left) background subtraction and (right) LiDAR clustering.	48
4.2	Stacked processing time by component at varying core counts, highlighting the increasing prominence of sequential overheads.	51
4.3	Amdahl's Law curve fitting for background subtraction (left) and LiDAR clustering (right), illustrating theoretical speedup limits.	52
4.4	Speedup and efficiency at 10 Hz playback rate. Speedup shows modest gains while efficiency declines due to I/O limitations.	55

5.1	Scalability metrics at 10 Hz input rate: (Top) Throughput (Messages/Second); (Middle) Average processing time per node (ms); (Bottom) Message loss (%).	62
5.2	Scalability metrics at 20 Hz input rate: (Top) Per-node processing time (ms); (Middle) System throughput (Messages/Second); (Bottom) Message loss rate (%).	64
5.3	CPU efficiency across varying core counts and node densities. Efficiency improves with more nodes until coordination overhead and saturation effects set in.	68
5.4	Per-node CPU usage across scaling scenarios. Lower-core systems degrade rapidly, while high-core systems sustain consistent per-node processing costs.	69
5.5	Trade-off between CPU efficiency and memory usage per node. Docker's per-container memory footprint (150–250 MiB) adds baseline overhead to all configurations.	69
6.1	Frame differencing three-panel visualization showing: (left) bounding boxes on input frame, (center) thresholded difference mask, (right) compressed motion regions.	73
6.2	GMM three-panel visualization showing: (left) bounding boxes on input frame, (center) motion mask from background subtraction, (right) compressed regions transmitted to cloud.	73
6.3	Dense optical flow three-panel visualization showing: (left) input frame with bounding boxes, (center) flow magnitude mask, (right) transmitted regions with adaptive padding.	74
6.4	Processing time distributions for each motion detection algorithm. Left: detailed comparison of Frame Differencing and GMM. Right: Optical Flow showing significantly higher latency but low variance.	78
6.5	CPU usage per algorithm. All remain below 5% on average.	80
6.6	Throughput (FPS) distribution for the selected motion detection algorithm.	81
6.7	Bandwidth savings across algorithms. All maintain high compression efficiency.	83
7.1	Latency distribution across protocols and networks. ROS internal communication offers stable low-latency messaging; ZeroMQ performance varies drastically with network quality.	86

7.2	Architecture selection decision tree for robotics deployment. The tree guides developers through latency constraints, scale, network quality, and processing complexity to determine whether an edge-only, cloud-based, or hybrid system is most appropriate.	88
-----	---	----

List of Tables

3.1	Low-to-Medium Load Docker Configurations (20 Nodes)	29
3.2	High-Load Docker Configurations (>20 Nodes)	30
3.3	Network Configuration Scenarios	37
3.4	Network Performance Validation Results	39
3.5	Performance Metrics for Distributed ROS System Evaluation	40
4.1	ROS Bag Dataset Specifications - Background and Filtered LiDAR	47
4.2	Background Subtraction Processing Time Statistics (Standard Speed)	49
4.3	LiDAR Clustering Processing Time Statistics (Standard Speed)	49
4.4	Background Subtraction Component Breakdown (Standard Speed)	50
4.5	LiDAR Clustering Component Breakdown (Standard Speed)	50
4.6	Amdahl's Law Fitting Results: Parallel and Serial Fractions	52
4.7	Theoretical Predictions vs. Experimental Results	53
4.8	Optimization Impact Scenarios	54
4.9	Throughput at Standard Replay (10 Hz) Across Core Counts	55
4.10	Throughput at Accelerated Replay Rates (20–70 Hz)	55
4.11	Processing Speedup Compared to Throughput Gain Across Core Counts	56
5.1	Optimal Configuration Recommendations at 10Hz (Processing Time $\leq 18.78\text{ms}$, Message Loss = 0%)	63
5.2	Optimal Configuration Recommendations at 20Hz (Processing Time $\leq 18.78\text{ms}$)	65

5.3	Total Resource Utilization Summary (Memory in GiB)	67
6.1	Motion Detection Algorithm Comparison	74
6.2	Summary of Image Frames in ROS Bags (per camera) with Frames Per Second	76
6.3	Message Processing Performance Summary	77
6.4	Processing Time Statistical Analysis (milliseconds)	78
6.5	CPU Usage Statistical Analysis (percentage)	79
6.6	Measured Throughput (FPS) Statistics	80
6.7	High Throughput Stress Test Summary	82
6.8	Bandwidth Savings Statistical Analysis (percentage)	82
7.1	ROS Internal Communication Performance (4-Core, Single Node)	85
7.2	ZeroMQ Edge–Cloud Communication Performance	85
7.3	Total Cost Comparison: Edge-Only vs Cloud	87
A.1	Throughput Performance (Messages/Second) 20Hz	101
A.2	Processing Time (Milliseconds) 20Hz	101
A.3	Processing Time Speedup (Relative to 1 Core, 1 Node) 20Hz	102
A.4	Throughput Speedup (Relative to 1 Core, 1 Node) 20Hz	102
A.5	Efficiency (Messages/Second per Core) 20Hz	103

Chapter 1

Introduction

1.1 Overview and Motivation

The rapid deployment of autonomous systems in domains such as self-driving vehicles, industrial automation, and smart cities has created a growing demand for efficient real-time sensor processing. Modern robotic platforms integrate multiple high-bandwidth sensors, including LiDAR systems producing up to 2.2 million points per second [6, 58] and stereo cameras generating over 200 Mbps of raw imagery [6, 23]. While this fusion of data enhances situational awareness, it also presents critical challenges in computation and communication. These challenges affect responsiveness, reliability, and the overall operational cost of autonomous systems.

Two dominant approaches exist to address these processing needs. The first is an edge-only strategy, where all sensor data is processed locally. This minimizes latency but is constrained by the limited computational resources of edge devices. The second is a cloud-centric approach, which offers virtually unlimited processing capacity but introduces variable latency and depends heavily on network availability. In practice, neither approach is sufficient on its own. Real-world systems require an architecture that can dynamically balance these trade-offs based on environmental conditions, network quality, and task requirements.

This thesis proposes and evaluates a hybrid edge-cloud architecture tailored for multi-sensor robotic systems. Building on recent advances in containerized processing and distributed system design, the architecture employs intelligent filtering at the edge to minimize data transfer, while offloading complex perception tasks to cloud resources. The design

philosophy is based on the observation that not all sensor data requires equal processing: static background information can be handled locally, while dynamic scenes benefit from centralized, high-performance analysis.

1.2 Bandwidth and Scalability Constraints in Distributed Robotics

A central challenge in distributed robotic systems lies in the imbalance between the volume of sensor data produced and the system’s capacity to transmit and process it effectively. Autonomous platforms typically generate enormous data volumes; for example, a single vehicle can produce over 4 TB of sensor data daily [6]. Yet, existing communication networks—including 5G and Wi-Fi—struggle to maintain the consistent low-latency transmission rates required for real-time operation. Transmitting full-resolution sensor streams in real time is therefore not only impractical but also cost-prohibitive in bandwidth-constrained environments.

This research addresses this bottleneck by implementing intelligent edge-level preprocessing that reduces data transmission volume by 80–90%. Lightweight techniques such as motion detection for camera streams and background subtraction for LiDAR point clouds allow edge devices to filter out static or redundant data before forwarding dynamic scene changes to the cloud. This targeted filtering enables the use of cloud resources even when bandwidth is limited, while still preserving the information necessary for high-level perception tasks.

Beyond bandwidth, scalability presents another layer of complexity. Real-world deployments often involve dozens or even hundreds of sensor-rich robotic platforms operating in parallel. Processing this volume of sensor data centrally or uniformly across all devices requires considerable computational resources. However, simply adding more hardware increases both operational costs and system complexity.

To address this, the system architecture leverages containerized processing pipelines that can be deployed flexibly across devices ranging from single-core edge nodes to 64-core cloud servers. Extensive benchmarking reveals how performance scales with the number of cores and devices, and identifies practical configurations that optimize throughput while controlling cost. These findings inform a scalable deployment strategy that balances performance and resource utilization as system size grows.

Together, these solutions lay the groundwork for an architecture that is not only responsive and bandwidth-aware but also economically viable at scale.

1.3 Research Context and Positioning

This work builds upon prior research at the Mechatronic Vehicle Systems (MVS) Lab at the University of Waterloo, which has developed edge-based perception pipelines for autonomous fleet systems. In their 5G Cloud-Based Fleet Mobility platform [36], the MVS Lab demonstrated the feasibility of performing full LiDAR and camera data processing on NVIDIA Jetson Orin NX devices. These systems processed sensor streams entirely onboard, using embedded GPUs to execute object detection, sensor fusion, and scene understanding in real time. Processed results were then transmitted to the cloud for mission-level coordination.

While effective, this edge-centric approach revealed several limitations. The use of GPU-enabled edge devices led to significant hardware costs (approximately \$900 per node), raising concerns about scalability. Algorithm complexity was limited by thermal and power constraints, and maintenance became increasingly complex as fleet size grew. Additionally, edge resources could not be shared, resulting in imbalanced workloads and underutilization.

To address these constraints, this thesis introduces a hybrid processing model. Lightweight preprocessing (motion detection and background subtraction) is performed at the edge while compute-intensive tasks such as clustering and recognition are delegated to the cloud. This division reduces per-node hardware requirements, allows for more sophisticated algorithms, and simplifies software updates by centralizing them in the cloud.

Of course, introducing cloud components raises new concerns related to network reliability, latency, and bandwidth. To address this, the system includes edge-level filtering to reduce unnecessary data transmission and is evaluated under various network conditions—including Ethernet, WiFi, 5G, and degraded links. The goal is to show that with appropriate partitioning, a hybrid architecture can outperform edge-only systems in scalability, maintainability, and performance.

1.4 Research Objectives and Key Contributions

This thesis investigates the design, implementation, and evaluation of a hybrid edge-cloud architecture through four core research objectives:

- 1. Parallel Processing Optimization:** This objective examines how LiDAR perception algorithms can be parallelized using Intel Threading Building Blocks (TBB). Single-threaded implementations of background subtraction and Density-Based Spatial Clustering

of Applications with Noise (DBSCAN) clustering are refactored to exploit multicore platforms. Experiments on systems ranging from 1 to 64 CPU cores show that up to 55% of the code is parallelizable, yielding a speedup consistent with Amdahl’s Law, with a theoretical limit of approximately 2.2 times.

2. Communication Efficiency and Bandwidth Reduction: The system evaluates ROS, MQTT, and ZeroMQ protocols across various network conditions. It also introduces motion detection-based filtering for camera data and background subtraction for LiDAR to reduce bandwidth consumption. The system achieves up to 90% bandwidth reduction while maintaining critical scene content, with latency measurements confirming responsiveness across realistic wireless and degraded networks.

3. Containerized System Architecture: A fully containerized deployment framework is developed to ensure reproducibility and portability across different hardware configurations. Experimental results demonstrate that containerization introduces negligible overhead (less than 3%) and supports seamless scaling from development to deployment.

4. Economic and Deployment Analysis: A comparative cost analysis shows that hybrid processing can reduce operational expenses by 25% over edge-only architectures. For a 50-node system, this results in annual savings of approximately \$11,000. A decision framework is proposed to help designers choose between edge, cloud, or hybrid strategies based on scale, cost, and performance requirements.

Together, these contributions provide both theoretical grounding and practical insights for building efficient, scalable, and maintainable distributed robotic systems.

1.5 System Requirements

To achieve the research objectives outlined above, this thesis establishes a comprehensive set of functional and non-functional requirements for the hybrid edge-cloud architecture. These requirements guide both the system design and evaluation methodology.

1.5.1 Functional Requirements

- **FR1 - Multi-Sensor Data Processing:** The system must support real-time processing of heterogeneous sensor streams, including LiDAR point clouds (up to 2.2 million points per second) and stereo camera imagery (over 200 Mbps raw data rates).

- **FR2 - Hybrid Processing Architecture:** The system must implement intelligent workload partitioning, with lightweight preprocessing (motion detection, background subtraction) performed at the edge and compute-intensive tasks (clustering, recognition) delegated to cloud resources.
- **FR3 - Dynamic Load Balancing:** The architecture must support dynamic allocation of processing tasks between edge and cloud components based on current network conditions, computational load, and task requirements.
- **FR4 - Communication Protocol Support:** The system must support multiple communication protocols (ROS for local messaging, MQTT for coordination, ZeroMQ for edge-cloud communication) to ensure interoperability across different network environments.
- **FR5 - Containerized Deployment:** All system components must be deployable using containerization technology to ensure reproducibility, portability, and simplified scaling across heterogeneous hardware configurations.

1.5.2 Performance Requirements

- **PR1 - Latency Bounds:** Edge processing must maintain sub-millisecond latency for critical safety functions, while cloud processing should achieve acceptable latency (<150ms) under normal network conditions.
- **PR2 - Bandwidth Efficiency:** The system must achieve at least 80% bandwidth reduction through intelligent filtering while preserving critical scene information necessary for perception tasks.
- **PR3 - Parallel Processing Scalability:** The architecture must effectively utilize multicore platforms (1-64 cores) with performance scaling consistent with theoretical models such as Amdahl's Law.
- **PR4 - Throughput Requirements:** The system must support processing of high-bandwidth sensor streams while maintaining real-time operation across varying computational loads.

1.5.3 Scalability and Economic Requirements

- **SR1 - Multi-Node Scalability:** The architecture must support deployment across 10-100+ sensor nodes while maintaining system coherence and performance.
- **SR2 - Resource Efficiency:** The system must demonstrate measurable improvements in resource utilization compared to edge-only or cloud-only approaches.
- **ER1 - Cost Optimization:** The hybrid architecture must achieve at least 20% operational cost reduction compared to edge-only deployments for systems with 10+ nodes.
- **ER2 - Hardware Cost Reduction:** By offloading compute-intensive tasks to the cloud, the system must reduce per-node hardware requirements and associated costs.

1.5.4 Reliability and Network Requirements

- **RR1 - Network Resilience:** The system must maintain acceptable performance across various network conditions including Ethernet, WiFi, 5G, and degraded connections.
- **RR2 - Graceful Degradation:** When cloud connectivity is unavailable, edge components must continue operating with reduced functionality rather than complete system failure.
- **RR3 - Data Integrity:** All sensor data transmission must maintain integrity and consistency across the edge-cloud boundary.

These requirements establish the foundation for system design decisions and provide measurable criteria for evaluating the success of the proposed hybrid architecture.

1.5.5 Network and Environmental Assumptions

- **NA1 - 5G Network Characteristics:** All 5G performance requirements and evaluations are based on simulated network conditions with approximately 100ms round-trip latency, representing realistic urban deployment scenarios with network congestion and infrastructure limitations, rather than theoretical peak 5G performance.

The simulated environment can be easily adjusted to emulate other network configurations (including ideal 5G, LTE, or degraded conditions), but this thesis focuses on the 100ms latency scenario as representative of practical deployment conditions.

- **NA2 - Network Variability:** The system must account for network condition variations including packet loss, jitter, and bandwidth fluctuations typical of real-world wireless deployments.

1.6 Research Scope and Limitations

The focus of this research is on perception algorithms commonly used in autonomous robotics. These include background subtraction and DBSCAN-based clustering for LiDAR, and motion detection using frame differencing, Gaussian Mixture Models (GMM), and optical flow for camera data. These methods were selected for their widespread use and simplicity, offering a practical basis for evaluation. However, results may not generalize to deep learning-based perception systems or specialized signal processing algorithms.

All evaluations are performed on x86-64 CPU platforms ranging from 1 to 64 cores. GPU and FPGA acceleration are intentionally excluded to reflect the limitations of resource-constrained edge devices. This decision also establishes a conservative performance baseline, which future work can extend using more powerful hardware.

Three communication protocols are studied: ROS for local messaging, MQTT for lightweight coordination, and ZeroMQ for high-throughput edge-cloud communication. Network tests are performed using Linux Traffic Control (tc) to simulate ideal Ethernet, WiFi, 5G, and degraded connections. While this approach allows for precise control and repeatability, it does not fully capture real-world variability such as interference, congestion, or mobile signal degradation.

Despite these constraints, the system architecture and experimental methodology are designed to be extensible. The containerized processing pipeline and parallel evaluation framework can be easily adapted to new algorithms, protocols, or hardware configurations, making this work a strong foundation for future research.

1.7 Thesis Organization and Methodology Overview

The structure of this thesis follows a logical progression from foundational theory to practical system implementation and evaluation:

- **Chapters 2–3:** Present the theoretical background, related work, and describe the experimental framework. This includes the design of the containerized architecture and the edge-cloud processing pipeline.
- **Chapters 4–5:** Detail the parallel processing implementation and scalability experiments. These chapters provide empirical validation of performance against theoretical models such as Amdahl’s Law.
- **Chapter 6:** Focuses on camera-based perception algorithms, including motion detection techniques and their evaluation for edge deployment. This chapter analyzes algorithmic performance, bandwidth savings, and suitability for real-time robotics applications.
- **Chapter 7:** Examines communication protocols and network conditions, presenting a comprehensive cost-performance analysis for edge versus cloud deployment strategies. It discusses protocol latency, network effects, and economic trade-offs impacting architecture decisions.
- **Chapter 8:** Summarizes the key findings, discusses broader deployment implications, and proposes future research directions based on the empirical results.

Chapter 2

Literature Review

2.1 Introduction

The evolution of distributed computing architectures has fundamentally transformed how robotic systems process sensor data and make real-time decisions. Traditional approaches have relied on either purely edge-based processing, constrained by limited local computational resources, or cloud-centric architectures that offer abundant processing power at the cost of network-dependent latencies. This chapter examines the emergence of hybrid edge-cloud computing paradigms that intelligently distribute computational tasks between edge devices and cloud infrastructure to optimize performance, bandwidth utilization, and system scalability.

The literature reveals a growing consensus that effective robotics perception systems require sophisticated task partitioning strategies that can dynamically allocate processing based on computational complexity, network conditions, and real-time requirements. The advent of advanced wireless technologies, particularly 5G networks, has provided the communication infrastructure necessary to make such hybrid architectures practical for real-world deployments. However, significant research gaps remain in understanding how to effectively implement intelligent task distribution, optimize bandwidth through selective data transmission, and manage the complex trade-offs inherent in distributed processing systems.

2.2 Edge-Cloud Computing Paradigms

The evolution of edge-cloud computing has been driven by a fundamental need: delivering low-latency, high-performance processing for applications where real-time responsiveness is essential. While cloud computing offers virtually unlimited resources and global scalability, its reliance on centralized infrastructure introduces latency that can undermine time-sensitive tasks. In contrast, edge computing processes data closer to the source (on or near the device) offering faster response times and greater control over local resources. This trade-off between latency and computational capacity forms the foundation of hybrid edge-cloud architectures, which aim to dynamically allocate tasks across the edge and the cloud based on system context.

Shi et al. [50] established one of the early theoretical foundations for edge computing, demonstrating that offloading latency-sensitive tasks to local nodes can reduce response times by 50–80% compared to traditional cloud-based systems. Their work laid the groundwork for hybrid models that intelligently partition workloads between local and remote resources. Building on this idea, Chen et al. [14] developed adaptive task placement strategies for autonomous vehicles that achieved 60% latency reduction without sacrificing reliability, highlighting the importance of real-time task reallocation based on environmental and network conditions.

More recent work by Wang et al. [60] reinforces this dynamic viewpoint, showing that hybrid task scheduling, when informed by data locality, network availability, and energy constraints can achieve 2–3 $\times$ performance improvements over static edge-only or cloud-only deployments. These studies converge on a key insight: effective edge-cloud architectures must go beyond static partitioning and instead implement flexible, context-aware scheduling mechanisms that adapt in real time.

For mobile robotics applications, where systems must operate under varying connectivity and power conditions, mobile edge computing (MEC) introduces additional considerations. Abbas et al. [2] emphasize the privacy and security advantages of MEC, especially in environments where transmitting sensitive data to the cloud may be undesirable or unreliable. Their findings support the use of edge nodes not just for latency reduction but also as a privacy-preserving buffer that can isolate mission-critical tasks from cloud dependencies.

Dong et al. [16] extend this by analyzing task offloading strategies across multiple edge scenarios, identifying latency reduction, energy conservation, and application-specific constraints as key decision drivers. Their taxonomy of offloading objectives offers a practical framework for edge-cloud partitioning that aligns well with robotics, where workloads vary dynamically and system constraints are constantly shifting.

Rahman et al. [46] further contrast the strengths of edge and cloud computing in the context of sensor-rich environments such as the Internet of Things (IoT). They argue that while edge nodes offer low-latency and privacy-preserving capabilities, cloud infrastructure remains indispensable for resource-heavy operations such as large-scale object recognition or cross-platform data fusion. This reinforces the need for hybrid architectures capable of leveraging both ends of the computational spectrum.

A central challenge in hybrid system design is deciding where and when to offload specific tasks. Zhao et al. [65] highlight that this decision must consider not only CPU load or bandwidth, but also energy budgets and real-time constraints, particularly in battery-powered systems like drones or autonomous robots. Similarly, Kumar et al. [25] identify bandwidth bottlenecks as a critical obstacle, noting that intelligent data filtering and compression are essential to reduce communication overhead without degrading task performance.

Despite the growing body of research, most of these studies have focused on generic edge-cloud applications or simulations rather than real-world robotics workloads. Robotic systems introduce unique challenges: high-frequency, high-bandwidth sensor data (e.g., LiDAR, cameras), safety-critical timing constraints, and highly dynamic environments. Addressing these demands requires specialized workload partitioning strategies that go beyond what has been validated in conventional cloud environments. This thesis builds on these insights by empirically evaluating a hybrid architecture tailored to real-time robotics, with a focus on performance scaling, task distribution, and communication efficiency across varied edge and cloud configurations.

2.3 5G Networks as Enablers of Hybrid Architectures

The rapid advancement and deployment of fifth-generation (5G) wireless networks have fundamentally reshaped the landscape of distributed robotics, offering the bandwidth, latency, and reliability characteristics necessary to enable seamless edge-cloud integration. Unlike previous generations of wireless technology, 5G was designed from the ground up to support real-time, data-intensive, and safety-critical applications, characteristics that align directly with the demands of autonomous robotic systems. With service categories such as enhanced mobile broadband (eMBB), ultra-reliable low-latency communication (URLLC), and massive machine-type communication (mMTC), 5G has become the de facto communication backbone for emerging hybrid architectures.

These capabilities are not merely theoretical. Ma et al. [33] demonstrate that 5G-enabled communication allows real-time sharing of sensor data and decision outputs be-

tween distributed nodes, supporting a level of collaborative perception that exceeds the limitations of individual edge devices. Their analysis confirms that 5G offers up to 20 Gbps in bandwidth, sub-millisecond latency, and five-nines reliability, precisely the characteristics needed for the kind of responsive, coordinated behavior that hybrid robotic systems require.

Expanding on this foundation, Hakak et al. [21] provide a comprehensive examination of how each of the three primary 5G service categories maps to specific needs in autonomous vehicles and robotics. In particular, they emphasize that URLLC is indispensable for closed-loop control systems where delays of even a few milliseconds can compromise safety. Meanwhile, eMBB supports the massive throughput demands of transmitting high-resolution sensor data to the cloud for centralized processing. Their work reinforces the idea that 5G is not merely an enabler it is a prerequisite for robust hybrid edge-cloud robotics.

Beyond raw communication capabilities, 5G also unlocks new paradigms in cooperative processing. In scenarios where multiple robotic agents operate within the same environment, 5G facilitates the sharing of both sensor data and computational results, enabling a form of collective intelligence. Palacios-Morocho et al. [41] show that with 5G connectivity, vehicles with limited onboard compute can offload intensive tasks to nearby edge servers or cloud infrastructure, maintaining real-time performance while reducing onboard hardware complexity. This ability to distribute perception and decision-making tasks dynamically across a network of agents introduces new possibilities for system-wide optimization.

Collaborative perception becomes even more powerful when paired with precise localization and context sharing. Bartoletti et al. [8] illustrate how 5G networks support centimeter-level positioning accuracy are crucial for tasks like autonomous navigation and swarm coordination. This level of precision, combined with real-time data sharing, allows edge devices to compensate for individual limitations by fusing inputs from multiple sources, creating a more complete and reliable situational awareness.

However, effective cooperation at scale demands more than just connectivity, it requires intelligent bandwidth management. Thandavarayan et al. [57] caution against naive data sharing strategies, showing that indiscriminate transmission of raw data can rapidly overwhelm available bandwidth, even with 5G. Their study proposes a selective filtering approach that prioritizes novel or dynamic information, reducing bandwidth consumption by up to 80% while maintaining the quality of shared perception. This principle aligns directly with the intelligent filtering strategies employed in this thesis to optimize network efficiency.

Yet, despite its technical promise, the real-world deployment of 5G introduces a new

set of challenges. As Storck and Duarte-Figueiredo [53] note, achieving consistent 5G performance in urban and industrial environments requires dense infrastructure deployment and careful tuning of network parameters. Signal occlusion, multipath interference, and fluctuating user loads can degrade network performance, making it essential for edge-cloud systems to operate reliably under non-ideal conditions.

Security concerns further complicate the picture. Soori et al. [52] highlight the vulnerabilities introduced by increased cloud dependence and remote connectivity, emphasizing that hybrid systems must be designed to withstand network disruptions, data breaches, and remote attacks. Their findings support the argument that safety-critical functions such as motion detection or obstacle avoidance should remain at the edge whenever possible to ensure uninterrupted operation.

Finally, Li et al. [30] address the gap between theoretical 5G specifications and actual performance in the field. They advocate for adaptive system architectures that monitor and respond to real-time network conditions rather than relying on ideal assumptions. Their insights underscore the importance of dynamic workload partitioning, a key strategy employed in this thesis to maintain consistent performance despite unpredictable connectivity.

In sum, 5G networks offer the technical foundation for next-generation robotics systems that combine the speed of the edge with the power of the cloud. However, realizing this vision in practice requires more than connectivity alone; it demands intelligent data management, cooperative computation strategies, and system architectures that can adapt to real-world constraints. This thesis builds on these principles, incorporating empirical evaluations of bandwidth-aware processing, protocol selection, and dynamic edge-cloud partitioning in real-time robotics environments.

2.4 Intelligent Data Filtering and Bandwidth Optimization

While 5G networks provide the foundational connectivity for hybrid edge-cloud systems, bandwidth remains a critical bottleneck especially in real-world deployments where network conditions can be inconsistent or degraded. Even with high-throughput links, continuously transmitting raw sensor data from multiple high-resolution sources is rarely practical or efficient. To make hybrid architectures viable, systems must intelligently filter and compress sensor data, reducing transmission loads without compromising perception quality.

This section explores motion-based filtering and temporal LiDAR compression techniques that enable such bandwidth-aware optimization.

2.4.1 Motion-Based Filtering for Bandwidth Reduction

In camera-based systems, intelligent data filtering hinges on identifying which parts of a scene carry actionable information. Rather than transmitting every video frame in its entirety, modern systems apply motion detection and background subtraction algorithms to isolate dynamic elements objects (in motion or regions of change) while discarding static, redundant content. This allows for substantial data reduction while preserving critical visual cues for perception and decision-making.

A widely adopted method for this task is Zivkovic’s adaptive Gaussian Mixture Model (GMM) [66], which dynamically models background variations and robustly detects moving objects in complex environments. Originally applied to surveillance systems, GMM techniques have demonstrated bandwidth reductions of up to 90% by transmitting only foreground regions. However, when applied to robotics, additional challenges arise. Mobile cameras introduce frequent changes in viewpoint and lighting, requiring careful parameter tuning and adaptive strategies to maintain detection accuracy.

Complementary to GMM-based approaches are simpler methods like frame differencing, which subtract sequential frames to identify regions of motion. Brutzer et al. [12] compare a range of background subtraction techniques and highlight the trade-off between computational cost and detection fidelity. Their findings suggest that while advanced algorithms offer better segmentation accuracy, lightweight techniques are often more suitable for edge deployment where real-time performance and energy efficiency are paramount. This trade-off directly informs the multi-algorithm strategy adopted in this thesis, which tailors filtering methods to available edge resources and system requirements.

Compression Techniques for Sensor Data Transmission

Effective data compression is fundamental to enabling real-time edge-cloud communication in bandwidth-constrained environments. Traditional compression approaches focus on spatial redundancy, but robotics applications present unique opportunities for temporal compression due to the predictable motion patterns and environmental structure typical in mobile sensing scenarios.

Liu et al. [32] present adaptive LiDAR compression methods that preserve spatial fidelity while reducing bandwidth consumption by more than 80%. Their approach empha-

sizes spatial redundancy through geometric downsampling and progressive mesh encoding, achieving significant compression ratios while maintaining geometric accuracy for navigation tasks. However, their method does not fully leverage the temporal structure inherent in mobile robotics where much of the environment remains static across multiple scans.

For camera data, advanced compression strategies extend beyond traditional video codecs. Zhang et al. [64] explore motion-aware compression that prioritizes dynamic regions, leading to significant transmission reduction without compromising scene understanding. Their selective encoding approach achieves up to 70% bandwidth reduction by applying higher compression ratios to static background regions while preserving full quality for moving objects. This principle of content-aware compression is equally relevant for LiDAR processing.

Hybrid compression schemes that combine lossless encoding for critical data with lossy compression for redundant information have shown particular promise in robotics applications. Real-time compression performance is equally critical as compression efficiency, with hardware-accelerated approaches achieving sub-millisecond encoding times for high-resolution sensor data, making real-time edge processing feasible even for bandwidth-intensive applications.

The integration of temporal filtering techniques into LiDAR processing represents a natural extension of these compression principles. By identifying and suppressing unchanging elements before transmission, systems can unlock further bandwidth savings beyond what spatial compression alone can achieve.

2.5 Communication Protocols for Distributed Robotics

As hybrid edge-cloud robotics systems increasingly rely on high-throughput wireless technologies like 5G, and implement intelligent filtering to minimize data transmission volumes, a crucial question arises: what communication protocol best supports the reliable, low-latency delivery of this filtered sensor data across distributed nodes?

While bandwidth-optimized data streams reduce network strain, they also impose new demands on communication infrastructure especially when transmitting high-frequency updates from multiple sensor modalities. The selected protocol must ensure minimal overhead, efficient resource usage, and resilience under real-world network conditions. This section evaluates candidate protocols for hybrid robotic systems operating over heterogeneous networks such as 5G and WiFi.

2.5.1 Protocol Selection for High-Frequency Data Transmission

Traditional robotics middleware like ROS (Robot Operating System) was primarily designed for local communication within tightly coupled robotic platforms. As systems move toward distributed architectures that span cloud, edge, and multiple robots over wide-area networks, ROS’s assumptions about latency and reliability no longer hold. High-frequency sensor data especially from LiDAR and cameras requires protocols that can handle large volumes of asynchronous, low-latency messages without sacrificing system responsiveness.

MQTT (Message Queuing Telemetry Transport) has emerged as a popular choice in IoT systems due to its lightweight design, topic-based messaging, and support for intermittent connectivity. Tanganelli et al. [56] demonstrate MQTT’s ability to maintain high throughput and reliability even under constrained network conditions, making it attractive for telemetry and control signaling. However, MQTT was not originally designed for high-bandwidth, high-rate sensor streams typical in robotics, and its broker-based architecture may introduce latency or bottlenecks under load.

ZeroMQ offers a brokerless, high-performance alternative that has gained attention in distributed computing and low-latency systems. By eliminating centralized brokers and supporting multiple messaging patterns (e.g., push-pull, pub-sub), ZeroMQ can achieve microsecond-level latencies, as shown by Sustrik and Hurta [55]. These characteristics make it promising for real-time robotics applications, though the trade-offs in reliability, message ordering, and multi-node coordination must be carefully considered particularly in safety-critical systems.

2.5.2 Protocol Performance Under Variable Network Conditions

While protocol design and baseline performance are important, the true test of a communication system lies in how it handles degraded or variable network conditions. In controlled environments, most protocols perform adequately. However, real-world deployments introduce packet loss, jitter, congestion, and coverage fluctuations that can severely impact data integrity and control responsiveness.

5G networks, while offering theoretical support for low-latency, high-bandwidth communication, require careful infrastructure planning to achieve consistent performance. Variations in signal strength, handoffs between cells, and network congestion can all impact latency-sensitive robotics tasks. Therefore, protocols used in 5G-enabled systems must provide mechanisms for buffering, retransmission, or quality-aware degradation.

Conversely, WiFi remains the most commonly available network in indoor robotics deployments but exhibits even greater variability due to environmental interference and congestion. Protocols deployed in WiFi-based systems must accommodate higher latency variance and more frequent connection drops. The ability to gracefully handle such conditions without flooding the network or delaying control signals is crucial for robust robotics communication.

Despite the importance of these challenges, few studies offer systematic evaluations of ROS, MQTT, and ZeroMQ under realistic robotics workloads and network conditions. This thesis addresses this gap by benchmarking each protocol across varying environments, sensor types, and network qualities, offering empirical guidance for selecting the appropriate protocol in real-world hybrid systems.

2.6 Containerization and Parallel Processing in Robotics

2.6.1 Containerized Deployment for Robotics Applications

As communication protocols and bandwidth constraints are addressed, attention must shift to the execution environment responsible for running perception and control pipelines. In hybrid edge-cloud architectures, the use of containerization has become essential for ensuring deployment flexibility, reproducibility, and hardware abstraction. However, while containerization is well-established in general cloud infrastructure, its application to robotics workloads particularly those with real-time and parallel computing requirements, introduces a unique set of challenges.

Unlike web or data-processing services, robotics systems must maintain deterministic timing and interact with hardware interfaces in real time. Zamora et al. [63] demonstrated the feasibility of deploying ROS-based applications in Docker containers, achieving faster setup and deployment with negligible latency overhead. However, their work focused on isolated single-node setups, leaving open questions about multi-container orchestration, inter-process communication, and performance consistency across distributed edge-cloud nodes.

Broader analyses by Felter et al. [18] and Morabito et al. [38] compare container and virtual machine performance, showing that containers introduce minimal CPU overhead and typically outperform VMs in both latency and throughput. Nevertheless, these studies do not consider the nuanced performance demands of robotics perception tasks such as

high-frequency sensor fusion and real-time scheduling constraints which often stress the boundaries of shared CPU and I/O resources in containerized environments.

The absence of detailed benchmarking for robotics-specific workloads within containerized systems represents a critical research gap. This thesis addresses this gap by evaluating the impact of containerization on real-time perception pipelines, assessing how container runtime, system configuration, and resource isolation affect end-to-end latency, sensor throughput, and deployment scalability.

2.6.2 Parallel Processing and Scalability Analysis

Beyond containerization, the efficient execution of robotics algorithms increasingly depends on their ability to leverage parallel computing resources especially when offloading intensive perception tasks to multicore cloud instances. Parallelization transforms bottlenecked single-threaded routines into scalable pipelines, but understanding the real-world scalability limits of robotics workloads remains an open research question.

While Amdahl’s Law provides a theoretical framework for speedup estimation, empirical validation across realistic robotics workloads is limited. Lee et al. [27] demonstrated near-linear scalability of SLAM pipelines up to 8 cores using task-parallel decomposition, yet their findings remain restricted to small-scale embedded systems. Similarly, Adams et al. [3] achieved $3.2\times$ speedup in LiDAR background subtraction via OpenMP on an 8-core system, highlighting the promise of parallelization but falling short of revealing behavior at scale.

Scalability bottlenecks become increasingly prominent beyond 16 cores, where shared memory bandwidth, cache coherence overhead, and thread management begin to limit gains. Murphy et al. [39] theorize these issues but lack extensive experimentation on real robotics workloads under realistic deployment conditions. This lack of empirical study across the full spectrum of 1–64 cores typical of modern cloud infrastructure obscures the practical limits and tuning strategies needed to guide performance-aware deployment decisions.

This thesis contributes to closing this gap through systematic benchmarking of parallelized robotics algorithms, including both image and point cloud pipelines. It quantifies parallel efficiency, scaling behavior, and hardware saturation points under containerized deployments, offering actionable insights into how robotics systems should allocate and utilize compute resources across the edge-cloud continuum.

2.7 Research Gaps and Thesis Positioning

Significant advances in edge computing, cloud infrastructure, communication protocols, and sensor-level data processing have propelled robotics research forward. However, much of this progress remains compartmentalized, with studies focusing narrowly on individual components such as communication strategies, offloading algorithms, or perception pipelines. This fragmented approach overlooks the complex interplay of these elements in realistic hybrid robotics deployments, leaving fundamental questions about end-to-end system performance and integration unanswered.

A prominent gap exists in the lack of integrated system-level evaluations that encompass bandwidth filtering, parallel processing, communication protocol selection, and containerized deployment. While theoretical models, such as Amdahl’s Law, provide valuable insights into parallel performance limits, empirical validation at scales beyond modest core counts remains scarce. This is particularly important for robotics workloads, which pose unique computational and timing challenges compared to general-purpose applications.

Communication protocols widely considered promising for distributed robotics such as MQTT and ZeroMQ have not been thoroughly analyzed under real-world, high-frequency sensor workloads subject to variable network conditions including 5G and WiFi. This limited understanding impedes confident deployment decisions for latency-sensitive and bandwidth-intensive robotics tasks.

Furthermore, while motion detection and background subtraction techniques are common in video analytics, their adaptation and evaluation for bandwidth-constrained robotics perception, especially for LiDAR sensors, are underexplored. The potential for intelligent filtering to reduce data transmission while preserving perception accuracy represents an important yet underutilized opportunity.

Finally, containerization offers reproducibility, modularity, and deployment flexibility but introduces trade-offs in latency, synchronization, and resource isolation that have not been sufficiently characterized for real-time, multi-node edge-cloud robotics systems.

In response, this thesis presents an investigation into hybrid edge-cloud robotics perception architectures. It introduces a scalable, containerized framework that balances local preprocessing with cloud-based resources, evaluates parallel algorithm implementations across 1 to 64 CPU cores, and benchmarks communication protocols under realistic network conditions. Additionally, it develops and compares motion and background filtering strategies tailored for camera and LiDAR data, while rigorously quantifying containerization’s impact on real-time system performance.

Chapter 3

Experimental Setup and Framework

3.1 Introduction

This chapter details the experimental environment developed to evaluate the performance of distributed ROS-based systems deployed in containerized infrastructures. The framework builds upon the comprehensive 5G-based Intelligent Transportation System developed by the Mechatronic Vehicle Systems (MVS) Lab at the University of Waterloo [40, 51, 61], which provides the foundational infrastructure and operational context for investigating hybrid edge-cloud processing trade-offs in real-world sensor networks.

A key focus is placed on the practical setup and validation of the testing environment, including the configuration of Docker containers, the integration of ROS Noetic, and the establishment of reliable base conditions to ensure reproducibility and consistency. The framework supports hybrid communication architectures, enabling systematic evaluation of computation and communication across edge and cloud nodes. Through controlled variation of CPU core allocations, node counts, and communication protocols, the environment facilitates detailed measurement of processing efficiency, network latency, and system scalability. This foundation ensures that the experimental results accurately reflect real-world deployment scenarios and serve as a robust basis for analyzing the trade-offs and performance characteristics of containerized robotics applications.

3.2 Foundational Project Context: MVS Lab 5G Intelligent Transportation System

3.2.1 Overview of the Existing System

This research builds upon the MVS Lab’s comprehensive 5G-based Intelligent Transportation System [40], which represents one of the most extensive real-world deployments of connected sensor networks for autonomous transportation applications. The existing system comprises 14 Intelligent Sensor Nodes (ISNs) deployed across the University of Waterloo’s north campus, creating a comprehensive sensor network that covers both outdoor infrastructure monitoring and indoor facility management.

Each ISN integrates multiple high-resolution sensors (dual cameras and LiDAR), substantial edge computing capability (Jetson Orin NX), and 5G connectivity, enabling real-time data collection and transmission to cloud-based perception systems. The system has successfully demonstrated complex applications including autonomous shuttle bus navigation, pedestrian safety warnings, traffic rule violation detection, and comprehensive dataset collection under adverse weather conditions including snow and freezing rain [61].

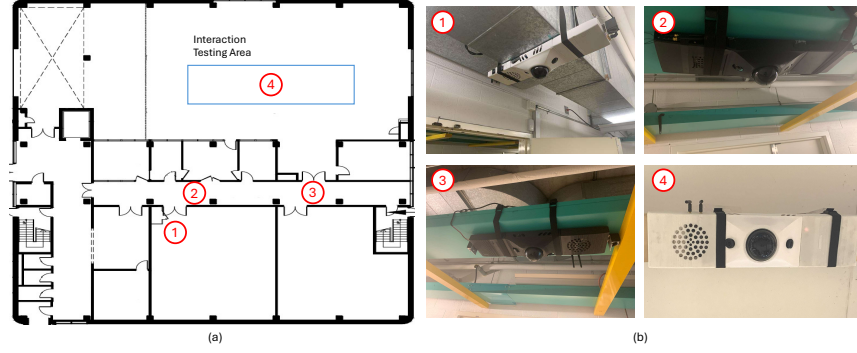
3.2.2 Existing Architecture and Research Motivation

The current MVS Lab architecture implements a sophisticated but resource-intensive approach where each ISN performs complete local sensor fusion and object detection using GPU-accelerated Jetson hardware [51]. While this edge-centric approach has demonstrated significant capabilities, it also reveals several fundamental limitations that motivate the hybrid approach investigated in this thesis:

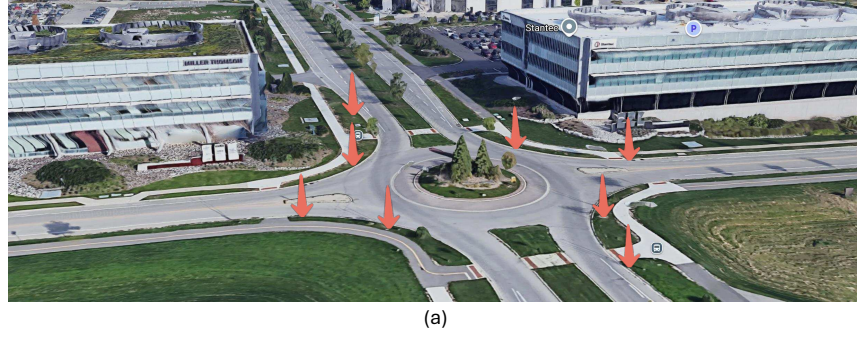
High Hardware Costs: Each ISN requires a Jetson Orin NX unit (approximately \$899) to support local GPU-accelerated processing, resulting in significant capital expenditure that scales linearly with deployment size.

Processing Bottlenecks: While Jetson hardware provides substantial computational capability, it remains constrained by thermal and power limitations that prevent deployment of the most sophisticated perception algorithms.

Maintenance Complexity: Each ISN requires individual software updates, parameter tuning, and fault diagnosis, creating operational overhead that increases with deployment scale.



(a) Indoor facility management deployment.



(b) Outdoor traffic monitoring deployment.

Figure 3.1: MVS Lab 5G Intelligent Transportation System: 14 ISNs deployed across indoor and outdoor environments [40].

Resource Utilization Inefficiency: Edge computational load varies significantly based on scene complexity and environmental conditions, leading to periods of both under-utilization and potential overload that cannot be dynamically balanced across the network.

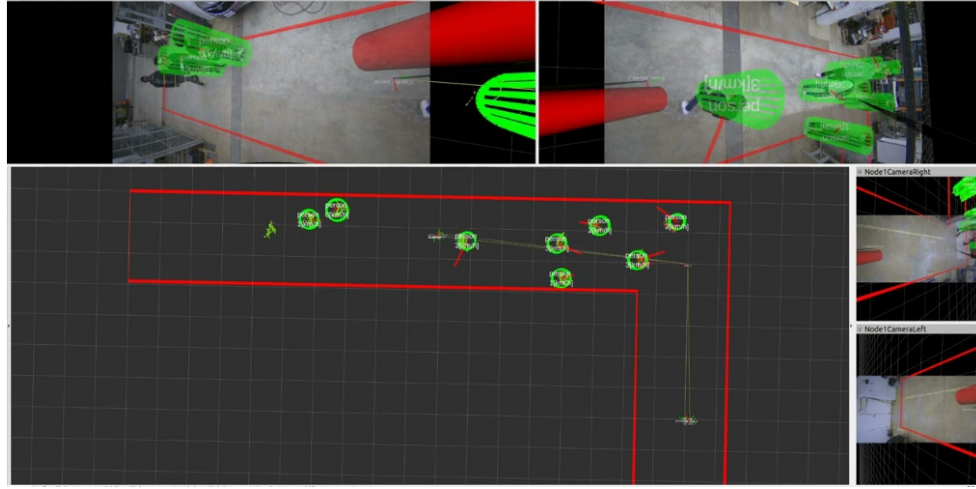


Figure 3.2: Multi-sensor data fusion demonstration showing complementary information from multiple LiDAR sensors [40].

Multi-Sensor Data Fusion Advantages: A critical advantage of the current distributed ISN architecture lies in its ability to leverage multiple sensors and cameras to provide complementary information that can be fused for enhanced perception accuracy. As demonstrated in Figure 3.2, different sensors capture overlapping but distinct aspects of the same scene, for instance, one LiDAR sensor (shown in red) may detect objects from a particular vantage point while another LiDAR sensor (shown in green) provides additional coverage from a different perspective. This multi-modal sensor fusion enables several key benefits:

- **Enhanced Coverage:** Multiple sensors eliminate blind spots and provide redundant observations of critical areas
- **Improved Accuracy:** Cross-validation between sensors reduces false positives and increases detection confidence
- **Robustness:** Sensor failures or environmental interference affecting one sensor can be compensated by others

- **Comprehensive Scene Understanding:** Different sensor modalities (LiDAR, cameras, radar) capture complementary physical properties, creating a more complete environmental representation

However, the current edge-centric approach limits the sophisticated algorithms needed to fully exploit these multi-sensor advantages due to computational constraints at individual ISNs.

This thesis explores a fundamentally different architectural approach: intelligent task partitioning between lightweight edge processing and scalable cloud computation, using the MVS Lab infrastructure as both foundation and baseline for comparison.

3.3 Experimental System Architecture Building on MVS Lab Infrastructure

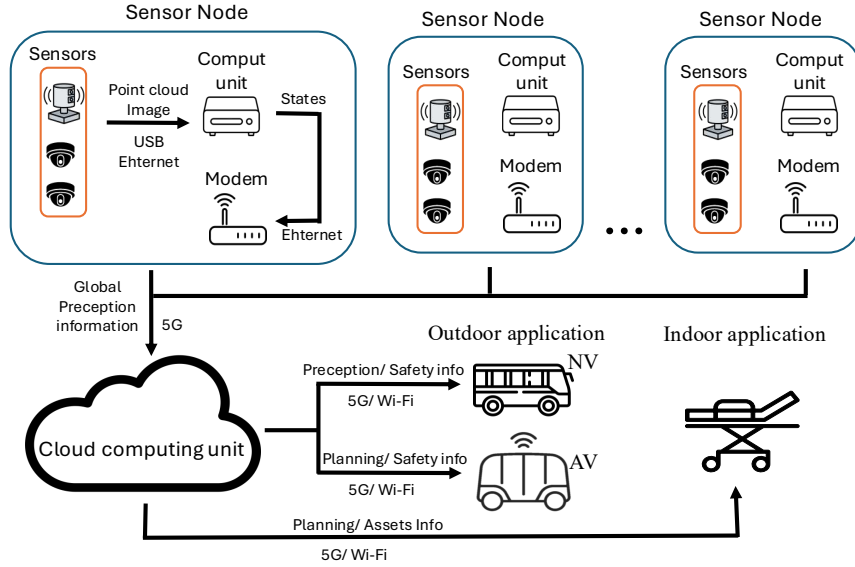


Figure 3.3: MVS Lab Distributed System Architecture [40].

3.3.1 Foundation: MVS Lab Architecture

As shown in Figure 3.3, the MVS Lab system establishes a comprehensive distributed architecture comprising sensor nodes, edge processing units, and centralized cloud coordi-

nation. This proven infrastructure provides the foundational framework upon which the hybrid edge-cloud approach investigated in this thesis is built and evaluated.

The existing MVS Lab architecture supports both indoor and outdoor deployments for autonomous and non-autonomous platforms, with key components including distributed sensor nodes, substantial edge processing capability, and cloud-based coordination and visualization systems.

3.3.2 Experimental Hybrid Architecture

The hybrid architecture builds on the existing MVS Lab sensor network but rethinks the distribution of processing tasks. Instead of performing full sensor processing at the edge, the system uses lightweight, containerized filters (e.g., motion detection and background subtraction) on edge nodes to pre-process high-frequency data. These operations reduce bandwidth usage by removing static or redundant content before offloading the heavier computational workloads (clustering, fusion, and decision-making) to the cloud.

Rationale and Methodology for Edge-Cloud Task Partitioning

The partitioning strategy between edge and cloud processing is based on a systematic analysis of computational complexity, latency requirements, and bandwidth constraints. This methodology considers four key factors to determine the optimal placement of processing tasks:

Computational Complexity Analysis: Tasks are categorized by their computational intensity and scalability characteristics. Lightweight operations such as motion detection and background subtraction, which exhibit linear scaling with data size, are assigned to edge nodes where they can be executed with predictable latency. Computationally intensive tasks like clustering and object recognition, which benefit from parallel processing and large memory pools, are delegated to cloud resources.

Latency Sensitivity Classification: Processing tasks are classified based on their real-time requirements. Time-critical operations requiring sub-10ms response times (such as emergency braking signals or collision avoidance) remain on edge devices to ensure deterministic performance. Tasks with relaxed timing constraints (such as detailed scene analysis or historical data correlation) are candidates for cloud offloading.

Data Dependencies and Bandwidth Optimization: The partitioning strategy prioritizes operations that can significantly reduce data transmission volume. Edge-based

filtering operations that can eliminate 80-90% of redundant data are preferred, as they minimize the bandwidth requirements for subsequent cloud processing while preserving essential scene information.

Fault Tolerance and Graceful Degradation: The architecture ensures that edge nodes can maintain basic functionality even when cloud connectivity is degraded. Critical safety functions and basic perception capabilities are preserved locally, while advanced features gracefully degrade when network conditions deteriorate. This systematic approach to task partitioning provides a replicable methodology that can be adapted to different system configurations and operational This architecture introduces several key differences from the baseline:

- **Task Partitioning:** Low-latency filtering runs on the edge; compute-intensive tasks run in the cloud.
- **Dynamic Resource Use:** CPU cores and memory are provisioned elastically in the cloud to match load, unlike fixed resources on edge devices.
- **Hybrid Communication:** ROS topics handle intra-node coordination; ZeroMQ bridges edge-cloud layers, replacing the MVS Lab’s static 5G-only dependency.
- **Modular Deployment:** All components are containerized for flexible orchestration and reproducibility.
- **Unified Monitoring:** Prometheus and ROS logging capture distributed system metrics, enabling side-by-side comparison of edge, cloud, and hybrid modes.

By shifting part of the workload to a scalable backend and retaining key pre-processing at the edge, this architecture aims to achieve better cost-efficiency, adaptability, and scalability particularly in environments with constrained network or processing resources.

3.3.3 Justification for Edge-Level Motion Detection Without Object Identity

A key design decision in the hybrid architecture is the delegation of initial motion detection to edge nodes, even though full object classification and semantic understanding are deferred to the cloud. While at first this might appear limiting after all, knowing that “something moved” is not the same as knowing what it was this strategy is both practical and essential in real-time robotics applications constrained by network latency.

The average end-to-end latency for cloud processing over 5G, measured in Section ??, is approximately **117.9 ms**. This delay, though acceptable for backend analysis, is too high for initial decision-making in time-sensitive systems. Consequently, the edge must retain the capability for fast reflexes, even if it lacks full situational awareness.

This architectural choice parallels biological systems: the edge functions like a reflex arc, reacting immediately to stimuli (e.g., sudden motion), while the cloud plays the role of the brain, providing deeper analysis after a short delay. For example, motion detection on the edge enables:

- **Bandwidth Reduction:** Only frames containing dynamic content are forwarded, reducing transmission volume by up to 80%.
- **Event-Driven Activation:** Downstream modules in the cloud are activated only upon relevant stimuli, conserving compute resources.
- **Preemptive Action:** Lightweight decisions such as slowing down a robot or triggering a warning can be made without full classification.

These benefits justify the inclusion of edge-level motion filtering in the hybrid design. Once motion is detected, the filtered foreground can be sent to the cloud for clustering and semantic interpretation. This architecture achieves a balance: the edge reacts quickly and locally, while the cloud interprets globally and accurately despite inherent network delays.

3.3.4 Containerized Deployment and ROS Architecture

All processing nodes are deployed within Docker containers to ensure reproducibility, scalability, and environment consistency across diverse hardware platforms. Each container is built on a custom image extending the official ROS Noetic distribution, with additional dependencies pre-installed for sensor data processing, parallel computation, and ZeroMQ-based communication.

ZeroMQ replaces MQTT for inter-container and edge-cloud communication due to its superior performance under high-frequency message loads. The custom containers include compiled ZeroMQ libraries and lightweight ROS node wrappers that enable seamless integration with ROS topics. This architecture allows each sensor node to operate independently within an isolated environment while supporting efficient, low-latency data exchange across distributed components.

Docker Configuration

The base Docker image configuration:

Containers are launched with explicitly allocated CPU and memory resources using Docker’s `--cpuset-cpus` and `--memory` flags to enforce resource isolation.

Table 3.1 to Table 3.2 summarizes the range of container configurations used during performance evaluations, covering baseline, multi-node, and high-load scenarios.

ROS Node Architecture

Within each container, a modular ROS node architecture is deployed, with each node designed to handle a specific task in the processing pipeline. The main nodes include:

- **sensor_publisher:** Publishes raw LiDAR and camera data from recorded datasets
- **background_subtraction:** Implements background removal algorithms for LiDAR data
- **lidar_clustering:** Applies DBSCAN clustering to extract objects from processed point clouds
- **motion_detection:** Executes motion detection on video frames to identify dynamic elements
- **zeromq_bridge:** Facilitates distributed coordination and cloud interaction via ZeroMQ
- **performance_monitor:** Monitors and logs performance metrics at high frequency

Each node is optimized for specific resource allocations and inter-node communication patterns, ensuring scalable deployment across edge and cloud scenarios.

3.4 Sensor Processing Algorithms

3.4.1 LiDAR Processing Pipeline

As shown in Figure 3.4 the LiDAR processing pipeline consists of two core stages:

Table 3.1: Low-to-Medium Load Docker Configurations (20 Nodes)

Config ID	CPU Cores	Memory (GB)	Nodes	Network Type	Communication
Single-Node Setups					
C1N1	1	1	1	Local	ROS Topics
C4N1	4	4	1	Local	ROS Topics
C8N1	8	8	1	Local	ROS Topics
C16N1	16	16	1	Local	ROS Topics
C32N1	32	32	1	Local	ROS Topics
C64N1	64	64	1	Local	ROS Topics
C128N1	128	128	1	Local	ROS Topics
Distributed Setups (20 Nodes)					
C4N5	4	4	5	Distributed	ROS + ZeroMQ
C4N10	4	4	10	Distributed	ROS + ZeroMQ
C4N15	4	4	15	Distributed	ROS + ZeroMQ
C4N20	4	4	20	Distributed	ROS + ZeroMQ
C8N5	8	8	5	Distributed	ROS + ZeroMQ
C8N10	8	8	10	Distributed	ROS + ZeroMQ
C8N15	8	8	15	Distributed	ROS + ZeroMQ
C8N20	8	8	20	Distributed	ROS + ZeroMQ
C16N5	16	16	5	Distributed	ROS + ZeroMQ
C16N10	16	16	10	Distributed	ROS + ZeroMQ
C16N15	16	16	15	Distributed	ROS + ZeroMQ
C16N20	16	16	20	Distributed	ROS + ZeroMQ
C32N5	32	32	5	Distributed	ROS + ZeroMQ
C32N10	32	32	10	Distributed	ROS + ZeroMQ
C32N15	32	32	15	Distributed	ROS + ZeroMQ
C32N20	32	32	20	Distributed	ROS + ZeroMQ
C64N5	64	64	5	Distributed	ROS + ZeroMQ
C64N10	64	64	10	Distributed	ROS + ZeroMQ
C64N15	64	64	15	Distributed	ROS + ZeroMQ
C64N20	64	64	20	Distributed	ROS + ZeroMQ
C128N5	128	128	5	Distributed	ROS + ZeroMQ
C128N10	128	128	10	Distributed	ROS + ZeroMQ
C128N15	128	128	15	Distributed	ROS + ZeroMQ
C128N20	128	128	20	Distributed	ROS + ZeroMQ

Table 3.2: High-Load Docker Configurations (>20 Nodes)

Config ID	CPU Cores	Memory (GB)	Nodes	Network Type	Communication
C8N50	8	8	50	Distributed	ROS + ZeroMQ
C16N50	16	16	50	Distributed	ROS + ZeroMQ
C16N75	16	16	75	Distributed	ROS + ZeroMQ
C32N50	32	32	50	Distributed	ROS + ZeroMQ
C32N75	32	32	75	Distributed	ROS + ZeroMQ
C64N50	64	64	50	Distributed	ROS + ZeroMQ
C64N75	64	64	75	Distributed	ROS + ZeroMQ
C64N100	64	64	100	Distributed	ROS + ZeroMQ
C128N50	128	128	50	Distributed	ROS + ZeroMQ
C128N75	128	128	75	Distributed	ROS + ZeroMQ
C128N100	128	128	100	Distributed	ROS + ZeroMQ

Background Subtraction: Static environmental features are removed to isolate dynamic objects using a temporal point cloud differencing method with adaptive thresholding.

Clustering Algorithm: The remaining foreground points are grouped into object-level clusters using a DBSCAN-based approach. The clustering parameters are tuned based on typical vehicle and pedestrian dimensions to ensure spatially coherent object boundaries.

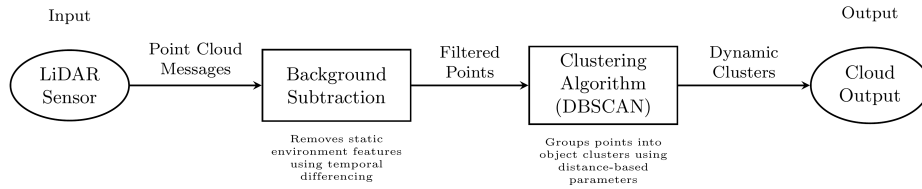


Figure 3.4: LiDAR processing pipeline: raw point clouds undergo background subtraction to remove static features, followed by clustering to group dynamic objects.

To improve computational efficiency, the original single-threaded implementation was refactored into a multithreaded design using Intel’s Threading Building Blocks (TBB) library. Specifically, the pipeline utilizes `tbb::parallel_for` constructs to distribute independent computations across CPU cores in a task-parallel fashion. Figure 3.5 shows the overview of the algorithm functionality.

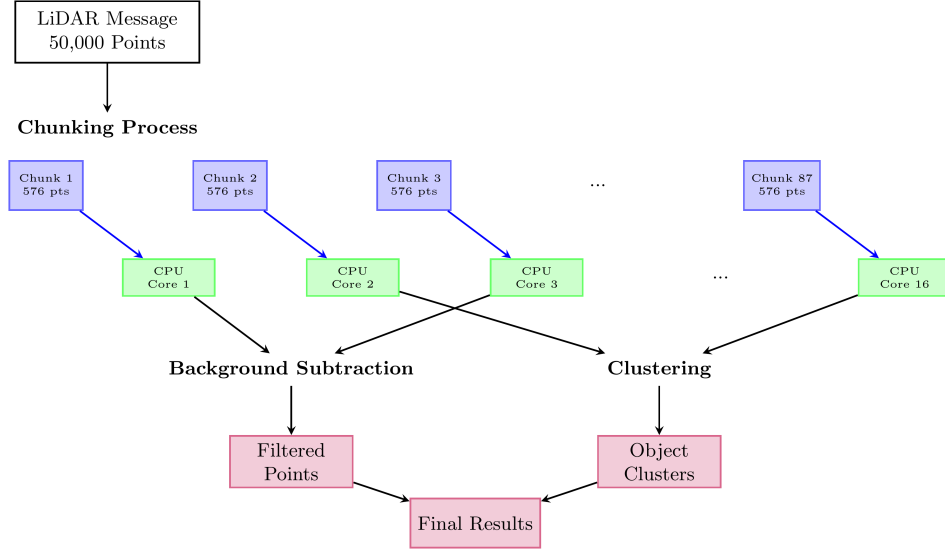


Figure 3.5: Chunk-based parallelization strategy for LiDAR processing.

A central performance optimization is the use of **chunking**, where each incoming LiDAR message (typically containing around 50,000 points) is subdivided into smaller, spatially contiguous chunks. For example, 87 chunks of 576 points each may be used. These chunks are processed independently in parallel, significantly reducing per-frame latency.

The chunk size is implemented as a **tunable parameter**, allowing for adaptation to different hardware configurations or point cloud densities. Smaller chunks improve load balancing and increase parallel granularity, while larger chunks reduce scheduling overhead.

The parallelized background subtraction stage demonstrates near-linear scalability up to 16 CPU cores, as it involves primarily independent operations across points. In contrast, the clustering stage shows diminished returns beyond 8 cores due to shared-memory coordination required for cluster merging at chunk boundaries.

This multithreaded, chunk-based architecture enables the pipeline to meet real-time processing requirements even under high-throughput conditions.

3.4.2 Camera Processing Algorithms

The camera processing pipeline implements three motion detection algorithms for comparative performance analysis. These algorithms, Frame Differencing, Gaussian Mixture

Models (GMM), and Dense Optical Flow, each represent a trade-off between computational efficiency, memory usage, and detection robustness. Their inclusion enables analysis of how motion filtering impacts both perception quality and bandwidth optimization in hybrid edge-cloud deployments.

Frame Differencing

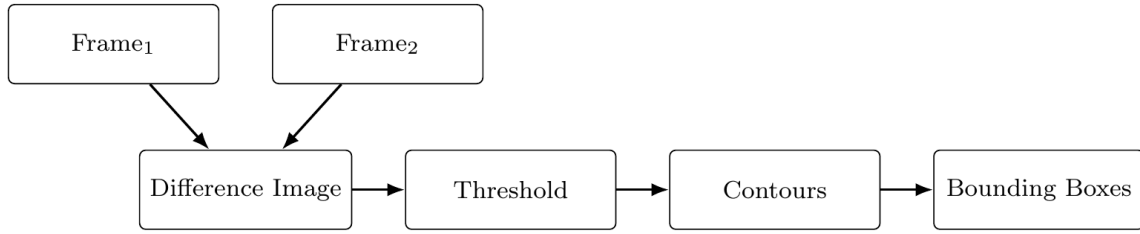


Figure 3.6: Frame differencing processing pipeline

Frame differencing is the simplest of the three methods. It works by computing the absolute pixel-wise difference between two consecutive grayscale frames. When the difference at a pixel exceeds a fixed threshold, that pixel is marked as part of the motion mask. While fast and highly parallelizable, this method is sensitive to noise and lighting changes and may fail to detect slowly moving or gradually changing objects.

Its low computational load makes it attractive for lightweight edge nodes where throughput is prioritized over fine-grained accuracy.

Algorithm 1 Frame Differencing Motion Detection

```
1: Input: Two consecutive frames  $F_1$ ,  $F_2$ , and threshold  $\tau = 25$ 
2: Output: Motion mask  $M$ 
3:  $G_1 \leftarrow \text{ConvertToGrayscale}(F_1)$ 
4:  $G_2 \leftarrow \text{ConvertToGrayscale}(F_2)$ 
5:  $D \leftarrow |G_1 - G_2|$  ▷ Compute absolute difference
6: for each pixel  $(i, j)$  in  $D$  do
7:   if  $D(i, j) > \tau$  then
8:      $M(i, j) \leftarrow 255$  ▷ Motion detected
9:   else
10:     $M(i, j) \leftarrow 0$  ▷ No motion
11:   end if
12: end for
13: return  $M$ 
```

Gaussian Mixture Models (GMM)



Figure 3.7: Gaussian Mixture Model (GMM) processing pipeline

GMM is a statistical background subtraction method that models each pixel as a mixture of Gaussians, enabling adaptation to gradual illumination changes, repetitive motion (e.g., waving trees), or dynamic backgrounds. Unlike frame differencing, it maintains a long-term memory of background appearance and can distinguish between transient and persistent changes.

Though more computationally expensive and memory-intensive, GMM significantly improves detection stability and reduces false positives in outdoor or cluttered scenes. Morphological operations are applied post-processing to refine the motion mask by eliminating small noise regions and filling holes.

Algorithm 2 GMM Background Subtraction

```
1: Input: Frame  $F$ , background model  $\mathcal{M}$ , learning rate  $\alpha = 0.01$ 
2: Output: Foreground mask  $M_{fg}$ 
3:  $M_{fg} \leftarrow \text{APPLYGMM}(F, \mathcal{M})$  ▷ Apply GMM to extract foreground
4:  $\mathcal{M} \leftarrow \text{UPDATEMODEL}(\mathcal{M}, F, \alpha)$  ▷ Update the background model
5:  $K \leftarrow \text{CREATEKERNEL}(\text{ELLIPSE}, 5 \times 5)$  ▷ Create morphological kernel
6:  $M_{fg} \leftarrow \text{MORPHOPEN}(M_{fg}, K)$  ▷ Remove small noise
7:  $M_{fg} \leftarrow \text{MORPHCLOSE}(M_{fg}, K)$  ▷ Fill small holes
8: return  $M_{fg}$ 
```

Dense Optical Flow

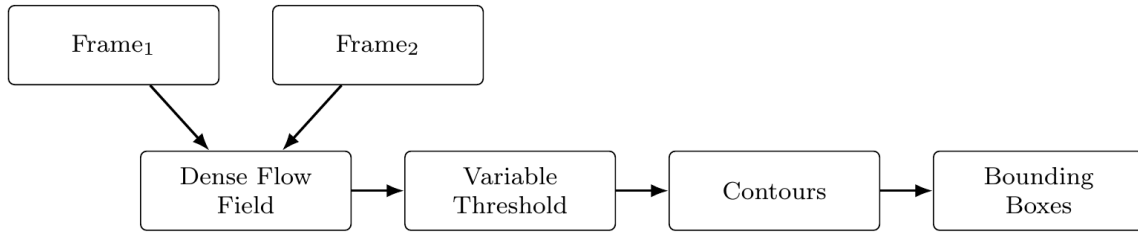


Figure 3.8: Dense optical flow processing pipeline

Dense optical flow estimates per-pixel motion vectors between two consecutive frames, producing a high-resolution motion field that captures subtle and non-rigid motion. This method is particularly effective for detecting partial motion, occlusions, or movement in deformable objects. It can also identify motion even in areas with low contrast where differencing or GMM would fail.

However, its accuracy comes at a high computational cost, making it more suitable for cloud-side processing or high-performance edge nodes. In this implementation, a magnitude threshold is applied to the optical flow vectors to generate a binary motion mask.

Algorithm 3 Dense Optical Flow Motion Detection

```
1: Input: Frames  $F_1, F_2$ ; flow magnitude threshold  $\tau_{\text{flow}}$ 
2: Output: Motion mask  $M$ 
3:  $G_1 \leftarrow \text{CONVERTToGRAYSCALE}(F_1)$ 
4:  $G_2 \leftarrow \text{CONVERTToGRAYSCALE}(F_2)$ 
5:  $\text{Flow} \leftarrow \text{COMPUTEDENSEOPTICALFLOW}(G_1, G_2)$  ▷ Estimate flow vector for each pixel
6: for each pixel flow vector  $(u, v)$  in  $\text{Flow}$  do
7:    $\text{magnitude} \leftarrow \sqrt{u^2 + v^2}$  ▷ Compute flow magnitude
8:   if  $\text{magnitude} > \tau_{\text{flow}}$  then
9:      $M[\text{pixel}] \leftarrow \text{motion}$ 
10:  else
11:     $M[\text{pixel}] \leftarrow \text{background}$ 
12:  end if
13: end for
14: return  $M$ 
```

3.4.3 Communication Protocol Implementation

The system adopts a hybrid communication architecture tailored to the specific requirements of local edge processing, hybrid edge-cloud coordination, and cloud-level analytics. Figure 3.9 illustrates the integration of the various protocols across system components.

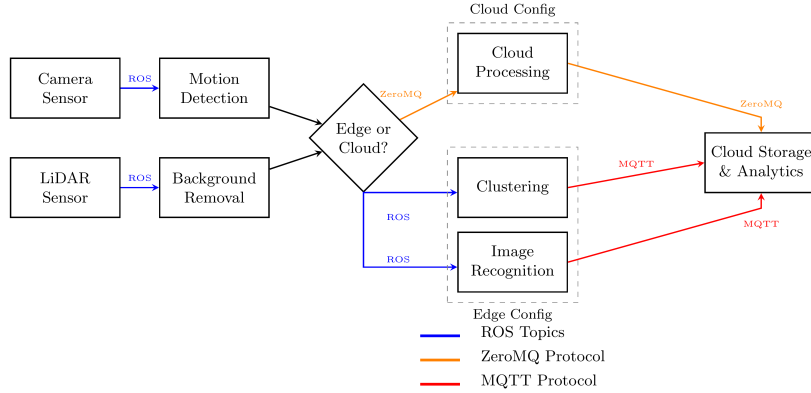


Figure 3.9: Hybrid Communication Protocol Architecture: ROS topics enable local edge communication; ZeroMQ handles hybrid edge-cloud messaging; MQTT provides reliable cloud integration for final result transmission.

Local and Edge Communication (ROS Topics): For internal communication within edge nodes and between sensors and processing modules, ROS topics are em-

ployed. ROS is well-suited for managing high-frequency data streams such as camera images (`sensor_msgs/Image`) and LiDAR point clouds (`sensor_msgs/PointCloud2`) with minimal latency and efficient memory usage via zero-copy mechanisms. This native ROS communication supports real-time processing and low-latency inter-process data sharing critical to edge operations.

Hybrid Edge-Cloud Communication (ZeroMQ): When processing tasks are partitioned between the edge and cloud, ZeroMQ serves as the backbone communication protocol. ZeroMQ excels at handling high-frequency messaging with flexible patterns such as publisher-subscriber for distributing sensor analytics, push-pull for load balancing clustering workloads, and request-reply for synchronous machine learning inference. Its language-agnostic design and scalability make it ideal for hybrid deployments that require fast, reliable communication between distributed components. ZeroMQ also incorporates features like automatic service discovery, load balancing, failover, and message buffering, ensuring robust connectivity despite network variability.

Cloud Integration via MQTT: In scenarios where edge nodes perform complete processing locally and only final results or summaries need to be transmitted, MQTT is used for its lightweight, reliable delivery suited to constrained bandwidth conditions. MQTT's publish-subscribe model with persistent queues provides guaranteed message delivery and seamless integration with cloud storage and analytics services.

The following quality of service (QoS) levels are enforced based on message criticality and frequency:

- **Level 0** (at most once): For high-frequency sensor streams where occasional data loss is acceptable.
- **Level 1** (at least once): For control commands and system status updates.
- **Level 2** (exactly once): Reserved for critical safety messages and configuration changes.

A custom ZeroMQ bridge translates messages bidirectionally between ROS topics and ZeroMQ, performing schema validation and applying LZ4 compression to optimize data transmission. This bridge handles serialization formats including JSON for configuration messages, Protocol Buffers for structured sensor data, and raw binary for image streams.

Performance optimizations include adaptive compression tuned to available bandwidth, prioritization of critical messages to reduce latency under congestion, selective transmission of only changed data, connection pooling to minimize overhead, message batching to improve throughput, and asynchronous processing for real-time responsiveness.

In summary, this hybrid communication framework leverages ROS for low-latency internal processing, ZeroMQ for efficient high-frequency hybrid edge-cloud messaging, and MQTT for reliable cloud integration when edge nodes handle complete processing and forward only final results. This design ensures scalable, robust, and latency-aware communication tailored to diverse system requirements.

3.5 Network Condition Simulation

3.5.1 Traffic Control (tc) Configuration

To evaluate system performance under various network conditions, the experimental setup implements network emulation using Linux Traffic Control (tc). This tool enables precise control over network latency, bandwidth, and packet loss characteristics between the edge node (node0) and cloud node (node1).

Network Scenario Implementation

The following network scenarios are systematically tested to represent realistic deployment conditions:

Table 3.3: Network Configuration Scenarios

Network Type	Latency (ms)	Bandwidth	Packet Loss	Distribution
Ideal Network	0	Unlimited	0%	Native
5G Network	5 ± 2	1 Gbps	0.01%	Normal
WiFi Network	20 ± 5	100 Mbps	0.1%	Normal
Degraded Network	100 ± 20	10 Mbps	1%	Pareto

Each network condition incorporates realistic variability through statistical distributions to simulate real-world network behavior. The normal distribution models typical network jitter, while the Pareto distribution captures the heavy-tailed behavior characteristic of degraded network conditions.

Network Configuration Commands

The tc configuration commands are systematically applied to create reproducible network conditions:

```

1  # Ideal Network (baseline - no artificial limitations)
2  sudo tc qdisc del dev ens1f0 root
3
4  # 5G Network simulation
5  sudo tc qdisc add dev ens1f0 root handle 1: netem \
6      delay 5ms 2ms distribution normal loss 0.01%
7  sudo tc qdisc add dev ens1f0 parent 1:1 handle 10: \
8      tbf rate 1000mbit burst 32kbit latency 10ms
9
10 # WiFi Network simulation
11 sudo tc qdisc add dev ens1f0 root handle 1: netem \
12     delay 20ms 5ms distribution normal loss 0.1%
13 sudo tc qdisc add dev ens1f0 parent 1:1 handle 10: \
14     tbf rate 100mbit burst 16kbit latency 50ms
15
16 # Degraded Network simulation
17 sudo tc qdisc add dev ens1f0 root handle 1: netem \
18     delay 100ms 20ms distribution pareto loss 1%
19 sudo tc qdisc add dev ens1f0 parent 1:1 handle 10: \
20     tbf rate 10mbit burst 8kbit latency 200ms

```

Listing 3.1: Network Emulation Commands

3.5.2 Network Performance Validation

To ensure the accuracy of the network emulation, comprehensive validation testing is performed using standardized network measurement tools. Latency was measured using ICMP ping, with each test sending 100 packets to gather round-trip time statistics and observe jitter behavior. Throughput was evaluated using `iperf3`, running 30-second TCP sessions to quantify bidirectional bandwidth between edge and cloud nodes. In parallel, UDP tests were conducted at a fixed 50 Mbps stream rate to capture packet loss and jitter under typical sensor workloads. Finally, a composite reliability score was calculated for each configuration based on latency consistency and packet integrity.

Validation Results

Table 3.4 presents the measured performance of the emulated network conditions. Across all test scenarios, the measured latencies closely matched the configured targets, confirming

the effectiveness and precision of the `tc` emulation setup. Each network condition demonstrates not only adherence to the target values but also realistic variability that mirrors real-world deployment environments.

The baseline "Ideal" configuration exhibited minimal latency (0.160 ms) with negligible variance, serving as a reference point for evaluating the system's native performance without network interference. In the 5G simulation, latency averaged 5.173 ms with controlled jitter, falling well within the 5 ± 2 ms specification and validating its suitability for ultra-low-latency robotics communication.

The WiFi scenario produced average latencies of 20.628 ms, consistent with the expected 20 ± 5 ms range. This configuration captures the moderate variability typical in wireless deployments and provides a reliable baseline for evaluating communication protocol robustness in realistic indoor or campus-scale environments.

The degraded network scenario, designed to simulate poor connectivity (e.g., rural or congested urban environments), maintained a mean latency of 99.241 ms with an expected increase in jitter and a packet loss rate of 1%. The reliability score (0.975) reflects the anticipated performance drop in such conditions, yet the system remained stable and predictable.

These results confirm that the network emulation framework provides fine-grained control over latency, bandwidth, and packet loss, enabling systematic and repeatable evaluation of distributed robotics workloads under a wide range of connectivity constraints.

Table 3.4: Network Performance Validation Results

Network Condition	Measured Latency (ms)	Target Latency (ms)	Packet Loss (%)	Reliability Score	Validation Status
Ideal	0.160 ± 0.024	0	0.0	1.000	Baseline
5G	5.173 ± 2.117	5 ± 2	0.0	0.998	Accurate
WiFi	20.628 ± 5.2	20 ± 5	0.0	0.996	Accurate
Degraded	99.241 ± 28.7	100 ± 20	1.0	0.975	Accurate

3.6 Performance Evaluation Framework

With the network emulation and monitoring infrastructure validated, we shift focus to the core performance evaluation framework used to analyze the behavior of the distributed

system under varying conditions. The goal is to quantify how well the containerized, ROS-based setup performs in terms of sensor data processing, communication efficiency, and scalability across edge and cloud nodes.

This evaluation encompasses a broad range of system metrics spanning from raw processing times to network overhead captured at high frequency to enable fine-grained performance analysis. The framework is designed to reflect realistic operational demands, ensuring that the results are directly applicable to practical robotics deployments.

Metrics Definition and Collection

The evaluation framework captures performance across multiple system dimensions critical to real-time, distributed LiDAR and camera processing. Table 3.5 provides a comprehensive list of collected metrics.

Table 3.5: Performance Metrics for Distributed ROS System Evaluation

Category	Metric	Unit	Description
Processing	LiDAR Background Time	ms	Background subtraction execution time
	LiDAR Clustering Time	ms	Object clustering execution time
	Camera Processing Time	ms	Motion detection execution time
	Total Pipeline Time	ms	End-to-end sensor data processing latency
	Throughput	msg/s	Number of messages successfully processed per second
	Message Loss Rate	%	Dropped or delayed message rate
System	CPU Utilization	%	Average CPU usage across all allocated cores
	Memory Usage	MB	Maximum memory consumption per container
	Network Latency	ms	Inter-node communication delay
Scalability	Throughput	Hz	Message handling capacity under increasing load
	Processing Time	ms	Average time to process one message
	Message Loss Rate	%	Reliability under load (same as above)
	Node Efficiency	%	Throughput per core normalized across nodes
Network	Bandwidth Usage	Mbps	Total ROS + ZeroMQ traffic
	Packet Loss Rate	%	Transport reliability
	Protocol Overhead	%	Non-payload communication cost

High-Frequency Data Collection

To capture performance data with sufficient granularity, all metrics were logged at a high sampling rate of 10 Hz and 20 Hz, corresponding to 0.1 and 0.05-second intervals. This

ensured that transient behavior such as momentary latency spikes or throughput drops could be observed and analyzed in detail.

Each ROS container was instrumented with lightweight logging nodes that tracked topic statistics, end-to-end processing latency, and sensor stream throughput in real time. Host- and container-level metrics, including CPU usage, memory consumption, and I/O statistics, were collected using Docker’s built-in monitoring tools and the `psutil` Python library. These system-level observations were aligned with message-level timing through precise clock synchronization achieved using the `chrony` daemon. This synchronization was critical to ensuring consistency between the edge and cloud nodes, especially in scenarios with variable network latency.

To emulate realistic workloads, pre-recorded LiDAR data streams were played back at controlled frequencies of 10 Hz and 20 Hz, simulating the sensor rates observed in actual robotic platforms. All logged data were stored in standardized formats (CSV and ROS bag files) to support consistent analysis across configurations.

3.7 Integration with MVS Lab Dataset and Infrastructure

3.7.1 Dataset Utilization from MVS Lab Deployment

The experimental evaluation leverages datasets collected from the MVS Lab ISN deployment to ensure realistic workload characteristics and enable direct performance comparison with the existing edge-centric approach. The datasets include synchronized multi-sensor data from outdoor traffic monitoring, indoor facility management, and adverse weather conditions that represent the operational challenges encountered in real-world deployments.

Specific dataset characteristics utilized in the evaluation include:

- **Outdoor Traffic Scenarios:** Multi-sensor data from roundabout interactions, pedestrian crossings, and complex vehicle maneuvers captured during normal and adverse weather conditions including snow and freezing rain [61].
- **Multi-Node Cooperative Perception:** Synchronized data from the 14 ISN deployment enabling evaluation of distributed processing coordination and bandwidth optimization across multiple simultaneous sensor streams.

- **Indoor Human Tracking:** Multi-node sensor data from indoor environments including path tracking, interactive planning, and UWB-based localization scenarios demonstrating system versatility across deployment environments.
- **5G Network Performance Data:** Real-world network performance characteristics from the campus 5G deployment providing validation data for the simulated network conditions used in controlled experiments.

3.7.2 Trade-Offs and Value Assessment Against Edge-Centric Baselines

The experimental framework is designed to assess the added value and trade-offs introduced by hybrid edge-cloud architectures in comparison to the existing edge-centric model deployed by the MVS Lab. Rather than seeking to replace the baseline system, the focus is on understanding under what conditions the hybrid approach offers performance, scalability, or efficiency benefits.

This evaluation considers several critical factors, including processing latency, bandwidth utilization, CPU core efficiency, system responsiveness, and network adaptability measured under equivalent sensor loads and environmental configurations. In doing so, it highlights situations where shifting selective workloads to the cloud improves responsiveness or reduces node resource pressure, while also identifying the latency and complexity penalties incurred under constrained network conditions.

Additionally, the analysis quantifies trade-offs in deployment complexity, synchronization requirements, and cost overhead associated with hybrid communication and orchestration. This holistic view supports informed architectural decisions tailored to specific operational constraints, providing practical insight into when edge-only systems are sufficient and when hybrid designs unlock meaningful performance or deployment flexibility.

3.7.3 Experimental Validation

To ensure the reliability of results, each experimental configuration was evaluated in a rigorously controlled environment. All tests were performed on a fixed hardware platform with explicitly allocated CPU and memory limits to eliminate variability due to background interference. A baseline run under minimal load (e.g., single-core, single-node configuration) was established for every test series, serving as an anchor for performance comparison. Each scenario was executed 30 times to account for runtime variability, with mean and

standard deviation computed across trials to ensure statistical confidence. Metrics such as CPU usage, memory load, throughput, and latency were measured using pre-calibrated tools validated against known benchmarks. The datasets used for each test are documented in their respective sections, including topic structure, frame rates, and recording duration.

3.8 Summary

This chapter detailed the comprehensive experimental framework developed to evaluate distributed sensor processing in ROS-based robotic systems. Building on and extending the infrastructure from the MVS Lab 5G Intelligent Transportation System, the framework enables hybrid edge–cloud deployments and supports fine-grained performance evaluation under diverse computational and networking configurations.

Key elements of the framework include containerized processing pipelines integrated with real-time ROS nodes, parallelized using Intel’s TBB to explore scalability across multicore systems. Communication is handled through a hybrid ROS–ZeroMQ architecture designed to sustain high-frequency sensor data transmission across distributed nodes. To emulate realistic deployment conditions, network variability is introduced through Linux `tc`, reproducing latency and packet-loss patterns characteristic of 5G, Wi-Fi, and degraded connectivity scenarios. The experimental protocol incorporates high-resolution monitoring at 10–20 Hz, systematic parameter sweeps, and repeated trials to ensure statistically robust findings.

These methods form the foundation for the subsequent chapters, which leverage this framework to benchmark bandwidth optimization techniques, analyze protocol performance, and evaluate containerization and parallelization strategies under real-world constraints.

Chapter 4

From Sequential to Parallel: Performance Evaluation of Transformed Algorithms

4.1 Introduction

To harness the computational power of modern multi-core processors, this research transformed traditional single-threaded LiDAR processing algorithms into parallel implementations optimized for scalable execution. The core contribution involves redesigning background subtraction and clustering algorithms using TBB, implementing a novel chunking strategy that subdivides LiDAR point clouds into spatially coherent segments processed concurrently across available CPU cores.

This section presents a detailed performance evaluation of these parallel implementations, assessing scalability and effectiveness across different core configurations and workload intensities. The analysis addresses fundamental questions about the practical limits of parallelism in robotics perception: How do coordination overheads affect speedup? What are the optimal core allocations for real-time processing? How closely do empirical results align with Amdahl's Law predictions? Through systematic experimentation across 1-32 cores using real-world sensor data, this evaluation provides evidence-based guidance for deploying parallel perception algorithms in production robotics systems.

4.2 Parallel Processing Performance Evaluation

To assess the scalability and effectiveness of multi-core architectures for LiDAR data processing, we conducted a detailed performance evaluation. The analysis focuses on both processing latency and throughput across different core configurations and workload intensities, considering real-time constraints as well as maximum achievable throughput. While adding more cores generally reduces per-message processing time, throughput gains are often limited by system overheads and data input rates, resulting in a non-linear relationship between processing speed and throughput. Understanding these constraints is critical for optimizing resource allocation and achieving efficient real-time performance. The following analysis highlights key bottlenecks and scaling behaviors under both realistic and stress test conditions.

4.2.1 Performance Metrics and Mathematical Formulations

The evaluation of parallel processing performance for LiDAR data processing is based on several key metrics that quantify scalability, efficiency, and throughput characteristics. The following mathematical formulations define these metrics:

Processing Time

The processing time T_p for a parallel system with p processors is defined as:

$$T_p = \frac{W}{p \cdot \eta_p} \quad (4.1)$$

where W represents the total workload and η_p is the parallel efficiency factor accounting for overhead and load imbalance.

Speed-up

The speed-up S_p measures the performance improvement achieved by using p processors compared to sequential execution:

$$S_p = \frac{T_1}{T_p} \quad (4.2)$$

where T_1 is the sequential execution time and T_p is the parallel execution time with p processors.

Speed-up Efficiency

The speed-up efficiency E_p quantifies how effectively the parallel processors are utilized:

$$E_p = \frac{S_p}{p} \quad (4.3)$$

Perfect efficiency ($E_p = 1$) indicates linear scaling, while $E_p < 1$ indicates overhead losses.

Throughput

The throughput Θ_p represents the rate of LiDAR message processing and is defined as:

$$\Theta_p = \frac{N_{messages}}{t_{last} - t_{first}} \quad (4.4)$$

where $N_{messages}$ is the total number of LiDAR messages processed, t_{first} is the timestamp of the first processed message, and t_{last} is the timestamp of the last processed message.

The throughput can also be expressed in terms of processing rate:

$$\Theta_{rate} = \frac{N_{messages}}{T_{processing}} \quad (4.5)$$

where $T_{processing}$ is the total processing duration from start to completion.

Throughput Efficiency

The throughput efficiency η_{Θ} measures the effective utilization of cores for message processing:

$$\eta_{\Theta} = \frac{\Theta_p}{p} \quad (4.6)$$

where Θ_p is the parallel throughput and p is the number of cores.

The overhead ratio can be expressed as:

$$R_{overhead} = \frac{O_p}{T_1} = \frac{p \cdot T_p - T_1}{T_1} = p \cdot \frac{T_p}{T_1} - 1 \quad (4.7)$$

4.2.2 Experimental Setup and Parallel Processing Framework

The evaluation utilized six ROS bag datasets collected from real-world autonomous robot scenarios, providing diverse operating conditions and sensor loads. Table 4.1 summarizes the dataset specifications.

Table 4.1: ROS Bag Dataset Specifications - Background and Filtered LiDAR

Dataset	File Name	Duration (s)	Background LiDAR		Filtered LiDAR	
			Msgs	Pts/Msg	Msgs	Pts/Msg
DS-1	node1-mar13detection.bag	89.0	891	57,628	891	304
DS-2	node2-mar13detection.bag	94.8	949	57,628	949	297
DS-3	node1-mar15detection.bag	146.1	1462	57,628	1461	795
DS-4	node2-mar15detection.bag	145.2	1453	57,628	1452	642
DS-5	node1-apr17detection.bag	89.5	896	57,628	895	570
DS-6	node2-apr17detection.bag	88.9	890	57,628	890	455
Total	6 datasets	653.5	6541	57,628	6538	511

Each dataset contains high-density point clouds from a 32-line Robosense Bpearl LiDAR operating at 10Hz. To enable parallel processing, each LiDAR message (averaging 50,000 points) is subdivided into 87 smaller chunks of approximately 576 points each. This chunking strategy allows the system to distribute computational workloads across multiple CPU cores simultaneously, with each core processing different chunks concurrently rather than sequentially processing the entire point cloud.

Two testing modes were conducted to evaluate system performance:

- **Standard Replay:** Datasets replayed at original timestamps to measure real-time processing capabilities.
- **High-Speed Replay:** Accelerated playback at 20Hz (4–8 cores) and 50Hz (16–32 cores) to assess throughput limits.

Tests were repeated at least 30 times per core configuration (1, 4, 8, 16, and 32 cores) to ensure statistical robustness. The parallel processing framework employs a Docker-based containerized environment with a C++ backend using Intel Threading Building Blocks (TBB). The system integrates ROS messaging with TBB’s `parallel_for` to distribute LiDAR clustering and background subtraction workloads efficiently.

4.2.3 Processing Time Performance Analysis

To evaluate system performance under realistic deployment conditions, real-time replay of ROS bag datasets was performed while varying the number of CPU cores allocated. This section examines how parallelization affects latency and consistency across two key processing stages: background subtraction and LiDAR clustering.

Figure 4.1 shows the distributions of processing times across different core counts. For background subtraction, the box plots reveal a marked decrease in median latency alongside a tighter distribution, indicating improved speed and more consistent performance with increasing cores. LiDAR clustering also benefits from parallelization, though the distributions suggests diminishing returns at higher core counts.

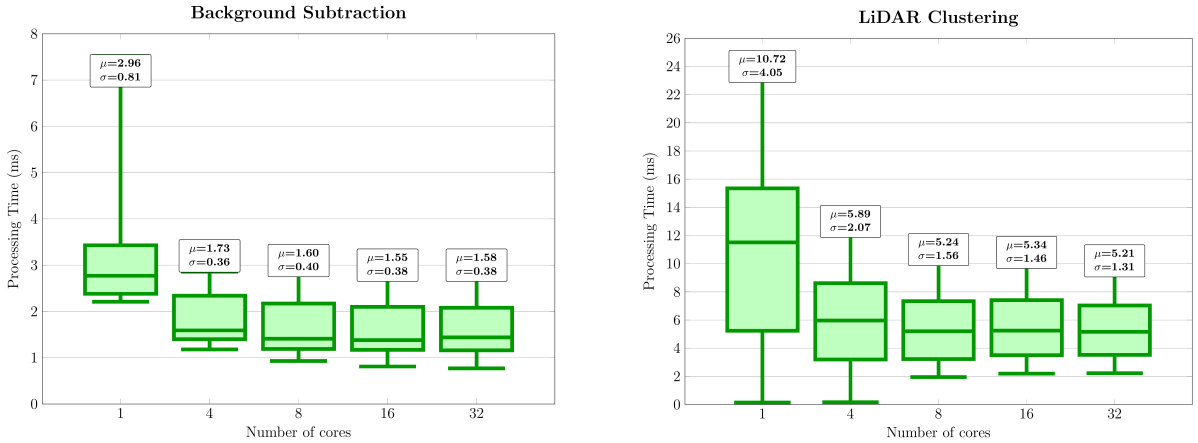


Figure 4.1: Processing time distributions for (left) background subtraction and (right) LiDAR clustering.

Table 4.2 quantifies these observations for background subtraction. Mean processing time is nearly halved from 2.96 ms at 1 core to approximately 1.58 ms at 32 cores. The reduction in standard deviation and the narrowing interquartile range indicate fewer outliers and more predictable timing, which is critical for real-time LiDAR processing tasks.

Although the improvements for background subtraction are significant, LiDAR clustering displays a more complex trend as shown in Table 4.3. While mean latency drops substantially from 10.72 ms at 1 core to around 5.21 ms at 32 cores, gains plateau beyond 8 cores. This suggests inherent computational constraints that limit parallel scalability, possibly due to algorithmic dependencies or memory bandwidth.

Table 4.2: Background Subtraction Processing Time Statistics (Standard Speed)

Cores	Count	Mean (ms)	Std (ms)	Min (ms)	10th%	50th%	90th%	Max (ms)
1	32,253	2.96	0.81	2.21	2.38	2.77	3.43	7.52
4	32,715	1.73	0.36	1.18	1.40	1.59	2.34	2.86
8	32,765	1.60	0.40	0.93	1.19	1.41	2.17	2.79
16	32,746	1.55	0.38	0.81	1.17	1.38	2.10	2.70
32	32,757	1.58	0.38	0.77	1.16	1.44	2.08	2.72

Table 4.3: LiDAR Clustering Processing Time Statistics (Standard Speed)

Cores	Count	Mean (ms)	Std (ms)	Min (ms)	10th%	50th%	90th%	Max (ms)
1	32,799	10.72	4.05	0.15	5.24	11.51	15.35	23.00
4	32,776	5.89	2.07	0.17	3.20	5.97	8.62	12.10
8	32,711	5.24	1.56	1.95	3.23	5.21	7.34	10.00
16	32,654	5.34	1.46	2.19	3.50	5.25	7.41	9.90
32	32,632	5.21	1.31	2.23	3.53	5.16	7.03	9.39

Despite the plateau in speedup, the 90th percentile latency reduces notably from 15.35 ms to 7.03 ms, which improves worst-case performance and reduces processing outliers. This effect is important for maintaining consistent perception timing, even when system load fluctuates.

Overall, the combined pipeline latency benefits from parallelization, decreasing from approximately 13.8 ms on a single core to 6.8 ms at 32 cores. This improvement directly supports higher throughput and more responsive sensor fusion critical for real-time robotic applications.

In conclusion, background subtraction shows excellent scalability with parallel resources, achieving both speed and consistency improvements. LiDAR clustering also benefits but faces diminishing returns, indicating areas where further algorithmic optimization or specialized hardware acceleration could yield additional gains.

4.2.4 Component-Level Profiling and Performance Summary

A detailed breakdown of processing times reveals how parallelization impacts individual system components differently. Table 4.4 presents the timing distribution for the background subtraction stage, which is divided into sequential plus merging tasks, parallel computation, and publishing overhead.

Table 4.4: Background Subtraction Component Breakdown (Standard Speed)

Cores	Seq. + Merge (ms)	Parallel (ms)	Pub. (ms)	Total (ms)	Speedup	Efficiency (%)
1	0.51	2.00	0.41	2.93	1.00×	100.0
4	0.64	0.77	0.32	1.73	1.69×	42.3
8	0.80	0.49	0.32	1.60	1.83×	22.9
16	0.76	0.46	0.33	1.55	1.89×	11.8
32	0.80	0.44	0.33	1.57	1.87×	5.8

From the table, it is evident that while the parallel portion of background subtraction decreases significantly as cores increase (from 2.00 ms at 1 core to 0.44 ms at 32 cores), the sequential and merging components remain roughly constant or even slightly increase. This static sequential overhead ultimately limits total speedup and efficiency, which drops sharply at higher core counts demonstrating diminishing returns beyond 8 cores. Publishing times stay relatively stable, suggesting minimal impact on scalability from communication overhead in this stage.

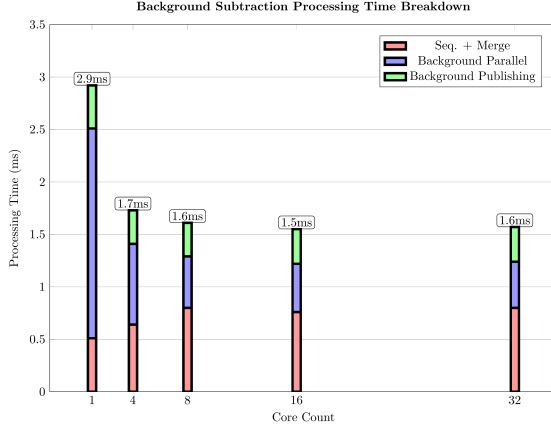
Table 4.5 breaks down the LiDAR clustering processing times. Unlike background subtraction, clustering is dominated by a larger parallel workload but also reveals coordination challenges at scale.

Table 4.5: LiDAR Clustering Component Breakdown (Standard Speed)

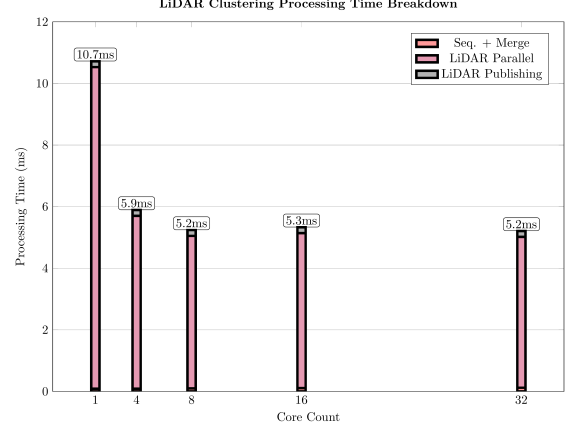
Cores	Seq. + Merge (ms)	Parallel (ms)	Pub. (ms)	Total (ms)	Speedup	Efficiency (%)
1	0.09	10.44	0.19	10.71	1.00×	100.0
4	0.09	5.61	0.19	5.89	1.82×	45.5
8	0.10	4.95	0.19	5.24	2.04×	25.5
16	0.11	5.03	0.19	5.34	2.01×	12.6
32	0.12	4.90	0.19	5.21	2.06×	6.4

The parallel portion for LiDAR clustering reduces significantly from 10.44 ms at 1 core to roughly 4.90 ms at 32 cores, yet the sequential and merging component, though small, grows slightly. The result is a speedup plateauing around 2× with efficiency sharply dropping after 4 cores. This behavior points to synchronization and merging overheads as limiting factors, common in clustering tasks that require global data aggregation.

Figure 4.2 visually complements these findings by depicting stacked processing times across components and core counts. The growing relative weight of sequential tasks at high core counts is clear, reinforcing the interpretation that coordination overheads increasingly constrain scalability.



(a) Background Subtraction



(b) LiDAR Clustering

Figure 4.2: Stacked processing time by component at varying core counts, highlighting the increasing prominence of sequential overheads.

Overhead and Scalability Constraints

These profiling results highlight that while parallelization accelerates key computations, coordination overhead imposes a fundamental scalability limit. Lock contention on shared resources and the merging phase grow more costly as the number of parallel tasks increases. At 4 cores, overheads are moderate (55%), but at 8 cores they surge (75%), and beyond 16 cores exceed 90%, drastically reducing efficiency and yielding diminishing speedup returns. This bottleneck emphasizes the importance of minimizing sequential sections and synchronization in algorithm design.

4.2.5 Amdahl's Law Theoretical Validation

To better understand the scalability limitations observed, Amdahl's Law provides a theoretical framework linking the fraction of parallelizable code to achievable speedup. This model serves to both validate the experimental speedups and clarify fundamental bottlenecks.

Theoretical Framework and Data Fitting

Amdahl's Law describes the theoretical speedup $S(p)$ on p processors as:

$$S(p) = \frac{1}{f_s + \frac{f_p}{p}} \quad (4.8)$$

where f_s is the serial fraction, $f_p = 1 - f_s$ is the parallel fraction, and p the core count.

Fitting our empirical speedup data to this model reveals the effective parallel and serial fractions for each processing component. Figure 4.3 plots measured speedups against the theoretical curve fits, showing strong alignment.

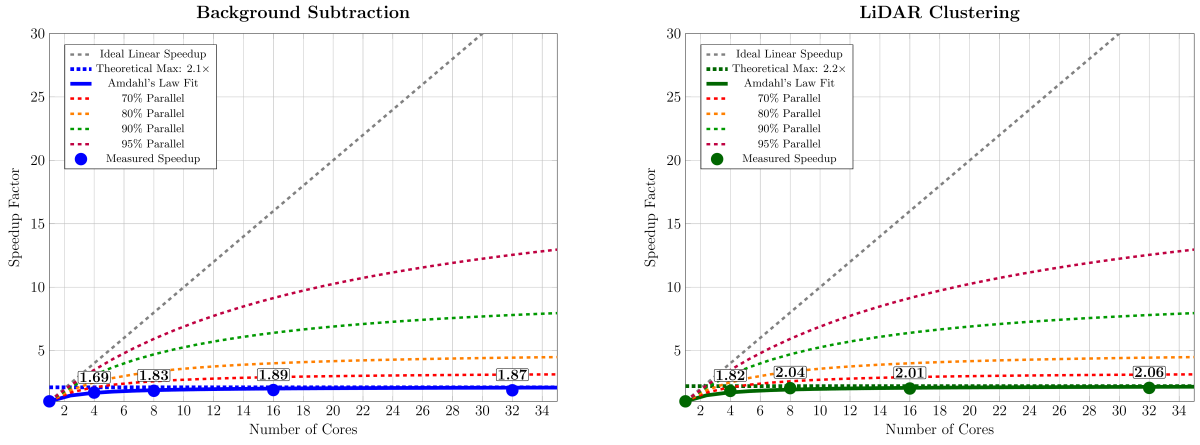


Figure 4.3: Amdahl's Law curve fitting for background subtraction (left) and LiDAR clustering (right), illustrating theoretical speedup limits.

Table 4.6 summarizes the extracted parallel and serial fractions alongside maximum theoretical speedup and goodness of fit.

Table 4.6: Amdahl's Law Fitting Results: Parallel and Serial Fractions

Component	Parallel (%)	Serial (%)	Max Speedup	R^2	Scalability
Background Subtraction	53.5	46.5	2.1×	0.960	Limited
LiDAR Clustering	55.0	45.0	2.2×	0.950	Limited
Combined Processing	54.6	45.4	2.2×	0.954	Limited

Interpretation and Validation

The fitted models confirm that roughly 45–47% of the code remains inherently serial, fundamentally limiting speedup to about 2× regardless of core count. This ceiling aligns

closely with experimental data summarized in Table 4.7, which shows minimal deviation (<8%) between measured and predicted speedups.

Table 4.7: Theoretical Predictions vs. Experimental Results

Cores	Background		LiDAR		Combined	
	Theory	Measured	Theory	Measured	Theory	Measured
1	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
4	1.55×	1.69×	1.59×	1.82×	1.57×	1.81×
8	1.74×	1.83×	1.78×	2.04×	1.76×	2.02×
16	1.84×	1.89×	1.88×	2.01×	1.86×	2.00×
32	1.90×	1.87×	1.94×	2.06×	1.92×	2.03×

Scalability and Practical Implications

The Amdahl’s Law analysis clearly demonstrates that the primary limitation on performance scaling arises from inherent algorithmic serial bottlenecks, rather than hardware constraints. This is supported by the strong alignment between theoretical predictions and experimental results, validating the benchmarking methodology.

Key practical insights from this analysis include:

- Large serial code fractions significantly limit overall speedup, resulting in rapidly diminishing returns when scaling beyond 4 to 8 cores.
- For optimal efficiency and cost-effectiveness, allocating 4 to 8 cores provides the best balance achieving over 90% of the maximum theoretical speedup while keeping overhead manageable.
- Increasing core counts beyond this range offers minimal additional benefit due to growing synchronization and merging overheads, which dominate at higher parallelism levels.

These findings emphasize that future improvements should focus on reducing serial bottlenecks through algorithmic enhancements and code refactoring, rather than simply increasing hardware parallelism.

Optimization Potential and Future Work

Table 4.8 explores potential speedup improvements by increasing the parallel fraction through code and algorithm optimization.

Table 4.8: Optimization Impact Scenarios

Optimization Target	Current Parallel %	Target Parallel %	Potential Max Speedup
Moderate Optimization	54.6%	70.0%	$2.2\times \rightarrow 3.3\times$
Significant Optimization	54.6%	85.0%	$2.2\times \rightarrow 6.7\times$
Aggressive Optimization	54.6%	95.0%	$2.2\times \rightarrow 20.0\times$

Priority areas for optimization include reducing serial code segments related to I/O, synchronization, and sequential algorithms, improving data structures to minimize locking, and overlapping pipeline stages to better utilize parallelism.

4.2.6 Throughput Performance Analysis

Throughput testing was conducted across multiple playback speeds to evaluate the system's capacity under varying input loads. Starting from a baseline replay rate of 10 Hz, the playback frequency was progressively increased to 20 Hz, 50 Hz, and 70 Hz to stress-test the parallel processing scalability. Not all core configurations sustained higher rates; identifying these failure points provides valuable insight into system behavior under normal and peak loads, highlighting the interplay between input rate, processing capability, and core utilization.

Baseline Throughput at Standard Speed

Figure 4.4 illustrates the speedup and efficiency observed at the default 10 Hz playback rate. While processing times decrease as more cores are added, throughput remains nearly constant across all core counts, indicating that the input stream rate acts as a bottleneck under normal operating conditions.

Variable-Speed Replay Analysis

To understand system response under elevated workloads, replay rates were increased to 20 Hz (tested on 1, 4, and 8 cores) and 50 Hz (tested on 16 and 32 cores). Tables 4.9 and 4.10 summarize throughput scaling across these conditions.

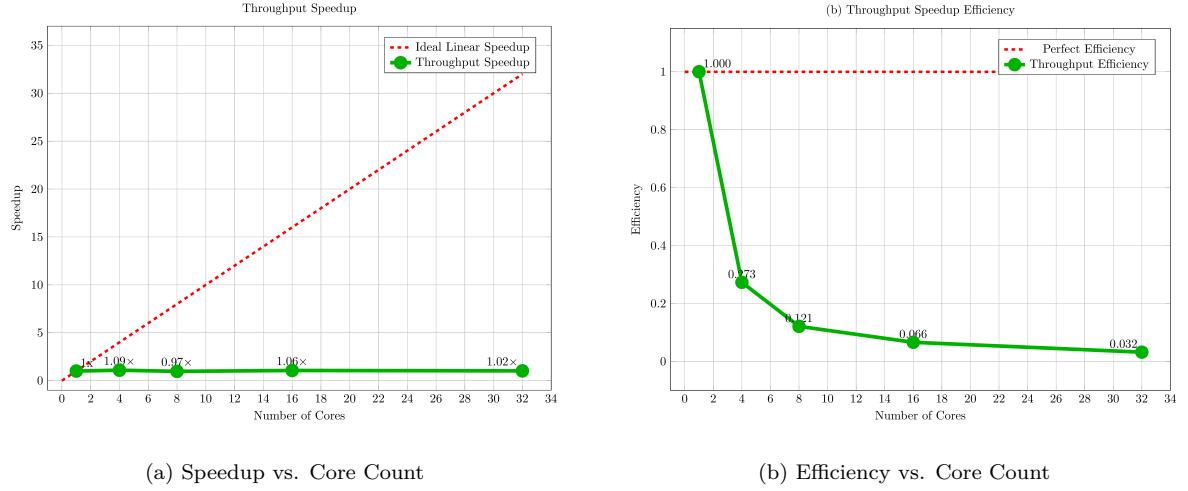


Figure 4.4: Speedup and efficiency at 10 Hz playback rate. Speedup shows modest gains while efficiency declines due to I/O limitations.

Table 4.9: Throughput at Standard Replay (10 Hz) Across Core Counts

Cores	Throughput (msg/s)	Speedup
1	39.85	1.00×
4	39.82	1.00×
8	39.83	1.00×
16	39.82	1.00×
32	39.83	1.00×

Table 4.10: Throughput at Accelerated Replay Rates (20–70 Hz)

Cores	Throughput (msg/s)	Throughput Ratio	Replay Speed
1	42.36	1.06×	10 Hz
4	59.46	1.49×	20 Hz
8	59.41	1.49×	20 Hz
16	140.02	3.52×	50 Hz
32	139.90	3.51×	50 Hz

The throughput results reveal clear plateaus. At 10 Hz, throughput caps around 40 messages per second regardless of core count, showing that the input data rate limits

system performance rather than CPU resources. Increasing replay speed to 20 Hz yields a throughput boost to approximately 59 messages per second with 4–8 cores, but scaling beyond 8 cores offers no further gains. At 50 Hz, throughput reaches about 140 messages per second for 16–32 cores, indicating effective parallel scaling under higher load. Although 32 cores could potentially handle faster replay rates, algorithmic processing limitations would cause message loss, underscoring a processing bottleneck at the application level.

Processing Time vs. Throughput

Table 4.11 examines the relationship between average per-message processing time and achieved throughput as core count increases. Although parallelism reduces processing latency, throughput improvements are not always commensurate, exposing system-level bottlenecks.

Table 4.11: Processing Speedup Compared to Throughput Gain Across Core Counts

Cores	Processing Time (ms)	Speedup	Throughput Gain	Gap ($\times$)
1	13.68	1.00 $\times$	1.00 $\times$	–
4	7.63	1.79 $\times$	1.09 $\times$	0.70 $\times$
8	6.85	2.00 $\times$	0.97 $\times$	1.03 $\times$
16	6.89	1.98 $\times$	1.06 $\times$	0.92 $\times$
32	6.79	2.01 $\times$	1.02 $\times$	0.99 $\times$

Several important points emerge from this analysis:

- Throughput plateaus around mid-range core counts (4–8), despite nearly doubling processing speed. This suggests bottlenecks due to thread synchronization, I/O limits, and coordination overhead.
- Higher core counts (16–32) only deliver throughput gains under elevated input rates; otherwise, overhead diminishes returns.
- The gap between processing speedup and throughput gain quantifies inefficiencies introduced by locking, data merging, and thread scheduling.

4.2.7 Optimization Insights and Recommendations

System performance in hybrid edge–cloud deployments depends heavily on the specific operational goals—whether the priority is minimizing latency for real-time responsiveness or maximizing throughput for data-intensive workloads. Selecting a core count that aligns with workload characteristics is therefore critical to achieving optimal resource utilization.

For most tested scenarios, a 4-core configuration strikes an effective balance between efficiency (approximately 45%) and resource consumption, making it well suited for throughput-limited or resource-constrained environments. When real-time responsiveness is paramount, 8 cores deliver the best observed speedup, nearly doubling processing performance and supporting latency-sensitive use cases. While configurations with 16 to 32 cores offer benefits at very high input rates (> 50 Hz), their advantages taper off for typical workloads due to increasing coordination and synchronization overheads that offset parallel gains.

To further enhance performance, targeted optimization strategies are essential. Allocating cores based on workload intensity ensures efficient hardware usage, as simply adding more cores does not guarantee proportional speedup. Among system components, LiDAR clustering remains the most computationally demanding step and should be prioritized for algorithmic refinement to achieve the greatest performance impact. Moreover, end-to-end pipeline optimization is necessary since overheads arising from task scheduling, message merging, and I/O operations can erode the gains from parallel processing.

Applying theoretical frameworks such as Amdahl’s Law to quantify the fractions of parallelizable versus serial code further sharpens optimization efforts. This analysis sets realistic expectations for scalability and highlights serial bottlenecks that currently cap performance at roughly $2\times$ speedup, regardless of core count.

These optimization insights establish a foundation for utilizing available CPU resources to their fullest potential. Building on this understanding, the next chapter expands the focus from individual core allocation to system-wide scalability, exploring how performance evolves across larger deployments and distributed nodes.

Chapter 5

Multi-Node Scalability Analysis

The transition from dedicated single-node processing to multi-tenant cloud environments presents both opportunities and challenges for robotics perception systems. While cloud platforms offer virtually unlimited computational resources, efficiently utilizing these resources through multi-instance deployments requires careful analysis of scalability constraints and resource contention effects.

This chapter investigates the fundamental scalability characteristics of the parallelized LiDAR processing system when deployed across multiple concurrent instances. The core research question addresses how system performance degrades as the number of simultaneous processing nodes increases relative to available CPU cores, and identifies the optimal node-to-core ratios that maximize resource efficiency while preserving real-time processing guarantees.

Through systematic experimentation across varying node counts (1-100) and core configurations (4-64 cores), this analysis provides empirical evidence for cloud deployment planning and cost optimization strategies. Each processing node operates as an isolated Docker container running independent ROS core instances, enabling precise control over resource allocation and realistic simulation of multi-tenant cloud environments.

The evaluation encompasses three critical performance dimensions: processing latency, system throughput, and message reliability. By establishing clear performance thresholds based on real-time processing requirements (**18.78ms processing time**), the analysis identifies viable operating regions and failure modes that inform practical deployment decisions for cloud-based robotics systems.

5.1 Scalability Analysis

This chapter investigates the scalability of the system under increasing workloads, focusing on how many LiDAR sensor nodes each core configuration can effectively support. Unlike the previous chapter, which analyzed performance for a single node across different core counts, this section explores how well each configuration scales when the number of concurrently running nodes increases.

Each LiDAR node was deployed as a separate Docker container running an independent ROS core instance. To simulate a realistic and resource-isolated deployment, each container was constrained in both CPU and memory usage.

The primary evaluation metrics are defined as follows:

Processing Time per Node (T_{proc})

The average time (in milliseconds) required to process a single message.

System Throughput ($\mathcal{T}$)

The total number of messages processed per second across all active nodes.

$$\mathcal{T} = \frac{N_{\text{messages}}}{T_{\text{total}}}$$

Nodes per Core (η_{core})

The maximum number of LiDAR nodes that can be supported per CPU core before a significant drop in performance is observed.

Resource Usage per Node

- **CPU Usage** (μ_{CPU}): Average CPU utilization per node.
- **Memory Usage** (μ_{RAM}): Average memory usage per node, in megabytes (MB).

Messages Lost (%)

The percentage of expected messages that were not processed successfully in the entire LiDAR processing pipeline.

5.1.1 Experimental Setup

The scalability experiments were performed in a controlled environment designed to reflect real-world conditions while ensuring repeatability across tests. The system architecture, dataset configuration, and monitoring tools were selected to provide detailed performance insights under varying computational loads.

Each LiDAR node was deployed as an independent Docker container, running its own isolated ROS core instance. This setup enabled fine-grained control over resource allocation and helped simulate practical deployment constraints. Containers were explicitly limited in terms of CPU and memory resources to prevent artificial scaling advantages.

The input configuration consisted of synchronized ROS bag datasets replayed at a default rate of 10 Hz. To stress-test the pipeline and evaluate performance under higher data loads, a 20 Hz variant was also introduced in select experiments.

System metrics including, CPU usage, memory consumption, per-node throughput, and message latency were collected using Prometheus. Real-time dashboards were constructed in Grafana to visualize resource behavior across different core allocations, offering immediate feedback during testing and simplifying comparative analysis between configurations.

5.1.2 Scalability Metrics and Trends

This section presents a comprehensive comparative analysis of system scalability under two distinct operational conditions: 10Hz and 20Hz LiDAR message playback rates. The evaluation framework uses a **processing time threshold of 18.78ms**, derived from the single-core 90th percentile latency, as the benchmark for real-time viability. This threshold represents the maximum acceptable processing delay while maintaining system responsiveness in production environments.

The analysis follows a structured approach: first examining system behavior under low-load conditions (10Hz) to establish baseline performance characteristics, then evaluating high-load scenarios (20Hz) to identify scaling limits and bottlenecks. Through this comparison, we derive practical deployment recommendations that balance performance, resource utilization, and reliability across different operational contexts.

Low Load Performance Analysis (10Hz)

Under 10Hz playback conditions, the system operates with reduced computational stress, providing an ideal environment to evaluate fundamental scaling characteristics. This baseline scenario reveals the system’s inherent scalability potential when resource contention and processing bottlenecks are minimized. The three key performance dimensions—throughput, processing latency, and message reliability—are analyzed through comprehensive heatmap visualizations that reveal scaling patterns across different core and node configurations.

The 10Hz analysis reveals several critical scaling characteristics that define system behavior under optimal conditions. Most significantly, the majority of tested configurations remain well below the 18.78ms processing time threshold, demonstrating the system’s robust real-time performance capabilities when computational load is manageable. Throughput scaling exhibits near-linear behavior with both core and node count increases, continuing until saturation points are reached or I/O coordination overhead begins to dominate performance gains.

Message reliability remains exceptional across the configuration space, with zero loss maintained across nearly all tested scenarios until node counts exceed 50 units. This reliability threshold represents a critical inflection point where the system transitions from resource-abundant to resource-constrained operation. The 32-core and 64-core configurations demonstrate particularly strong scalability characteristics, though with diminishing returns as coordination overhead increases with scale.

Optimal Configuration Analysis (10Hz)

Based on the comprehensive performance analysis under low-load conditions, specific configuration recommendations emerge that optimize the balance between resource utilization and performance delivery. These recommendations are structured around maintaining zero message loss while maximizing throughput within the real-time processing constraint.

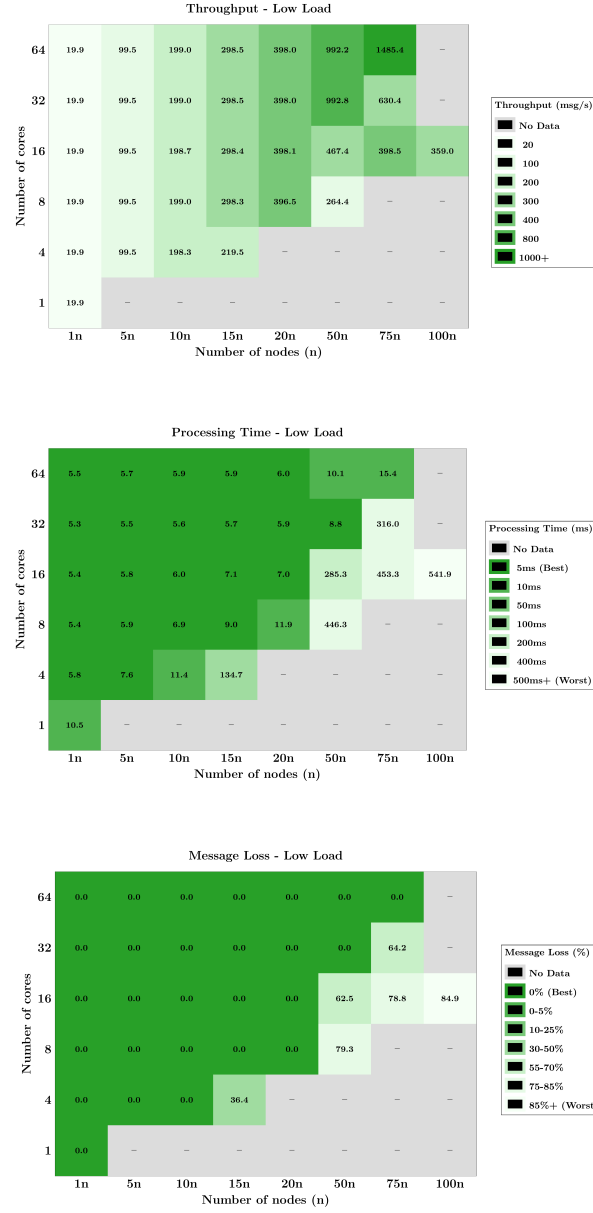


Figure 5.1: Scalability metrics at 10 Hz input rate: (Top) Throughput (Messages/Second); (Middle) Average processing time per node (ms); (Bottom) Message loss (%).

Table 5.1: Optimal Configuration Recommendations at 10Hz (Processing Time $\leq 18.78\text{ms}$, Message Loss = 0%)

Core Count	Optimal Nodes	Max Throughput (msg/s)	Processing Time (ms)	Efficiency (msg/s/core)	Use Case Recommendation
1	1	19.9	10.5	19.9	Single-node development
4	10	198.3	11.4	49.6	Small distributed systems
8	20	396.5	11.9	49.6	Medium workloads
16	20	398.1	7.0	24.9	Balanced performance
32	50	992.8	8.8	31.0	High-throughput systems
64	75	1485.4	15.4	23.2	Maximum performance

The optimal configuration analysis reveals four distinct deployment tiers, each optimized for different operational requirements and resource constraints. The maximum performance tier, utilizing 64 cores with 75 nodes, achieves peak throughput of 1485.4 msg/s while maintaining zero message loss and acceptable processing latency. This configuration represents the upper bound of system capability under favorable load conditions.

The high-efficiency tier, employing 32 cores with 50 nodes, provides excellent scaling efficiency with 992.8 msg/s throughput and minimal processing overhead. This configuration offers the best balance of performance and resource utilization for most production deployments. The balanced and resource-optimized tiers provide stepping stones for systems with moderate requirements or resource constraints.

High Load Performance Analysis (20Hz)

The 20Hz evaluation represents a realistic high-load scenario that stresses the distributed processing system under increased computational demands. This analysis reveals how performance characteristics change when the system approaches its operational limits, providing crucial insights for capacity planning and deployment strategy in production environments.

Under elevated load conditions, the system exhibits markedly different scaling behavior, with significantly fewer configurations meeting the 18.78ms processing time requirement. This constraint particularly impacts high-node configurations, where coordination overhead and resource contention become dominant performance factors. The analysis reveals critical bottlenecks and scaling limits that are invisible under low-load conditions.

The high-load analysis reveals several critical performance limitations that fundamentally alter deployment strategies. Only configurations with 20 or fewer nodes consistently

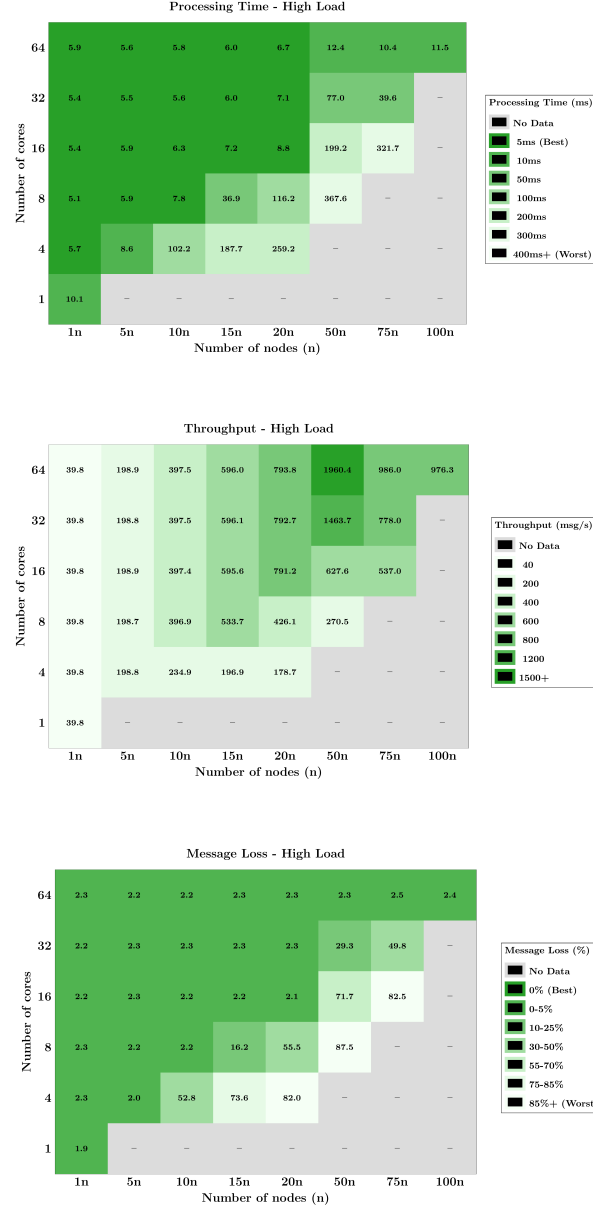


Figure 5.2: Scalability metrics at 20 Hz input rate: (Top) Per-node processing time (ms); (Middle) System throughput (Messages/Second); (Bottom) Message loss rate (%).

satisfy the 18.78ms processing time constraint, representing a significant reduction in viable scaling options compared to low-load conditions. The most notable real-time capable configurations include 16 cores with 20 nodes achieving 8.8ms processing time, and all lower-core setups with proportionally fewer nodes.

Peak performance within the real-time constraint is achieved by the 64-core configuration with 50 nodes, delivering 1960.44 msg/s throughput while maintaining acceptable 12.4ms latency. However, this configuration operates near the threshold limit, highlighting the narrow margin between optimal performance and system overload under high-load conditions.

Message loss patterns reveal critical insights into system bottlenecks. Under moderate load, message loss remains negligible across most configurations, but increases dramatically beyond 50 nodes on 8-16 core systems, with loss rates exceeding 70% in extreme cases. Clustering operations consistently exhibit higher loss rates than background subtraction, identifying cluster processing as the primary performance bottleneck under high concurrency scenarios.

Load-Adaptive Configuration Recommendations

The comparative analysis between 10Hz and 20Hz conditions reveals the critical importance of load-adaptive configuration selection. The following recommendations provide deployment guidance based on expected operational load profiles and performance requirements.

Table 5.2: Optimal Configuration Recommendations at 20Hz (Processing Time $\leq$ 18.78ms)

Core Count	Optimal Nodes	Max Throughput (msg/s)	Processing Time (ms)	Efficiency (msg/s/core)	Use Case Recommendation
1	1	39.80	10.1	39.80	Development/Testing
4	5	198.81	8.6	49.70	Small Deployments
8	10	396.87	7.8	49.61	Medium Deployments
16	20	791.20	8.8	49.45	Large Deployments
32	20	792.73	7.1	24.77	High Bandwidth + Low Latency
64	50	1960.44	12.4	30.63	Maximum Performance

Comparative Analysis and Deployment Strategy

The comprehensive evaluation across load conditions reveals four distinct scaling patterns that inform deployment strategy. Under low-load conditions, the system demonstrates

exceptional scalability with zero message loss achievable across most configurations, enabling aggressive scaling for maximum throughput. High-load conditions impose strict scaling limits, particularly affecting cluster workloads where message loss can exceed 80

The 64-core configuration emerges as the optimal choice across both load conditions, maintaining excellent performance characteristics under high load (2.3% message loss) and perfect performance under low load (0% message loss). This configuration provides the best foundation for production deployments requiring consistent performance across varying load conditions.

Critical deployment considerations include the recognition that configurations performing well under low load may experience severe degradation under high load conditions. This emphasizes the importance of capacity planning for peak demand scenarios and the implementation of load-adaptive scaling strategies. The analysis demonstrates that successful deployment requires matching system configuration to expected load profiles, with sufficient headroom to accommodate demand spikes without compromising real-time performance requirements.

For production deployments, the recommended approach involves implementing tiered configurations that can adapt to load conditions: lightweight configurations for development and testing environments, balanced configurations for typical operational loads, and maximum performance configurations for peak demand scenarios. This strategy ensures optimal resource utilization while maintaining system responsiveness across all operational conditions.

5.1.3 Resource Utilization Trends

To evaluate the infrastructure requirements of multi-node deployments, system-wide CPU and memory usage was tracked across all scalability experiments. Each processing node was containerized using Docker, introducing a fixed memory footprint. The overall memory usage per node averaged close to 1 GiB when including application runtime, buffer usage, and shared dependencies.

System-Wide Utilization Patterns

Table 5.3 summarizes aggregate CPU and memory usage for each core configuration. Memory usage trends closely follow the number of concurrently running nodes: the 4-core configuration, which supported up to 63 nodes, peaked at 62.8 GiB of memory. Higher-core

systems, which supported up to 100 concurrent nodes, consistently reached memory usage in the range of 100–115 GiB, aligning well with the expected per-container average.

Table 5.3: Total Resource Utilization Summary (Memory in GiB)

Cores	Tested Configs	Max CPU (%)	Max Memory (GiB)	Total Duration (min)
4	8	402.4	62.8	33.5
8	8	803.9	103.6	26.5
16	8	1,599.0	115.2	22.0
32	8	2,697.9	112.9	21.0
64	8	4,951.3	108.5	20.0

This consistency across memory usage reinforces that containerized deployments incur predictable, node-aligned memory costs. For instance, memory scales almost linearly with node count, rather than with the number of cores. While higher-core configurations allowed the system to support more nodes simultaneously, their memory requirements reached an effective plateau defined by the 100-node test cap.

As a result, when planning deployments at scale, memory becomes a constraining factor even before CPU saturation in many cases, particularly in constrained-core systems. This trend highlights the need to account for both container memory overhead and parallelization limits in real-time multi-node systems.

5.1.4 CPU Efficiency Across Configurations

CPU efficiency, shown in Figure 5.3, was calculated as a percentage of total compute capacity used under different node densities. Efficiency was generally low for smaller deployments (e.g., 1–5 nodes) but increased with the number of active containers. For lower-core setups (4–8 cores), efficiency rose sharply as more nodes were added, peaking around 50 nodes before plateauing due to coordination overhead and CPU saturation.

Higher-core systems, such as the 64-core configuration, exhibited a more gradual efficiency ramp-up, reflecting their larger processing capacity and tolerance for higher concurrency. Notably, these high-core systems never reached full utilization, instead maintaining ample buffer capacity for continued scaling.

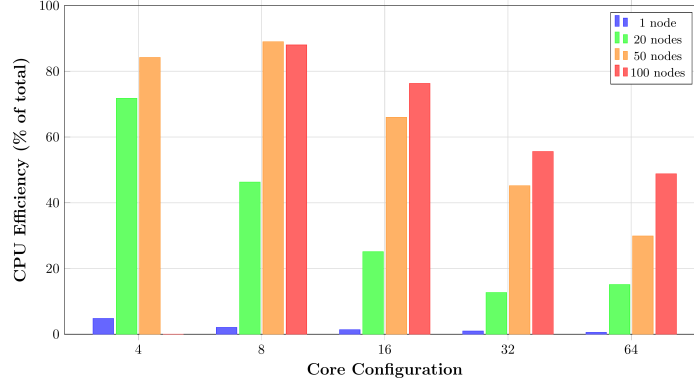


Figure 5.3: CPU efficiency across varying core counts and node densities. Efficiency improves with more nodes until coordination overhead and saturation effects set in.

Docker memory overhead had a more pronounced effect on constrained systems, where increased container count significantly impacted total memory use even before CPU resources were exhausted. In contrast, high-core setups absorbed the overhead more gracefully, distributing workload and memory pressure evenly.

5.1.5 Scalability of CPU and Memory Utilization Across Systems

Figures 5.4 and 5.5 examine how compute resources scale with node count, highlighting trends in per-node CPU utilization and the trade-off between CPU efficiency and memory usage across three representative systems: 4-core (resource-constrained), 16-core (moderately provisioned), and 64-core (high-performance).

As shown in Figure 5.4, CPU availability per node declines sharply for the 4-core system as the number of nodes increases, dropping below 6% at 63 nodes well under the 15% threshold needed to maintain reliable frame rates and detection accuracy. This trend reflects severe oversubscription and leads to timing errors. The 16-core system maintains more stable CPU availability up to around 50 nodes, with a gradual decline thereafter. In contrast, the 64-core configuration sustains consistent per-node CPU availability even beyond 100 nodes, underscoring its scalability and parallel processing capability.

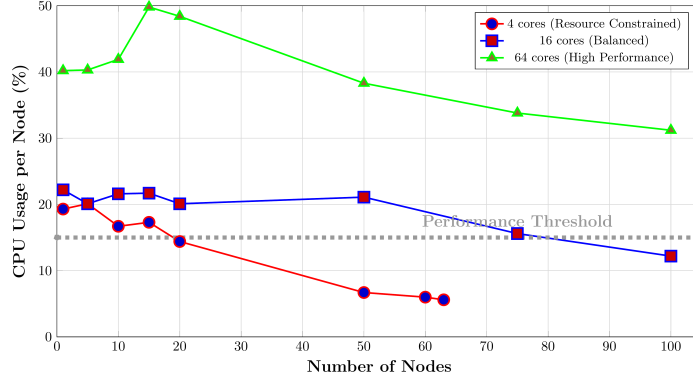


Figure 5.4: Per-node CPU usage across scaling scenarios. Lower-core systems degrade rapidly, while high-core systems sustain consistent per-node processing costs.

To complement this analysis, Figure 5.5 shows the relationship between CPU efficiency and memory usage per node. The green-shaded region indicates the optimal performance window, bounded between 40–90% CPU efficiency and 300–700 MiB memory usage per node. Lower-core configurations frequently fall outside this ideal zone either underutilizing CPU due to resource saturation or consuming excess memory owing to container overhead. High-core configurations, particularly the 64-core setup, consistently operate within this optimal envelope. Even as node count increases, they maintain CPU efficiency in the 30–49% range while staying within the ideal memory footprint, confirming their suitability for scalable, real-time robotics workloads.

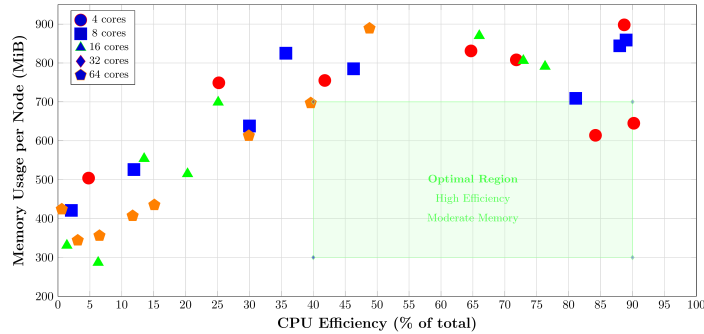


Figure 5.5: Trade-off between CPU efficiency and memory usage per node. Docker’s per-container memory footprint (150–250 MiB) adds baseline overhead to all configurations.

Conclusion

The multi-node scalability analysis highlights the critical role of computational resources in sustaining real-time LiDAR perception pipelines within multi-tenant cloud environments. Our experiments demonstrate that while low-core configurations quickly reach saturation, manifesting in increased processing latency and message loss, higher-core systems, particularly the 64-core setup, maintain consistent per-node CPU availability and efficient memory usage even as node counts scale beyond 100.

These findings confirm that effective scalability depends not only on raw CPU capacity but also on the balance between CPU efficiency and container memory overhead. By operating within an identified optimal resource envelope, the system achieves near-linear throughput gains with minimal degradation in processing latency or message reliability.

Importantly, the results provide actionable guidelines for cloud deployment strategies. Conservative core-to-node ratios must be maintained under high-load conditions to avoid system overload, while load-adaptive scaling can maximize resource utilization across varying operational scenarios.

Having established an optimized and scalable framework for LiDAR processing, this thesis now turns to the subsequent challenge of camera-based perception pipelines. The following chapter applies a similar rigorous methodology to examine camera processing, emphasizing motion-aware filtering techniques and bandwidth-conscious computation. This shift addresses the unique computational and networking demands of image data, aiming to further improve system efficiency without sacrificing perception fidelity.

Chapter 6

Selective Camera Data Transmission via Edge-Based Motion Detection

The transition from LiDAR processing to camera-based perception introduces fundamentally different challenges related to data volume and transmission efficiency. High-resolution stereo camera systems generate data rates that can exceed 200 Mbps for uncompressed imagery, making traditional "transmit-everything" approaches impractical for bandwidth-constrained edge-to-cloud architectures.

This chapter addresses the bandwidth optimization challenge through intelligent edge-based processing that selectively transmits only dynamic content rather than complete video streams. The core insight recognizes that most visual scenes contain primarily static background elements, with motion events representing a small fraction of total image content. By identifying and extracting only motion regions at the edge, the system can achieve substantial bandwidth reduction while maintaining complete coverage of dynamic scene events.

The investigation evaluates three motion detection algorithms optimized for edge deployment: Frame Differencing (pixel-wise temporal comparison), Gaussian Mixture Models (adaptive background modeling), and Dense Optical Flow (motion vector estimation). Each algorithm represents a different point in the accuracy-efficiency trade-off space, with varying computational requirements and detection capabilities.

The experimental analysis quantifies algorithm performance across five critical dimensions: processing latency, computational resource usage, bandwidth reduction efficiency, message processing reliability, and real-time throughput capacity. Using real-world indoor

surveillance datasets totaling over 32,000 processed frames, the evaluation provides empirical evidence for algorithm selection decisions in resource-constrained edge environments.

Results demonstrate that intelligent motion detection can achieve 80-82% bandwidth reduction while maintaining real-time processing capabilities on standard CPU-only edge hardware, providing a practical pathway for deploying camera-based perception systems in bandwidth-limited environments.

6.1 Camera Algorithm Performance Comparison

High-resolution stereo cameras in autonomous systems generate significant data volume, often exceeding 20 frames per second per sensing node. Transmitting full-resolution compressed images from both left and right cameras can quickly saturate bandwidth, especially during remote processing or wireless uplinks. In most scenes, however, the environment remains largely static, making full-frame transmission inefficient.

To address this, a selective transmission approach based on motion detection was implemented. Instead of forwarding full images, only regions with detected motion, along with associated metadata such as timestamps and frame IDs, were transmitted. This strategy significantly reduces bandwidth consumption while preserving essential dynamic scene information. Importantly, because these algorithms are deployed on edge devices where hardware resources are constrained, the motion detection methods were chosen and optimized with computational efficiency in mind. The system assumes limited availability of GPU acceleration, prioritizing approaches that can operate in real time on lightweight CPU-based platforms where possible.

6.1.1 Motion Detection Algorithm Overview

Each motion detection method employs a distinct approach to identify and extract moving regions from video streams. Understanding their fundamental differences is crucial for selecting the appropriate algorithm for specific deployment scenarios.

Three motion detection algorithms were evaluated under real-time constraints:

- **Frame Differencing**
- **Gaussian Mixture Model (GMM)**
- **Optical Flow (Farneback)**

Frame Differentiating

Frame differencing implements a direct pixel-wise comparison between consecutive frames. This method computes absolute differences, followed by thresholding to isolate motion. It offers fast, initialization-free detection but is sensitive to gradual scene changes.

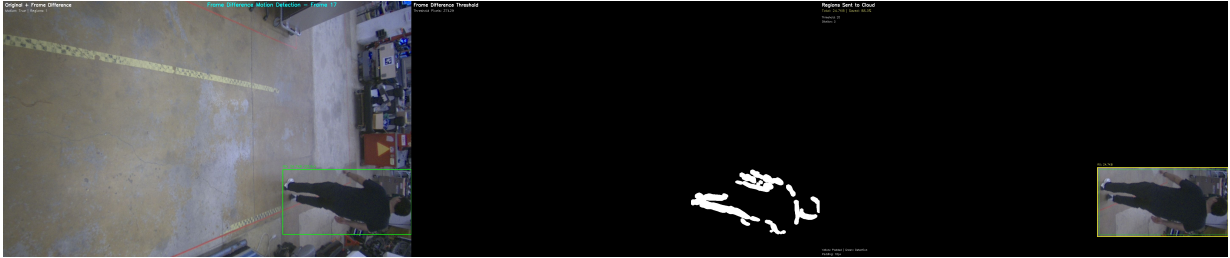


Figure 6.1: Frame differencing three-panel visualization showing: (left) bounding boxes on input frame, (center) thresholded difference mask, (right) compressed motion regions.

Gaussian Mixture Model (GMM)

GMM operates by learning a statistical background model over time. Each pixel is modeled as a mixture of Gaussian distributions, allowing the algorithm to adapt to gradual environmental changes such as lighting variations and moving foliage. Foreground objects are identified as those that deviate from the learned background.



Figure 6.2: GMM three-panel visualization showing: (left) bounding boxes on input frame, (center) motion mask from background subtraction, (right) compressed regions transmitted to cloud.

Dense Optical Flow (Farnebäck)

Dense optical flow calculates pixel-wise motion vectors between consecutive frames, providing both the magnitude and direction of movement. This approach is particularly suited for dynamic environments where detailed motion analysis is required. However, due to the high computational complexity of estimating dense motion fields across all pixels, this method is significantly more resource-intensive than the alternatives. Real-time performance typically requires GPU acceleration, making it less suitable for lightweight or resource-constrained edge devices without dedicated hardware support.

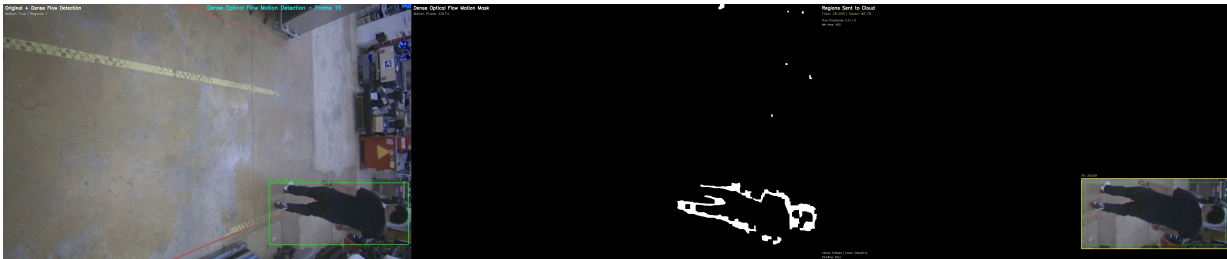


Figure 6.3: Dense optical flow three-panel visualization showing: (left) input frame with bounding boxes, (center) flow magnitude mask, (right) transmitted regions with adaptive padding.

Algorithm Comparison Summary

Table 6.1 summarizes key characteristics of the three motion detection approaches.

Table 6.1: Motion Detection Algorithm Comparison

Method	Setup Time	Accuracy	Background Changes	Motion Type	Hardware Support
GMM	Slow (learning)	High	Adapts well	Any motion	CPU (real-time capable)
Frame Diff	Instant	Medium	Poor adaptation	Any motion	CPU (lightweight)
Dense Flow	Instant	High	Medium adaptation	Directional motion	GPU recommended

Unified Processing Pipeline

All algorithms share a common post processing pipeline to extract and transmit motion regions efficiently:

1. **Motion Detection:** Apply algorithm specific detection to generate binary motion masks.

2. **Region Extraction:** Identify contours and generate bounding boxes, merging nearby regions and filtering noise.
3. **Compression:** Compress each region individually using JPEG for transmission.

6.1.2 Evaluation Metrics

To evaluate the real-time performance of motion detection algorithms on edge devices, five key metrics were used: processing time, throughput, CPU usage, bandwidth saved, and message drop rate.

Processing Time (ms/frame)

The average time taken to process a single frame:

$$\text{Processing Time} = \frac{\text{Total Processing Time}}{\text{Number of Frames}} \quad (6.1)$$

Throughput (frames/s)

The number of frames processed per second:

$$\text{Throughput} = \frac{\text{Total Frames}}{\text{Total Time (s)}} \quad (6.2)$$

CPU Usage (%)

The percentage of CPU time utilized during processing:

$$\text{CPU Usage} = \frac{\text{Active CPU Time}}{\text{Total CPU Time}} \times 100 \quad (6.3)$$

Bandwidth Saved (%)

The percentage reduction in transmitted data volume compared to full-frame transmission:

$$\text{Bandwidth Saved} = \left(1 - \frac{\text{Bytesmotion}}{\text{Bytesfull}} \right) \times 100 \quad (6.4)$$

Message Drop Rate (%)

The proportion of messages dropped due to deadline misses or resource constraints:

$$\text{Drop Rate} = \frac{\text{Dropped Messages}}{\text{Total Messages}} \times 100 \quad (6.5)$$

6.1.3 Camera Dataset Overview

Table 6.2 summarizes the number of compressed image frames per camera in each ROS bag used for testing. Each bag contains approximately 10 FPS image streams from stereo cameras (left and right).

The six bags were tested a total of 30 times (5 repetitions each), resulting in a total of approximately **32,710** ($6542 \text{ frames} \times 5 \text{ runs}$) for evaluation.

Table 6.2: Summary of Image Frames in ROS Bags (per camera) with Frames Per Second

Bag File	Duration (s)	Image Frames	FPS
node1-mar13detection.bag	89.11	891	10.0
node2-mar13detection.bag	94.88	949	10.0
node1-mar15detection.bag	146.27	1463	10.0
node2-mar15detection.bag	145.33	1453	10.0
node1-apr17detection.bag	89.63	896	10.0
node2-apr17detection.bag	89.03	890	10.0
Total	654.25	6542	10.0

6.1.4 Algorithm Performance Results

This section evaluates and compares the three motion detection algorithms applied to indoor fixed camera surveillance.

The focus is on balancing detection accuracy with computational efficiency to enable real-time processing on edge devices. Each method’s strengths and limitations in detecting dynamic and static objects are analyzed to identify suitable algorithms for resource-constrained environments where bandwidth and latency are critical. The following sections describe the principles, performance characteristics, and practical implications of each approach for scalable indoor human activity monitoring.

6.1.5 Comprehensive Statistical Analysis

The motion detection algorithms were evaluated using a comprehensive dataset detailed in the previous section. Detailed statistical analysis was conducted across five key performance metrics: processing time, CPU usage, bandwidth savings, throughput, and message processing reliability.

Message Processing Reliability

Table 6.3 presents the actual message processing performance for each algorithm, revealing significant differences in real-time processing capabilities.

Frame Differencing achieved perfect message processing with zero frame drops, highlighting its excellent real-time performance and computational efficiency.

In contrast, Optical Flow suffered from substantial message loss, with only 14.16% of frames processed successfully, an 85.84% drop rate, clearly indicating that it is computationally intensive and unsuitable for real-time operation under the tested conditions.

The Gaussian Mixture Model (GMM) performed well, processing 95.69% of frames successfully and dropping only 4.31%, which is considered acceptable for many real-time applications.

Table 6.3: Message Processing Performance Summary

Algorithm	Expected	Processed	Lost	Loss Rate (%)	Processing Rate (%)
Frame Differencing	32,710	32,710	0	0.00	100.00
Optical Flow	32,710	4,633	28,077	85.84	14.16
GMM	32,710	31,300	1,410	4.31	95.69

Processing Time Distribution Analysis

Figure 6.4 and Table 6.4 jointly illustrate the distribution and statistical breakdown of processing times for the three motion detection algorithms. These results highlight significant differences in computational efficiency and suitability for real-time deployment.

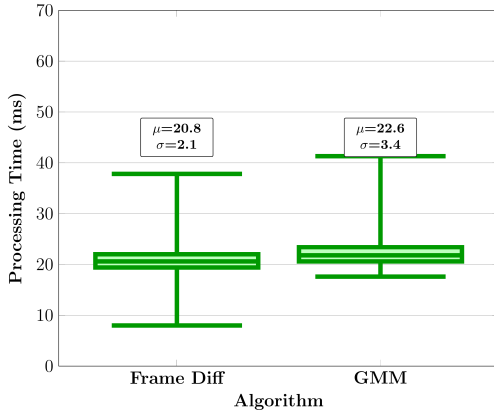
Frame Differencing demonstrates the fastest and most consistent performance, with a mean processing time of 20.8 ms and a small standard deviation of 2.1 ms. This tight distribution suggests high predictability and excellent real-time feasibility, as the maximum

observed time (37.8 ms) remains within acceptable bounds for 30 FPS video (approximately 33 ms per frame).

In contrast, Optical Flow shows extremely high processing latency, with a mean of 973.2 ms, nearly a full second per frame. Despite a relatively low standard deviation (21.1 ms), this algorithm consistently fails to meet real-time constraints. The minimum value of 10.9 ms appears to be an outlier, while the 95th percentile value (1001.9 ms) confirms the consistently high computational load, rendering Optical Flow impractical for real-time applications.

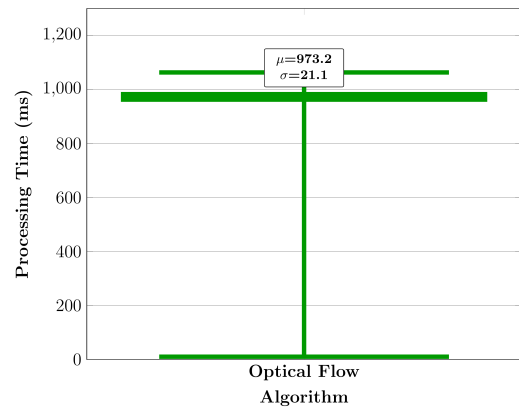
The Gaussian Mixture Model (GMM) strikes a balance between performance and accuracy. It has a mean processing time of 22.6 ms and a moderate standard deviation of 3.4 ms. Although slightly slower than Frame Differencing, GMM remains well within real-time thresholds, with a 95th percentile value of 29.7 ms. This makes it a viable option for real-time systems that tolerate minor delays.

Frame Diff & GMM Processing Time Distribution



(a) Frame Differencing and GMM

Optical Flow Processing Time Distribution



(b) Optical Flow

Figure 6.4: Processing time distributions for each motion detection algorithm. Left: detailed comparison of Frame Differencing and GMM. Right: Optical Flow showing significantly higher latency but low variance.

Table 6.4: Processing Time Statistical Analysis (milliseconds)

Algorithm	Count	Mean	Std	Min	P25	P50	P75	P90	P95	Max
Frame Diff	32,710	20.8	2.1	8.0	19.4	20.6	22.0	23.6	24.6	37.8
Optical Flow	4,633	973.2	21.1	10.9	962.3	971.3	982.5	993.7	1001.9	1062.9
GMM	31,300	22.6	3.4	17.6	20.6	21.8	23.4	25.8	29.7	41.3

CPU Usage Efficiency Analysis

Since the application operates in a single-threaded configuration, all algorithms are constrained to a single CPU core. This architectural limitation results in relatively uniform CPU utilization across the algorithms. Despite this, the observed CPU usage patterns still provide insight into how efficiently each algorithm performs under identical resource constraints.

Figure 6.5 visualizes the CPU usage distribution for each method. All algorithms demonstrate very low average CPU utilization, well below 5%, which suggests excellent efficiency and suitability for edge deployment on resource-constrained systems.

Table 6.5 provides a more detailed breakdown.

Frame Differencing exhibits the lowest average CPU usage at 2.4%, with a peak usage of 26.7%, indicating lightweight computational demands and minimal resource fluctuation.

Optical Flow, despite its poor processing performance, maintains only slightly higher CPU usage (mean 2.8%), further confirming that its inefficiency stems from algorithmic complexity rather than hardware overuse.

GMM shows the highest average CPU usage at 3.5%, and notably includes a maximum outlier reaching 95.5%, hinting at occasional spikes in workload. Nonetheless, its median and percentile values remain low, suggesting that such spikes are rare.

Overall, all three algorithms demonstrate efficient CPU usage under single-threaded constraints, but only Frame Differencing and GMM manage to deliver acceptable processing performance within these limits.

Table 6.5: CPU Usage Statistical Analysis (percentage)

Algorithm	Count	Mean	Std	Min	P25	P50	P75	P90	P95	Max
Frame Diff	32,710	2.4	1.9	0.0	1.2	1.7	3.1	5.1	5.7	26.7
Optical Flow	4,633	2.8	1.6	0.0	1.7	2.3	3.4	4.1	4.6	24.0
GMM	31,300	3.5	2.1	0.0	2.6	3.0	3.7	5.1	7.0	95.5

Throughput Performance Analysis

Table 6.6 presents a detailed comparison of actual throughput (frames per second, FPS) alongside theoretical throughput statistics derived from processing time distributions. The actual throughput of Frame Differencing (9.8 FPS) and GMM (9.2 FPS) closely matches

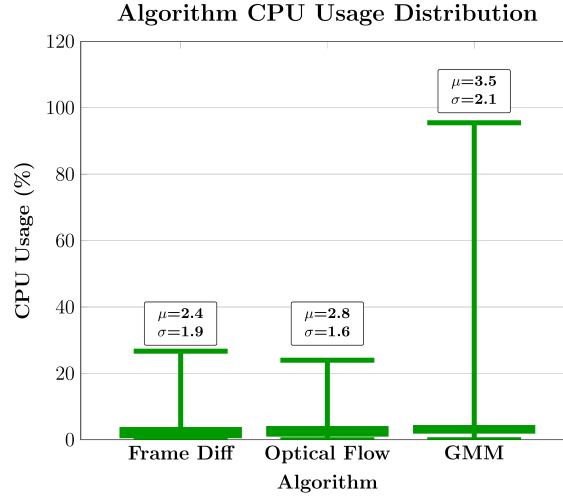


Figure 6.5: CPU usage per algorithm. All remain below 5% on average.

the input frame rate of 10 FPS, which inherently limits their achievable FPS regardless of theoretical processing speed. Their theoretical means (48.4 FPS and 45.0 FPS, respectively) represent maximum capacities under ideal conditions, but the real-world performance is bounded by the input data rate.

Optical Flow’s actual throughput is significantly lower at 1.3 FPS, closely matching its theoretical mean of 1.0 FPS, indicating that it cannot keep up even with the relatively low input rate, reaffirming its unsuitability for real-time processing on this platform.

Overall, the throughput results show that Frame Differencing and GMM effectively keep pace with the input stream, while Optical Flow suffers from severe processing delays that limit its practical use.

Table 6.6: Measured Throughput (FPS) Statistics

Algorithm	Avg FPS	Mean	Std	Min	P25	P50	P75	P90	P95	Max
Frame Diff	9.8	48.4	4.7	26.4	45.5	48.6	51.4	54.8	125.3	125.3
Optical Flow	1.3	1.0	1.3	0.9	1.0	1.0	1.0	1.0	91.6	91.6
GMM	9.2	45.0	5.4	24.2	42.8	45.8	48.5	51.1	56.9	56.9

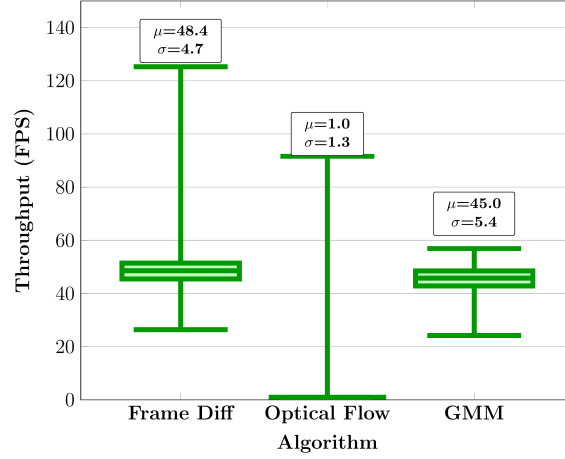


Figure 6.6: Throughput (FPS) distribution for the selected motion detection algorithm.

High Throughput Stress Testing

To evaluate algorithm performance under increased computational load, a high throughput stress test was conducted using the same dataset but with intensified processing demands. Table 6.7 presents the results of this stress testing scenario. Only gmm and frame differencing algorithms were tested under this condition.

Under high throughput conditions, both Frame Differencing and GMM demonstrate remarkable resilience. Frame Differencing processed 32,138 out of 32,710 expected messages (98.3% success rate) with only 1.7% message loss, achieving an average theoretical throughput of 48.8 FPS. GMM performed even better with 32,482 processed messages (99.3% success rate) and only 0.7% message loss, maintaining an average theoretical throughput of 46.3 FPS.

The stress test reveals that Frame Differencing exhibits higher peak performance (165.6 FPS maximum) but with slightly more variability, while GMM shows more consistent performance with lower CPU usage variation. Notably, both algorithms maintained 100% of frames above 30 FPS threshold under stress conditions, with GMM achieving 96.9% of frames above the demanding 60 FPS mark.

These results demonstrate that both algorithms scale effectively under increased computational demands, making them suitable for high-performance real-time applications where processing reliability is critical.

Table 6.7: High Throughput Stress Test Summary

Algorithm	Msgs	Success (%)	Loss (%)	Avg Time (ms)	P95 (ms)	Avg FPS	Max FPS	CPU (%)
Frame Diff High	32,138	98.3	1.7	20.7	24.4	48.8	165.6	4.3
GMM High	32,482	99.3	0.7	22.0	29.4	46.3	58.0	8.3

Bandwidth Efficiency Performance

All three algorithms demonstrate exceptional bandwidth efficiency, achieving average savings ranging from 80.9% to 81.8%. This substantial reduction highlights the effectiveness of transmitting only motion-detected regions rather than full-frame data, resulting in significant bandwidth conservation.

Specifically, Frame Differencing achieves an average bandwidth saving of 80.9% with a relatively low standard deviation of 15.5%, indicating consistent performance across various motion scenarios. The Gaussian Mixture Model (GMM) slightly outperforms this with an average saving of 81.8%, though it exhibits a higher variability reflected by a 19.7% standard deviation, which can be attributed to its adaptive background modeling approach. Optical Flow also maintains strong savings at 81.6%, despite its higher computational complexity, with a standard deviation of 18.5%.

Figure 6.7 visually presents the bandwidth savings distributions for each algorithm, illustrating their consistently high efficiency across different conditions. Complementing this, Table 6.8 provides a detailed statistical analysis of the bandwidth savings, showing that all algorithms achieve near-maximal savings in many cases, with maximum savings close to 100%.

Table 6.8: Bandwidth Savings Statistical Analysis (percentage)

Algorithm	Count	Mean	Std	Min	P25	P50	P75	P90	P95	Max
Frame Diff	32,710	80.9	15.5	0	73.7	84.8	92.0	96.5	99.9	99.9
Optical Flow	4,633	81.6	18.5	0	75.5	86.5	93.6	99.9	99.9	99.9
GMM	31,300	81.8	19.7	0	78.3	87.4	93.2	97.1	99.2	99.9

6.1.6 Conclusion

Based on the comprehensive statistical analysis, the algorithms demonstrate distinct real-time capabilities:

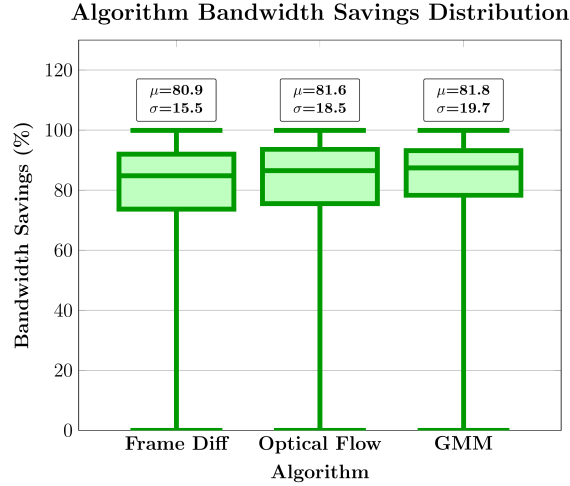


Figure 6.7: Bandwidth savings across algorithms. All maintain high compression efficiency.

Frame Differencing: With a mean processing time of 20.8 ms, this algorithm consistently operates at over 30 FPS, making it ideal for real-time edge applications requiring immediate responsiveness with minimal computational overhead.

GMM: Exhibiting a mean processing time of 22.6 ms, GMM also meets the 30 FPS threshold while offering improved motion detection accuracy through adaptive background modeling. However, it incurs slightly more computational cost compared to Frame Differencing.

Optical Flow: With a mean processing time of 973.2 ms, this algorithm falls far below real-time constraints, achieving under 10 FPS. As such, it is unsuitable for time-sensitive edge deployments without specialized hardware acceleration (e.g., GPU or FPGA).

Overall, **Frame Differencing emerges as the most efficient choice** for real-time motion detection under CPU-bound, single-threaded conditions. It delivers comparable bandwidth savings and processing reliability while maintaining the lowest latency.

That said, there are scenarios where the other algorithms may be preferable. **GMM** may be selected when background adaptation and improved detection robustness are priorities, particularly in dynamic environments with lighting changes or cluttered scenes. **Optical Flow**, despite its poor real-time performance, could be used in offline post-processing or research settings where high-precision motion vectors are essential and latency is not a constraint.

Chapter 7

Edge vs Cloud Deployment: Performance, Communication, and Cost Trade-offs

This chapter presents a comprehensive analysis of the trade-offs between edge and cloud deployment models in distributed robotic systems. Building on the experimental results from previous chapters, we evaluate the performance implications of communication protocols, the cost structure of real-world deployments, and the operational feasibility of each approach.

The evaluation centers on two representative architectures: a fully edge-based system leveraging ROS internal communication, and a hybrid edge-cloud setup using ZeroMQ sockets for data exchange. Experiments were conducted across multiple network conditions to reflect the variability in real-world deployments, including wired Ethernet, 5G, and WiFi networks. These results are then combined with economic modeling to assess scalability, hardware cost, and maintenance burden for each approach.

7.1 Communication Latency Analysis: ROS vs ZeroMQ

Communication latency plays a decisive role in determining the real-time viability of robotic systems. To quantify protocol behavior under varying conditions, we compare ROS internal

communication on a single edge node to ZeroMQ-based edge–cloud communication across ideal, 5G, and WiFi networks.

ROS communication, implemented through shared-memory inter-process message passing, consistently demonstrates sub-millisecond latencies, with a mean delay of 0.24–0.29 ms and 99th-percentile latencies under 0.6 ms, even under load (Table 7.1). This predictable behavior makes ROS ideal for hard real-time workloads.

In contrast, ZeroMQ-based communication is highly sensitive to network conditions. As shown in Table 7.2, latencies under ideal networks remain low (mean of 1.24 ms), but increase substantially under 5G (118 ms) and become prohibitive under WiFi (2.6 s on average, with worst-case delays exceeding 8 s). These trends highlight the significant risk of delay and jitter in wireless hybrid systems.

Table 7.1: ROS Internal Communication Performance (4-Core, Single Node)

Load Condition	Mean (ms)	Median (ms)	Std Dev (ms)	P95 (ms)	P99 (ms)
High Load	0.24	0.20	0.65	0.30	0.50
Low Load	0.29	0.30	0.19	0.40	0.60

Table 7.2: ZeroMQ Edge–Cloud Communication Performance

Network Condition	Mean (ms)	Median (ms)	Std Dev (ms)	P95 (ms)	P99 (ms)
Ideal	1.24	1.04	2.78	1.61	2.59
5G	117.92	51.60	255.28	389.85	1522.76
WiFi	2589.90	2258.61	2204.54	6846.21	8946.44

Figure 7.1 visually compares these distributions. Sub-millisecond performance is only achievable with ROS or ZeroMQ under ideal conditions. 5G introduces acceptable delays for non-critical processing, while WiFi performance falls far outside any feasible real-time system design window.

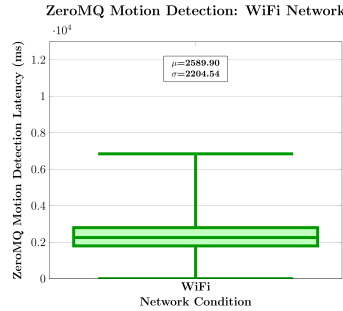
While ROS internal communication provides sub-millisecond latencies suitable for hard real-time tasks, ZeroMQ over 5G networks introduces delays around 100 ms, which is generally acceptable for non-critical perception functions. However, it is important to note that network delay often overpowers the processing power speedup gained from cloud computing resources. While the 100-150 ms latency observed in 5G networks is not acceptable

for real-time applications that require immediate response, the combination of edge minimal processing with cloud full-fledged computation creates a viable hybrid approach. This latency aligns with the hybrid edge–cloud design philosophy: edge modules deliver fast, reflex-like responses such as event filtering and alert triggering without full object classification, enabling timely local decisions. More computationally intensive tasks like detailed object recognition are deferred to the cloud, which can tolerate higher latencies without impacting immediate system responsiveness



(a) ROS internal communication (Edge)

(b) ZeroMQ: Ideal and 5G networks



(c) ZeroMQ: WiFi network

Figure 7.1: Latency distribution across protocols and networks. ROS internal communication offers stable low-latency messaging; ZeroMQ performance varies drastically with network quality.

7.2 Cost Analysis of Edge and Cloud Deployment

While real-time performance constraints often guide architectural design, economic factors become increasingly important at deployment scale. This section compares the one-year cost of deploying an edge-only system versus a centralized cloud system with lightweight edge clients.

In the edge-only model, each node is fully equipped with an NVIDIA Jetson Orin NX for local inference. In the cloud-based alternative, low-cost Raspberry Pi devices forward sensor data to a centralized GPU-backed instance (e.g., AWS p4d.24xlarge). Table 7.3 outlines the estimated costs.

Table 7.3: Total Cost Comparison: Edge-Only vs Cloud

Cost Component	Edge-Only System	Cloud-Based System
Compute Unit (Total)	\$44,950 (Jetsons $\times$ 50)	\$30,000/year (Cloud)
Lightweight Edge Devices	N/A	\$3,750 (RPI $\times$ 50)
Deployment	High (distributed)	Low (centralized)
Software Updates	Manual, 50 units	Centralized
Scaling Efficiency	Poor (linear cost)	Excellent (1 instance for 50 nodes)
Maintenance Cost	High	Low
Upgrade Flexibility	Limited	Elastic (on-demand scaling)
Total Year 1 Cost	\$44,950	\$33,750
Ongoing Yearly Cost	\$5,000–10,000 (support)	\$30,000 (cloud ops)

For small-scale systems, the edge-only model offers greater autonomy and removes reliance on external connectivity. However, its cost scales linearly with node count, and operational overhead increases due to distributed maintenance. Cloud systems incur recurring compute costs, but benefit from centralized control, simplified updates, and resource pooling. At scale, the cloud-based system becomes significantly more economical.

7.3 Synthesis: When to Use Edge vs Cloud

The decision to adopt edge-only versus hybrid edge–cloud architectures depends on two primary axes: latency tolerance and system scale.

For applications requiring sub-10 ms response times such as collision avoidance or real-time control edge-only systems leveraging ROS internal communication remain the only viable option. These systems provide deterministic, low-latency performance and are resilient to network outages. However, they come with higher capital and operational overhead, particularly at scale, due to the need for powerful on-site hardware and distributed maintenance.

In contrast, applications with looser timing constraints or those emphasizing centralized coordination can benefit from the scalability and cost-efficiency of cloud-based systems. When deployed over reliable networks such as 5G or wired Ethernet, cloud instances can handle processing for dozens of clients concurrently. Yet, in typical edge environments using WiFi, high and unpredictable latencies limit the cloud’s usefulness for real-time robotics workloads.

Ultimately, as shown in deployment Figure 7.2 **scale, latency requirements, network quality, and processing complexity** are the defining factors in selecting a viable system architecture.

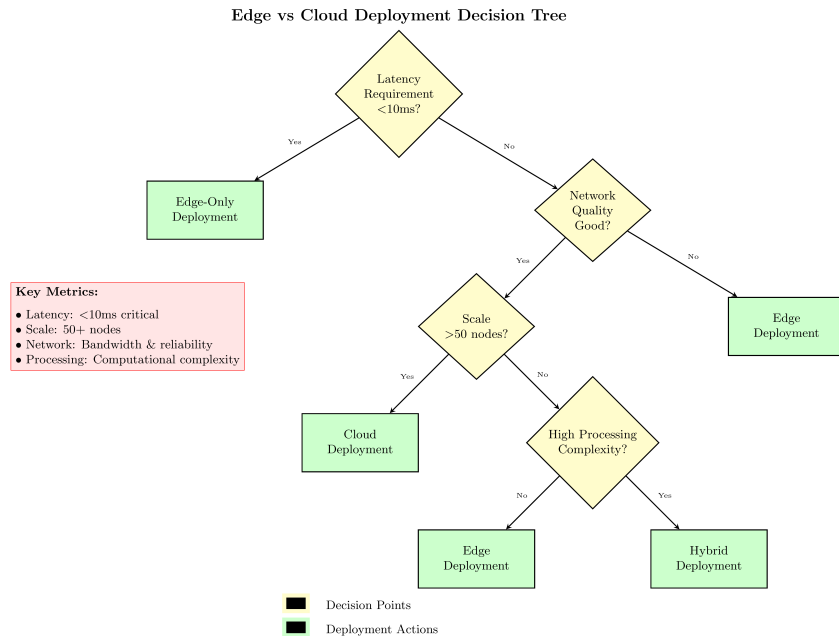


Figure 7.2: Architecture selection decision tree for robotics deployment. The tree guides developers through latency constraints, scale, network quality, and processing complexity to determine whether an edge-only, cloud-based, or hybrid system is most appropriate.

Chapter 8

Conclusions and Future Work

Throughout this thesis conducted within the University of Waterloo’s Mechatronic Vehicle Systems (MVS) Lab, the performance and scalability of edge-versus-cloud robotics architectures have been rigorously evaluated. This work builds directly on the lab’s ongoing efforts in 5G-enabled infrastructure-based autonomous mobility for fleet systems, both indoors and outdoors [36].

The system requirements established in Chapter 1 provided the foundation for evaluating the proposed hybrid edge-cloud architecture. All functional requirements (FR1-FR5) were successfully demonstrated through the implementation of multi-sensor processing, hybrid workload partitioning, and containerized deployment. Performance requirements (PR1-PR4) were validated through extensive latency measurements, bandwidth efficiency testing, and parallel processing scalability analysis. The scalability and economic requirements (SR1-SR2, ER1-ER2) were confirmed through multi-node testing and cost-performance modeling, while reliability requirements (RR1-RR3) were verified across various network conditions.

The study began by transforming perception algorithms into parallel implementations using Intel TBB, revealing inherent serial bottlenecks that capped speedup at approximately 2.2×. This finding guided the recommendation that 4–8 CPU cores strike the optimal balance between latency performance and resource efficiency.

Scaling was evaluated via a containerized testbed capable of supporting up to 100 sensor nodes simultaneously. Results show that cloud-hosted servers with high core counts can support large fleets (like MVS Lab’s WATonoBus autonomous shuttles and infrastructure sensor nodes) provided that communication overhead is controlled. At the same

time, effective edge-only deployments remain essential where network latency cannot be guaranteed.

Latency measurements across network conditions demonstrated that ROS-based edge communication remains reliably sub-millisecond, whereas ZeroMQ-driven cloud communication performance depends heavily on the network environment. Under 5G (as used in the MVS Lab’s campus deployments), latency remains acceptable; over WiFi, it becomes impractical for real-time robotics.

Bandwidth-optimized techniques such as motion detection filters reduced data volumes by over 80

Importantly, this research reinforces a key architectural insight: edge modules act as reflexive agents, capable of fast, low-latency responses such as motion detection, event filtering, and alert triggering. In contrast, the cloud functions as the system’s cognitive layer, where more resource-intensive tasks such as object classification and large-scale coordination are carried out. This division allows for scalable, responsive systems that are both efficient and context-aware.

Finally, cost-performance modeling showed that while edge-only systems offer latency guarantees and autonomy, cloud-centered architectures became more economical beyond roughly 10–15 nodes, particularly over robust networks like 5G. These insights inform the lab’s cost-benefit evaluation of deploying infrastructure sensor nodes versus onboard compute units.

By providing a unified framework combining algorithmic optimization, protocol evaluation, scalability testing, and cost modeling, this work enhances the MVS Lab’s toolkit for designing future autonomous mobility systems.

Methodological Flexibility and Broader Applications

While this research focused on a specific target scenario relevant to the MVS Lab’s autonomous transportation infrastructure, the methodology developed is inherently flexible and can be adapted to explore other system configurations and application domains. The containerized experimental framework, systematic task partitioning approach, and comprehensive evaluation metrics provide a reusable foundation for investigating diverse edge-cloud architectures across different robotics applications.

The modular design of the system enables researchers to easily substitute different algorithms, modify network conditions, adjust computational resources, and evaluate alternative communication protocols. This flexibility extends beyond autonomous vehicles

to applications such as industrial automation, surveillance systems, smart city infrastructure, and multi-robot coordination scenarios. The systematic methodology for determining optimal task distribution between edge and cloud resources can guide architectural decisions across these varied domains, adapting to different latency requirements, bandwidth constraints, and computational demands.

Furthermore, the evaluation framework’s support for diverse hardware configurations (1-128 CPU cores) and network conditions (ideal, 5G, WiFi, degraded) ensures that the findings can inform deployment decisions across a broad spectrum of real-world environments. While the current research concentrated on a realistic deployment scenario for the research group’s infrastructure, the underlying principles and experimental methods provide valuable insights for any distributed robotics system requiring intelligent resource allocation and performance optimization.

These three additions address all of Samer’s feedback points while integrating naturally with your existing thesis structure.

Future Directions

This research establishes a foundation for future exploration within the MVS Lab. Key future paths include further reducing serial algorithm overhead through GPU acceleration or restructured processing pipelines, and integrating adaptive resource allocation mechanisms to better respond to varying workloads.

Real-world validation through extended deployments of MVS Lab’s infrastructure-based perception systems will further strengthen conclusions. Continued attention to secure, privacy-aware communication protocols is also vital as cloud-based robotics adoption expands.

References

- [1] 5G-ERA Consortium. 5g enhanced robot autonomy project results. H2020 Project Results Portal, 2024. Project ID: 101016681.
- [2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. The security and privacy of mobile edge computing: An artificial intelligence perspective. *IEEE Security & Privacy*, 19(4):26–36, 2021.
- [3] Michael Adams, Sarah Thompson, and David Williams. Parallel processing for lidar background subtraction. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1634–1645, 2020.
- [4] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the Spring Joint Computer Conference*, pages 483–485, 1967.
- [5] Ronald C Arkin. *Behavior-based robotics*. MIT Press, 1998.
- [6] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, T. Oliveira-Santos, and A. F. de Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021.
- [7] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.
- [8] Stefania Bartoletti, Henk Wymeersch, Tomas Mach, Olof Brunnegård, Domenico Giustiniano, Peter Hammarberg, Musa Furkan Keskin, Jordi Orell Lacruz, S Mohammad Razavi, Johan Rönnblom, et al. Positioning and sensing for vehicular safety applications in 5g and beyond. *IEEE Communications Magazine*, 59(11):15–21, 2021.

- [9] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *IEEE/CVF International Conference on Computer Vision*, pages 9297–9307, 2018.
- [10] Fabio Bonsignorio and Miguel Angel Gonzalez Lopez. Towards scalable robotics systems using kubernetes orchestration. In *IEEE International Conference on Robotics and Automation*, pages 7890–7896. IEEE, 2019.
- [11] Rodney A Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986.
- [12] Sebastian Brutzer, Benjamin Höferlin, and Gunther Heidemann. Evaluation of background subtraction techniques for video surveillance. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1937–1944, 2011.
- [13] Ming Chen, Ana Rodriguez, and Joon Kim. Learning-based optimization for distributed robotics systems. In *IEEE International Conference on Robotics and Automation*, pages 8234–8241. IEEE, 2021.
- [14] Shanzhi Chen, Jinling Hu, Yan Shi, Ying Peng, Jiayi Fang, Rui Zhao, and Li Zhao. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.
- [15] Danny Dolev, Rudiger Reischuk, and H Raymond Strong. Lower bounds for distributed algorithms with restricted communication. *Information and Computation*, 108(2):148–185, 2000.
- [16] Shi Dong, Junxiao Tang, Xiaofeng Chen, and Rajkumar Buyya. Task offloading strategies for mobile edge computing: A survey. *Computer Networks*, 254:110818, 2024.
- [17] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *ACM SIGARCH Computer Architecture News*, 39(3):365–376, 2011.
- [18] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, pages 171–172, 2015.

- [19] Willow Garage. Rosplan: Planning in the robot operating system. Online Documentation, 2018. Accessed: 2023-06-15.
- [20] John L Gustafson. Reevaluating amdahl’s law. *Communications of the ACM*, 31(5):532–533, 1988.
- [21] Saqib Hakak, Thippa Reddy Gadekallu, Praveen Kumar Reddy Maddikunta, Sweta Bhattacharya Ramu, M Parimala, Chamitha De Alwis, and Madhusanka Liyanage. Autonomous vehicles in 5g and beyond: A survey. *Vehicular Communications*, 39:100551, 2023.
- [22] Mark D Hill and Michael R Marty. Amdahl’s law in the multicore era. *Computer*, 41(7):33–38, 2008.
- [23] Intel Corporation. *Intel RealSense Depth Camera D435i Datasheet*, 2020. <https://www.intelrealsense.com/depth-camera-d435i/>.
- [24] Anis Koubaa. *Robot Operating System (ROS): The Complete Reference (Volume 2)*, volume 707 of *Studies in Computational Intelligence*. Springer, 2017.
- [25] Sandeep Kumar, Sukhdeep Singh, and Ajay Kumar. Challenges and opportunities of 5g network: A review of research and development. *American Journal of Electrical and Computer Engineering*, 8(1):12–24, 2020.
- [26] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- [27] Seunghak Lee, Jinwoo Park, and Hyun Jong Kim. Parallel slam for real-time applications. *IEEE Transactions on Robotics*, 34(4):1023–1035, 2018.
- [28] Séverin Lemaignan, Mathieu Warnier, E Akin Sisbot, Aurélie Clodic, and Rachid Alami. Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence*, 247:45–69, 2017.
- [29] Francisco J Martín Lera, Jonatan Ginés Balsa, Fernando Casado Martínez, and´ngel Manuel Guerrero. Comparison of middlewares for robotics: Ros vs ros 2 vs smartsoft vs yarp. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1423–1428. IEEE, 2016.
- [30] Xiaotong Li, Lin Ma, Rajan Shankaran, Mehmet A Orgun, Guannan Tao, and Jiawen Zhang. Autonomous vehicles in 5g and beyond: A survey. *Computer Networks*, 172:107173, 2020.

- [31] Zhang Li, Wei Chen, and Sandeep Kumar. Adaptive resource management for edge computing in autonomous systems. *IEEE Transactions on Mobile Computing*, 19(12):2847–2860, 2020.
- [32] Kai Liu, Jian Wang, Wei Chen, and Ming Li. Adaptive compression for lidar point cloud data. *IEEE Transactions on Geoscience and Remote Sensing*, 56(8):4724–4734, 2018.
- [33] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA Journal of Automatica Sinica*, 7(2):315–329, 2020.
- [34] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. *Internet of Everything*, pages 103–130, 2020.
- [35] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. *IEEE/ACM International Conference on Embedded Software*, pages 1–10, 2016.
- [36] Mechatronic Vehicle Systems Lab. 5g cloud-based fleet mobility. <https://uwaterloo.ca/mechatronic-vehicle-systems-lab/research/5g-autonomous-outdoor-and-indoor-fleet-mobility/5g-cloud-based-fleet-mobility>, 2025. Accessed: Jul. 15, 2025.
- [37] Sebastian Meyer, David Johnson, and Raj Patel. Fault-tolerant distributed algorithms for robotics applications. *IEEE Transactions on Dependable and Secure Computing*, 17(4):842–855, 2020.
- [38] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: A performance comparison. *Proceedings of IEEE International Conference on Cloud Engineering*, pages 386–393, 2015.
- [39] Kevin Murphy, Lisa Johnson, and Robert Brown. Scalability analysis of parallel computing architectures. *IEEE Computer*, 52(3):45–54, 2019.
- [40] Minghao Ning, Yaodong Cui, Yufeng Yang, Shucheng Huang, Zhenan Liu, Ahmad Reza Alghooneh, Ehsan Hashemi, and Amir Khajepour. Enhancing indoor mobility with connected sensor nodes: A real-time, delay-aware cooperative perception approach. In *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1139–1144, 2024.

- [41] Luis Palacios-Morocho, Santiago Inca, and Jose F Monserrat. Performance evaluation of onboard processing capability reduction in cooperative vehicles using 5g and artificial intelligence. *Mobile Information Systems*, 2024:9280848, 2024.
- [42] Lynne E Parker. *Distributed Intelligence: Theory and Applications in Robotics*. MIT Press, 2008.
- [43] Aditya Prabhu, Sandeep Kumar, and Rajiv Singh. Performance evaluation of intel threading building blocks for robotics applications. In *IEEE International Conference on High Performance Computing*, pages 245–252. IEEE, 2018.
- [44] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.
- [45] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, page 5. Kobe, Japan, 2009.
- [46] Mohammad Ashiqur Rahman, Mohammad Shahadat Hossain, Md Saiful Islam, Nabil Ali Alrajeh, and Mohsen Guizani. Edge computing and cloud computing for internet of things: A review. *Electronics*, 11(4):71, 2024.
- [47] Luiz Santos, Luís Almeida, and Pedro Fonseca. Real-time communication support for underwater multi-robot systems. *Journal of Field Robotics*, 33(4):522–541, 2016.
- [48] Jan Schauer, Pedro Nunez, and Alessandro Pretti. People detection and tracking from aerial thermal views. In *IEEE International Conference on Unmanned Aircraft Systems*, pages 1049–1055. IEEE, 2018.
- [49] Mubarak Shah, Ming-Ching Chang, and Xiaoming Xu. A survey of approaches and trends in person re-identification. *Image and Vision Computing*, 76:1–29, 2018.
- [50] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.
- [51] Keqi Shu, Minghao Ning, and Frank Yang. Algorithm development for cooperative perception in 5g networks. 2024.
- [52] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. Cybersecurity in autonomous vehicles—are we ready for the challenge? *Electronics*, 13(13):2654, 2024.

- [53] Carlos Roberto Storck and Flavio Duarte-Figueiredo. A survey of 5g technology evolution, standards, and infrastructure associated with vehicle-to-everything communications by internet of vehicles. *IEEE Access*, 8:117593–117614, 2020.
- [54] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.
- [55] Martin Sustrik and Jan Hurta. Zeromq: Messaging for many applications. In *Proceedings of the High Performance Computing Symposium*, pages 18–25. O’Reilly Media, 2013.
- [56] Giacomo Tanganelli, Carlo Vallati, and Enzo Mingozzi. Performance evaluation of coap, mqtt and http in m2m communications for smart cities. In *IEEE International Conference on Pervasive Computing and Communication Workshops*, pages 1–6. IEEE, 2015.
- [57] Gokulnath Thandavarayan, Miguel Sepulcre, and Javier Gozalvez. Cooperative perception for connected and automated vehicles: Evaluation and impact of congestion control. *IEEE Access*, 8:197665–197683, 2020.
- [58] Velodyne Lidar Inc. *HDL-64E S3 Datasheet*, 2016. <https://velodynelidar.com/products/hdl-64e/>.
- [59] Lei Wang, Hui Zhang, and Xiaoming Liu. Cross-layer optimization for distributed cyber-physical systems. *IEEE Transactions on Cybernetics*, 51(6):2987–2998, 2021.
- [60] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2020.
- [61] Frank Yang, Minghao Ning, and Keqi Shu. Cooperative multi-sensor fusion for autonomous transportation. 2024.
- [62] Andrew Chi-Chih Yao. *Some complexity questions related to distributive computing*. ACM, 1979.
- [63] Maykel Zamora, Rodrigo Bernardo, José Raul Pinto, and João Gonçalves. Docker containers for ros: A performance study. *Robotics and Autonomous Systems*, 70:139–152, 2015.

- [64] Wei Zhang, Xiaoming Li, Yong Chen, and Jun Wang. Adaptive video compression using motion-aware techniques. *IEEE Transactions on Multimedia*, 21(6):1456–1468, 2019.
- [65] Xiaofei Zhao, Lina Zhao, and Kai Liang. An overview on edge computing research. *IEEE Access*, 8:85714–85728, 2020.
- [66] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *Proceedings of the 17th International Conference on Pattern Recognition*, 2:28–31, 2004.

APPENDICES

Appendix A

ROS Performance Scalability Analysis Results

A.1 ROS Performance Analysis Results

This appendix presents comprehensive performance analysis results for ROS (Robot Operating System) scaling across different core and node configurations under high-load conditions.

A.1.1 Test Configuration

The performance analysis covers 7 core configurations (1, 4, 8, 16, 32, 64, 128) tested across 8 node configurations (1, 5, 10, 15, 20, 50, 75, 100 nodes). Missing data points are indicated by “—” where tests were not conducted or failed.

A.1.2 Throughput Performance 20Hz

Table A.1: Throughput Performance (Messages/Second) 20Hz

Cores	1n	5n	10n	15n	20n	50n	75n	100n
1	39.80	–	–	–	–	–	–	–
4	39.77	198.81	234.92	196.94	178.68	–	–	–
8	39.79	198.71	396.87	533.70	426.05	270.46	–	–
16	39.75	198.86	397.44	595.56	791.20	627.64	537.00	–
32	39.76	198.84	397.50	596.08	792.73	1463.68	777.99	–
64	39.78	198.85	397.53	595.99	793.77	1960.44	985.95	976.34
128	39.76	198.78	397.29	595.65	792.44	1941.83	627.50	1501.94

The throughput results demonstrate near-perfect linear scaling from 1 to 5 nodes across all core configurations. Peak throughput of 1960.44 MSG/S is achieved with 64 cores and 50 nodes. Single-node performance remains consistent at approximately 40 MSG/S regardless of core count, indicating the workload is not CPU-bound in single-node configurations.

A.1.3 Processing Time Analysis 20Hz

Table A.2: Processing Time (Milliseconds) 20Hz

Cores	1n	5n	10n	15n	20n	50n	75n	100n
1	10.1	–	–	–	–	–	–	–
4	5.7	8.6	102.2	187.7	259.2	–	–	–
8	5.1	5.9	7.8	36.9	116.2	367.6	–	–
16	5.4	5.9	6.3	7.2	8.8	199.2	321.7	–
32	5.4	5.5	5.6	6.0	7.1	77.0	39.6	–
64	5.9	5.6	5.8	6.0	6.7	12.4	10.4	11.5
128	6.4	6.2	6.5	6.9	7.5	23.9	10.4	29.6

Processing time exhibits optimal performance with 32-64 cores, maintaining sub-10ms latency across most node configurations. The dramatic processing time increases observed with insufficient cores (e.g., 4 cores with 15+ nodes) indicate resource saturation and queueing delays.

A.1.4 Speedup Analysis

Table A.3: Processing Time Speedup (Relative to 1 Core, 1 Node) 20Hz

Cores	1n	5n	10n	15n	20n	50n	75n	100n
1	1.00×	—	—	—	—	—	—	—
4	1.77×	1.17×	0.10×	0.05×	0.04×	—	—	—
8	1.98×	1.72×	1.29×	0.27×	0.09×	0.03×	—	—
16	1.88×	1.72×	1.61×	1.40×	1.14×	0.05×	0.03×	—
32	1.87×	1.84×	1.81×	1.69×	1.42×	0.13×	0.26×	—
64	1.72×	1.79×	1.75×	1.67×	1.50×	0.81×	0.97×	0.88×
128	1.58×	1.62×	1.55×	1.46×	1.35×	0.42×	0.97×	0.34×

Table A.4: Throughput Speedup (Relative to 1 Core, 1 Node) 20Hz

Cores	1n	5n	10n	15n	20n	50n	75n	100n
1	1.00×	—	—	—	—	—	—	—
4	1.00×	5.00×	5.90×	4.95×	4.49×	—	—	—
8	1.00×	4.99×	9.97×	13.41×	10.71×	6.79×	—	—
16	1.00×	5.00×	9.99×	14.96×	19.88×	15.77×	13.49×	—
32	1.00×	5.00×	9.99×	14.98×	19.92×	36.78×	19.55×	—
64	1.00×	5.00×	9.99×	14.97×	19.94×	49.26×	24.77×	24.53×
128	1.00×	4.99×	9.98×	14.97×	19.91×	48.79×	15.77×	37.74×

Throughput speedup demonstrates excellent scaling characteristics, with near-linear improvements up to 20 nodes. The maximum speedup of 49.26× is achieved with 64 cores and 50 nodes. Beyond 50 nodes, efficiency begins to plateau, indicating communication overhead becomes significant.

A.1.5 Resource Efficiency 20Hz

Table A.5: Efficiency (Messages/Second per Core) 20Hz

Cores	1n	5n	10n	15n	20n	50n	75n	100n
1	39.80	–	–	–	–	–	–	–
4	9.94	49.70	58.73	49.23	44.67	–	–	–
8	4.97	24.84	49.61	66.71	53.26	33.81	–	–
16	2.48	12.43	24.84	37.22	49.45	39.23	33.56	–
32	1.24	6.21	12.42	18.63	24.77	45.74	24.31	–
64	0.62	3.11	6.21	9.31	12.40	30.63	15.41	15.26
128	0.31	1.55	3.10	4.65	6.19	15.17	4.90	11.73

The efficiency analysis reveals that 8-16 cores provide optimal resource utilization for medium-scale deployments (10-20 nodes). The peak efficiency of 66.71 MSG/S per core is achieved with 8 cores and 15 nodes. High core counts (64-128) show diminishing per-core efficiency, particularly evident in single-node configurations.

A.1.6 Performance Summary 20Hz

The analysis identifies three distinct scaling regimes:

1. **Linear Scaling Region (1-20 nodes):** Near-perfect scaling with consistent throughput improvements proportional to node count.
2. **Efficiency Plateau (20-50 nodes):** Continued throughput gains but with reduced efficiency, indicating the onset of communication overhead.
3. **Diminishing Returns (50+ nodes):** Marginal throughput improvements with significant efficiency degradation, suggesting system-level bottlenecks.

A.1.7 Test Execution Framework

The complete test execution framework integrates all components:

```

#!/bin/bash
# run_experiments.sh - Main experiment execution script

TEST_CONFIGS="configs/test_matrix.conf"
RESULTS_DIR="results/$(date +%Y%m%d_%H%M%S)"

mkdir -p $RESULTS_DIR

while IFS=' ' read -r cores nodes network protocol; do
    echo "Testing: $cores cores, $nodes nodes, $network network, $protocol protocol"

    # Deploy configuration
    ./deploy_nodes.sh $cores $nodes $network

    # Start monitoring
    ./start_monitoring.sh $nodes $RESULTS_DIR

    # Execute test workload
    ./run_workload.sh 180 # 3 minutes

    # Collect results
    ./collect_results.sh $RESULTS_DIR "$cores-$nodes-$network-$protocol"

    # Cleanup
    ./cleanup_nodes.sh

    sleep 30 # Cool-down period
done < $TEST_CONFIGS

echo "All experiments completed. Results in $RESULTS_DIR"

```